

CUBRID

Version 11.4

목차

매뉴얼 소개	9
• 매뉴얼 구성	
• 용어 정리	
• 매뉴얼 규약	
• 버전명 및 버전 문자열 규약	
CUBRID 소개	12
• 시스템 구조	
• CUBRID의 특징	
설치 및 업그레이드	24
• 설치와 실행	24
• 환경 변수 설정	40
• 포트 설정	
• 업그레이드	56
• HA 환경에서 DB 마이그레이션	
• CUBRID 제거	67
시작하기	69
• CUBRID 서비스 시작	69
• 질의 도구	72
• 관리 도구	
• 드라이버	
CSQL 인터프리터	78
• CSQL 인터프리터 소개	
• CSQL 실행	
• 세션 명령어	
CUBRID SQL	116
• 작성 규칙	117
• 데이터 타입	133
• 데이터 정의문	210
• 연산자와 함수	328
• 데이터 조작문	730

• 질의 최적화	819
• 병렬 실행	962
• 분할	989
• 다국어 지원	1026
• 트랜잭션과 잠금	1090
• 트리거	1149
• CUBRID DBLink	1167
• 클래스 상속	1191
• 데이터베이스 관리	1204
• 사용자 스키마	1289
• 시스템 카탈로그	1291
• Information Schema	1375
CUBRID 절차적 언어	1407
• 저장 프로시저의 생성	1409
• 저장 프로시저의 호출	1421
• 저장 프로시저의 권한	1424
• 트랜잭션 커밋과 롤백 지원	1427
• 성능 최적화	1429
• 시스템 패키지	1434
• PL/CSQL	1438
• 자바 저장 프로시저	1486
• 메서드	1528
CUBRID 운영	1530
• CUBRID 프로세스 제어	1533
• CUBRID 서비스	
• 데이터베이스 서버	
• 브로커	
• 게이트웨이	
• CUBRID 매니저 서버	
• CUBRID 프로시저 언어 서버	
• 데이터베이스 관리	1618
• cubrid 유틸리티	1625
• 시스템 설정	1774
• SystemTap	1891
• cubrid 스크립트	1903
• 트러블슈팅	1909

• DDL Audit Log	1917
• 데이터베이스 오류 로그	1919

CUBRID HA 2043

• CUBRID HA 기본 개념
• CUBRID HA 기능
• 빠른 시작
• 환경 설정
• 브로커와 DB 연결
• 구동 및 모니터링
• HA 구성 형태
• HA 제약 사항
• 운영 시나리오
• 복제 구축
• 복제 불일치 감지
• Fail-over 원인 분석
• 복제 재구축 스크립트

CUBRID 보안 2175

• 패킷 암호화
• 서버 접근제어
• 권한 관리
• TDE (Transparent Data Encryption)

CUBRID SHARD 2186

• Database Sharding
• CUBRID SHARD 기본 용어
• CUBRID SHARD 주요 기능
• 빠른 시작
• 구성 및 설정
• 구동 및 모니터링
• 설정 테스트
• CUBRID SHARD 로그
• 제약 사항

API 레퍼런스 2214

• JDBC 드라이버	2214
• CCI 드라이버	2258

• PHP 드라이버	2424
• PDO 드라이버	2453
• ODBC 드라이버	2466
• ADO.NET 드라이버	2483
• Perl 드라이버	2495
• Python 드라이버	2497
• Ruby 드라이버	2503
• Node.js 드라이버	2508
릴리스 노트	2509
• 11.4 릴리스 노트	2509
• 공통 정보	

CUBRID 11.4 사용자 매뉴얼

주제별 빠른 링크 모음

General	Administrators	Developers	Authority & Security	Connectors & API	Error Messages & Logs
CUBRID 소개	환경 변수 설정	연산자와 함수	사용자 관리	JDBC 드라이버	오류 메시지 관련 파라미터
시작하기	포트 설정	데이터 조작성문	데이터베이스 서버 접속 제한	CCI 개요	CCI 에러 코드와 에러 메시지
설치 및 업그레이드	시스템 설정	다국어 개요	브로커 서버 접속 제한	PHP 드라이버	JDBC 에러 코드와 에러 메시지
CUBRID SQL	CUBRID 프로세스 제어	조인 방식	DDL 제한	PDO 드라이버	데이터베이스 서버 에러
CUBRID 절차적 언어	cubrid 유틸리티	분할	WHERE 조건 없는 DML 제한	ODBC 드라이버	CAS 에러
	데이터베이스 볼륨	커서 유지	TDE (Transparent Data Encryption)	ADO.NET 드라이버	HA 오류 메시지
	backupdb	PL/CSQL		Perl 드라이버	데이터베이스 서버 로그

General	Administrators	Developers	Authority & Security	Connectors & API	Error Messages & Logs
	restoredb			Python 드라이버	브로커 로그
	트러블슈팅			Ruby 드라이버	
	CUBRID HA			Node.js 드라이버	

매뉴얼 목차

매뉴얼 소개

매뉴얼 구성

CUBRID 데이터베이스 관리 시스템(Database Management System, DBMS) 매뉴얼의 구성은 다음과 같다.

- **CUBRID 소개**: CUBRID 데이터베이스 관리 시스템의 구조 및 특징을 설명한다.
- **설치 및 업그레이드**: CUBRID 설치와 실행, 환경변수 설정, 포트설정, 업그레이드, HA 환경에서 DB 마이그레이션, CUBRID 제거에 대해서 설명한다.
- **시작하기**: CUBRID를 처음 시작하는데 참고할 수 있는 내용을 제공한다. 시스템 설치 및 실행 방법, CUBRID 서버에 연결 시 사용하는 포트, 질의 도구의 간단한 설명을 찾아볼 수 있다.
- **CSQL 인터프리터**: CSQL은 CUBRID에서 명령어 방식으로 SQL 문을 사용할 수 있는 프로그램이다. CSQL 인터프리터의 간단한 사용법과 관련 명령어들을 설명한다.
- **CUBRID SQL**: 데이터 타입, 함수와 연산자, 데이터 조회나 테이블 조작 등, CUBRID에서 사용할 수 있는 SQL 구문에 대해 설명한다. 인덱스나 트리거, 분할, 시리얼 및 사용자 정보 변경 등의 작업을 위한 SQL 구문도 찾아볼 수 있다.
- **CUBRID 절차적 언어**: 저장 프로시저와 저장 함수를 사용할 수 있도록 CUBRID 에서 제공하는 절차적 언어 확장 기능에 대해서 설명한다. 저장 프로시저와 저장 함수는 SQL로 구현하기 어려운 복잡한 비즈니스 로직을 처리하고, 데이터 조작을 간편하게 수행할 수 있도록 도와준다.
- **CUBRID 운영**: 데이터베이스를 생성, 삭제, 백업, 복구 및 마이그레이션하는 방법, 다국어(globalization)를 설정하는 방법 및 CUBRID HA를 수행하는 방법에 대해 설명한다. 또한 서버, 브로커 및 CUBRID Manager 서버 등을 구동하고 종료시키는 **cubrid** 유틸리티의 사용법에 대한 설명도 포함한다. 또한 이 장에서는 성능에 영향을 미칠 수 있는 시스템 파라미터를 설정하는 방법도 설명한다. 서버와 브로커에서 사용하는 설정 파일과 각 파라미터에 대해서도 설명한다.
- **CUBRID HA**: High Availability 기능을 CUBRID에 적용한 것이다. 여러 서버에서 데이터베이스를 항상 동기화된 상태로 유지하고, 서비스 중인 서버에 장애가 발생하면 자동으로 다른 서버가 서비스를 수행한다.

- **CUBRID SHARD**: sharding은 물리적으로 다른 데이터베이스에 데이터를 수평 분할 방식으로 분산 저장하고 조회하는 방법을 말한다. CUBRID SHARD는 하나의 데이터베이스 인스턴스에 넣을 수 없는 큰 데이터와 처리량을 분산하여 처리하기 위해 사용된다.
- **CUBRID 보안**: 패킷 암호화, 서버 접근제어, 권한 관리, TDE(Transparent Data Encryption) 등 CUBRID에서 제공하는 보안 기능에 대해 설명한다.
- **API 레퍼런스**: JDBC API, ODBC API, OLE DB API, PHP API 및 CCI API에 대해 설명한다.
- **릴리스 노트**: 이전 버전 대비 추가, 변경, 개선 및 버그 수정 등을 설명한다.

용어 정리

CUBRID는 상속의 개념을 사용하는 객체 관계형 데이터베이스 시스템이다. 이 매뉴얼에서는 내용의 이해를 돕기 위해 관계형 데이터베이스에서 사용하는 용어를 함께 사용했다. 상속의 개념을 포함하여 설명하는 경우는 클래스, 인스턴스, 속성 등과 같이 주로 객체 기반 용어를 사용했고 일반 SQL 설명에서는 주로 관계형 데이터베이스 용어를 사용했다.

일반 관계형 데이터베이스	CUBRID
테이블	클래스, 테이블
칼럼	속성(attribute), 칼럼
레코드	인스턴스(instance), 레코드
데이터 타입	도메인, 데이터 타입

매뉴얼 규약

다음 표는 CUBRID 데이터베이스 관리 시스템 제품 매뉴얼에서 ‘문장’, ‘명령어’, 그리고 ‘텍스트 안에서
서의 참조’를 식별하는데 사용되는 문서 규약이다.

규약	설명	예제
기울임꼴	변수 이름, 예시로 사용하는 값(시스템, 데이터베이스, 테	

규약	설명	예제
	이블 칼럼, 파일 등의 이름)을 나타낸다.	<i>persistent</i> : <i>stringVariableName</i>
굵게	상수, CUBRID 키워드 등 정해진 값을 나타낸다.	fetch () member function
고정폭 글꼴	구문, 코드 예제 또는 명령어의 실행 및 결과를 나타낸다.	csql database_name
대문자	CUBRID 키워드를 나타내는데 사용된다(굵게 참조).	SELECT
작은 따옴표(' ')	중괄호, 대괄호와 함께 사용되면 문법의 필요한 부분을 나타낸다. 스트링을 감쌀 때도 사용된다.	{'const_list'}
대괄호([])	파라미터나 키워드가 선택적임을 나타낸다.	[ONLY]
세로줄()	여럿 중의 하나만을 선택할 수 있음을 나타낸다.	[COLUMN ATTRIBUTE]
중괄호({ })로 쓴 파라미터	여럿 중에서 하나만을 반드시 선택해야 함을 나타낸다.	CREATE { TABLE CLASS }
중괄호({ })로 쓴 값	집합을 구성하는 원소를 나타낸다.	{2, 4, 6}
중괄호 뒤의 줄임표({ }...)	파라미터 지정이 반복될 수 있음을 나타낸다.	{, class_name }...
산괄호(< >)		<Ctrl+n>

규약	설명	예제

단일 키 또는 일련의 키 입력
을 나타낸다.

버전명 및 버전 문자열 규약

CUBRID 10.0 이후의 버전명 및 버전 문자열은 다음과 같이 표기한다. :

- 버전명: CUBRID M.m Patch p (Major 버전, Minor 버전, Patch 버전(필요한 경우) 표기) CUBRID 10.1 Patch 1 (줄여서 CUBRID 10.1 P1로 표기)
- 버전 문자열: M.m.p.build_number (Major 버전, Minor 버전, Patch 버전, 빌드 번호 표기) 10.2.0.8787-a31ea42

빌드 번호는 하이픈으로 구분되는 두 부분으로 구성된다. 앞 부분은 기본 리비전에서 변경된 횟수를 나타내며 일정하게 증가한다. 뒤 부분은 빌드된 버전의 SHA-1 해시 값이다.

CUBRID 9.0 이후 10.0 이전의 버전명 및 버전 문자열은 다음과 같이 표기한다. :

- 버전명: CUBRID M.m Patch p (Major 버전, Minor 버전, Patch 버전(필요한 경우) 표기) CUBRID 9.2 Patch 1 (줄여서 CUBRID 9.2 P1로 표기)
- 버전 문자열: M.m.p.build_number (Major 버전, Minor 버전, Patch 버전, 빌드 번호 표기) 9.2.1.0012

CUBRID 9.0 이전의 버전명 및 버전 문자열은 다음과 같이 표기한다. :

- 버전명: CUBRID 2008 RM.m Patch p (Major 버전은 2008, Minor 버전, Patch 버전, 빌드 번호 일부 표기) CUBRID 2008 R4.1 Patch 1
- 버전 문자열: 8.m.p.build_number (Major 버전, Minor 버전, Patch 버전, 빌드 번호 표기) 8.4.1.1001

CUBRID 소개

CUBRID의 구조 및 특징을 설명한다. CUBRID는 객체 관계형 데이터베이스 관리 시스템으로서, 데이터베이스 서버, 브로커, CUBRID 매니저로 구성된다. CUBRID는 인터넷 데이터 서비스에 최적화된 데이터베이스 시스템이며, 사용자가 편리하게 사용할 수 있는 다양한 기능을 제공한다.

이 장에서 설명하는 주요 내용은 다음과 같다.

- 시스템 구조: 데이터베이스 볼륨 구조, 서버 프로세스, 브로커 프로세스, 인터페이스 모듈에 대해 설명한다.

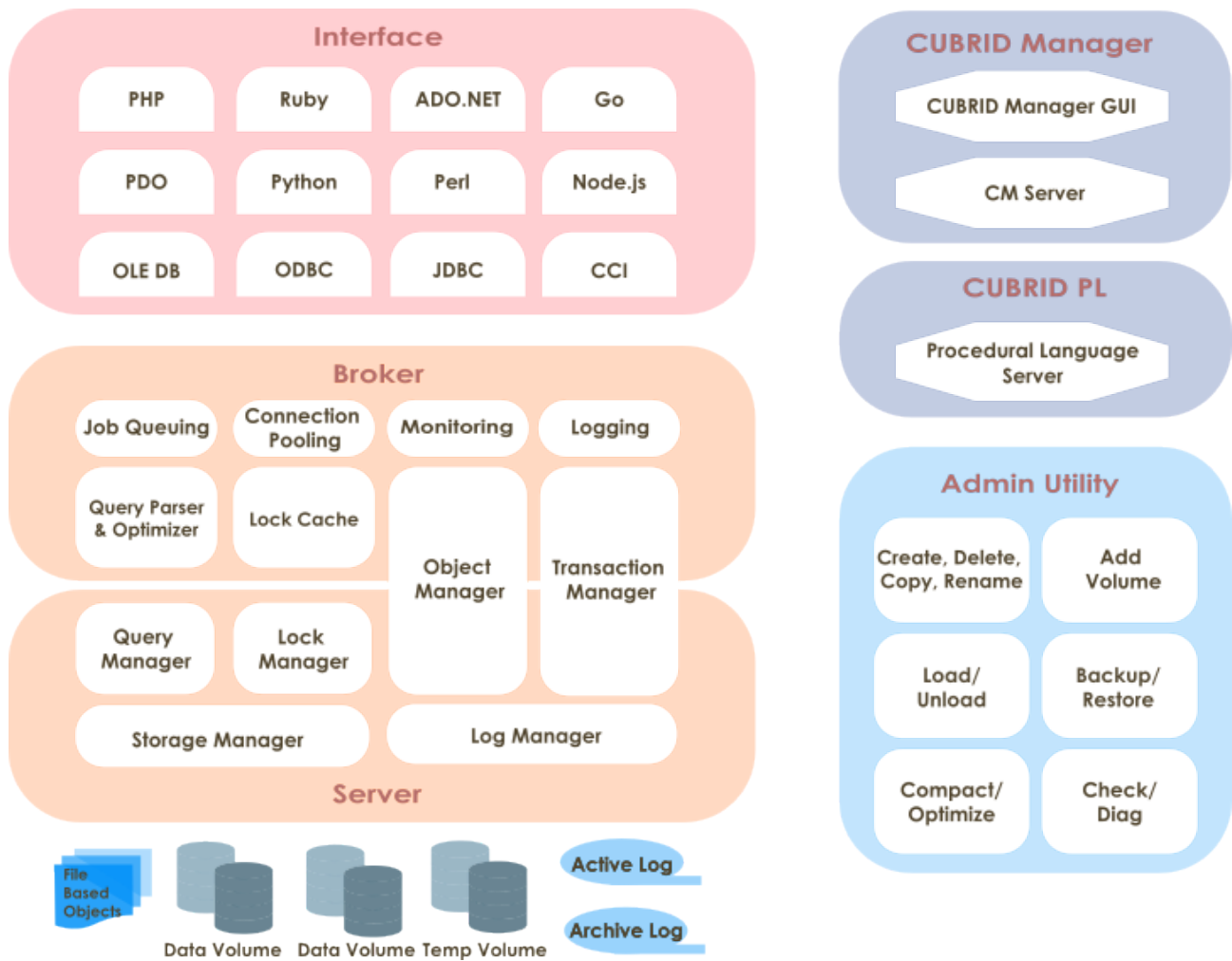
- CUBRID의 특징: CUBRID의 트랜잭션, 백업 및 복구, 파티션, 인덱스 기능, HA 기능, Java 저장 프로시저, 클릭 카운터, 관계형 데이터 모델 확장에 대해 간단히 설명한다.

시스템 구조

프로세스 구조

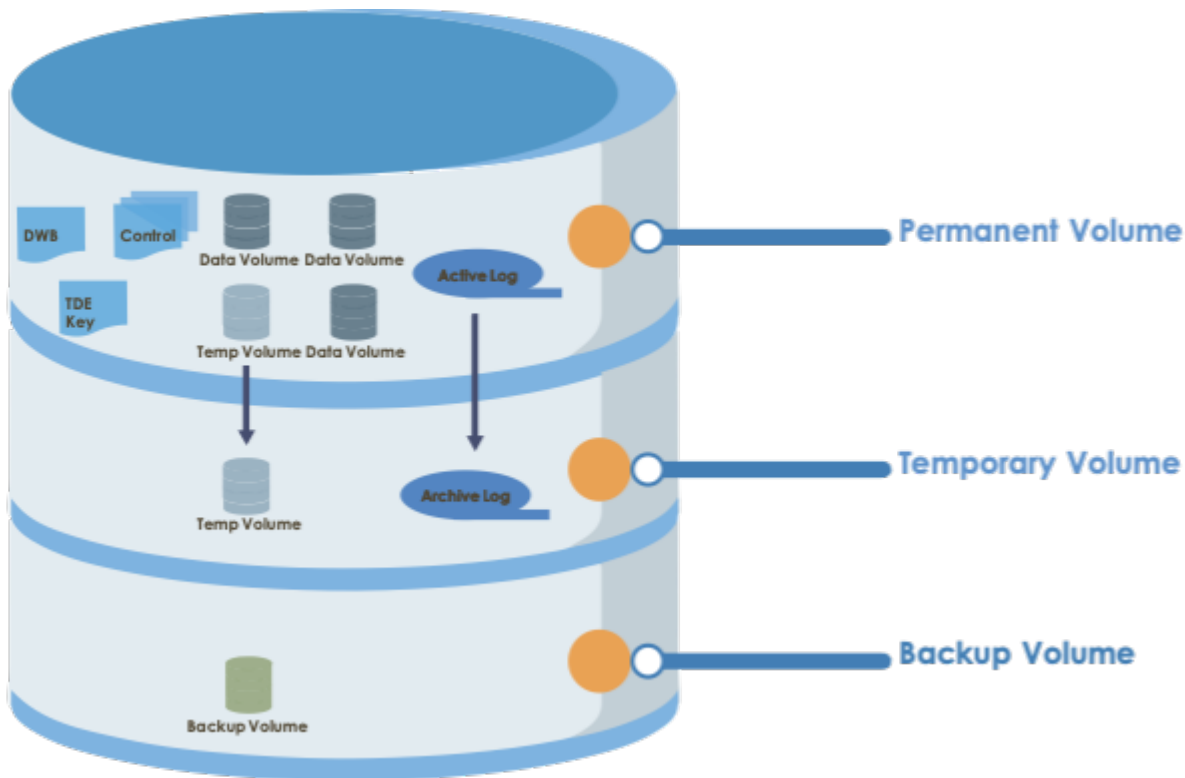
CUBRID는 객체 관계형 데이터베이스 관리 시스템으로서, 데이터베이스 서버, 브로커, CUBRID 매니저로 구성된다.

- 데이터베이스 서버는 CUBRID 데이터베이스 관리 시스템의 핵심 구성 요소로 데이터 저장 및 관리 기능을 수행하며, 멀티스레드 기반 클라이언트/서버 방식으로 동작한다. 데이터베이스 서버는 사용자가 입력한 질의를 처리하고, 데이터베이스 내의 객체를 관리한다. CUBRID 데이터베이스 서버는 잠금 기법과 로깅 기법을 이용해 다수 사용자가 동시에 사용하는 환경에서도 완벽한 트랜잭션을 지원하며, 운영에 필요한 데이터베이스 백업과 복구 기능을 지원한다.
- 브로커는 서버와 외부 응용 프로그램 간의 통신을 중계하는 CUBRID 전용 미들웨어로서, 커넥션 풀링, 모니터링, 로그 추적 및 분석 기능을 제공한다.
- CUBRID Manager는 사용자가 데이터베이스와 브로커를 원격으로 관리할 수 있게 해주는 GUI 도구이다. 사용자가 데이터베이스 서버에 SQL 질의를 수행할 수 있게 해주는 편리한 도구인 질의 편집기를 제공한다.
- CUBRID 프로시저 언어(PL) 서버는 데이터베이스 서버에서 요청한 저장 프로시저와 함수를 컴파일하고 실행하는 서버이다.



데이터베이스 볼륨 구조

아래 그림은 CUBRID 데이터베이스 볼륨의 구조를 도식화한 구성도이다. 데이터베이스 볼륨을 크게 영구적 볼륨, 일시적 볼륨, 백업 볼륨으로 분류하고, 아래 구성도를 참고하여 각각에 속하는 볼륨 및 특징을 살펴보기로 한다.



데이터베이스 볼륨을 생성, 추가, 삭제하는 명령에 관해서는 `createdb`, `addvoldb` 그리고 `deletedb`를 참고한다.

영구적 볼륨(Permanent Volume)

Data Volumes

영구적 볼륨은 한번 생성되면 영구적으로 유지되는 데이터베이스 볼륨이다.

일반적으로 데이터베이스 재시작이나 비정상 종료 후에도 보존해야 하는 데이터를 저장한다. 영구적 데이터의 타입은 다음과 같다.

- 테이블(행 및 멀티미디어 데이터)은 내부적으로 힙 파일 및 힙 오버플로우 파일에 저장되며, 테이블당 하나의 힙 파일 또는 힙 오버플로우 파일을 사용한다.
- 인덱스(키 및 멀티미디어 데이터)는 내부적으로 b-tree 파일 및 b-tree 오버플로우 파일에 저장되며, 인덱스당 하나의 b-tree 또는 b-tree 오버플로우 파일을 사용한다.
- 시스템 데이터는 내부적으로 여러 유형의 파일에 저장되며, 시스템 데이터의 종류로는 파일 추적기(tracker), vacuum 데이터, 삭제된 파일 추적기(tracker) 및 클래스명 해시에 해당한다

사용자는 일시적 데이터를 저장할 일부 영구적 볼륨을 명시적으로 할당할 수 있다. 이러한 볼륨은 절대 제거되지 않는다는 점에서는 영구적이지만 **일시적 볼륨(Temporary Volume)**와 비슷하게 동작한다.

기존 CUBRID 버전에서는 영구적 볼륨을 **generic**, **data** 및 **index** 와 같이 여러 타입으로 분류해 왔으나, 더 이상 이러한 분류를 사용하지 않는다. 이러한 옵션이 여전히 **cubrid createdb** 및 **cubrid addvoldb** 명령에서 허용되긴 하지만 생성되는 볼륨은 동일하며 모든 타입의 영구적 데이터를 저장한다. 볼륨의 용도는 영구적 데이터와 일시적 데이터로만 분류된다.

제어 파일(Control File)

제어 파일은 데이터베이스 내 존재하는 볼륨의 정보, 백업 정보 및 로그의 정보를 저장하는 파일이다.

- **볼륨 정보** : 데이터베이스 내 모든 볼륨의 이름과 위치, 그리고 내부 볼륨 식별자를 포함하는 정보로서, 데이터베이스가 재시작될 때 CUBRID는 볼륨 정보 제어 파일을 판독하며, 새로운 데이터베이스 볼륨이 추가될 때에 새로운 엔트리를 볼륨 정보 제어 파일에 기록한다.
- **백업 정보** : 정보 볼륨에 대한 모든 백업의 위치는 백업 정보 제어 파일에 기록된다. 이 제어 파일은 로그 파일이 관리되는 곳에 유지된다.
- **로그 정보** : 모든 활성 로그와 보관 로그의 이름을 포함하며, 사용자는 로그 정보 제어 파일을 통해 보관 로그의 정보를 확인할 수 있다. 이러한 로그 정보 제어 파일은 로그 파일과 동일한 위치에서 생성 및 관리된다.

이와 같이 각각의 제어 파일은 데이터베이스 볼륨의 위치, 백업 정보, 로그 정보를 포함하며, 데이터베이스가 재시작하면서 읽는 파일이므로 사용자 임의로 변경해서는 안 된다.

활성 로그(Active Log)

활성 로그(active log)는 데이터베이스의 최근 변경 사항을 포함하는 로그이며, 데이터베이스에 문제가 발생하는 경우 활성 로그 및 보관 로그를 이용하여 고장 발생 전의 커밋된 시점으로 완전하게 데이터베이스를 복구할 수 있다.

이중 쓰기 버퍼 (Double Write Buffer, DWB) 파일

이중 쓰기 버퍼 파일은 I/O 에러를 방지하기 위하여 디스크에 쓰여지는 데이터 페이지들의 복사본을 저장한다. 이에 대한 자세한 설명은 [데이터베이스 볼륨](#) 을 참고한다.

TDE 키 파일 (TDE Key File)

TDE (Transparent Data Encryption) 키 파일은 데이터베이스 암호화를 위한 키들을 담고 있다. 파일 내의 키들은 TDE 유틸리티를 사용하여 관리된다. 이에 대한 자세한 설명은 [파일 기반의 마스터 키 관리](#) 와 [TDE 유틸리티](#) 를 참고한다.

일시적 볼륨(Temporary Volume)

일시적 볼륨은 영구적 볼륨과 상반되는 개념이다. 즉, 일시적 볼륨은 서버 프로세스가 종료될 때 삭제되는 일시적으로 생성되는 저장소 파일이며, 질의 처리 및 정렬을 수행할 때 중간 결과와 최종 결과를 저장하는 데 사용된다.

이러한 파일은 질의의 중간 결과와 최종 결과를 저장할 공간을 제공한다. 요구되는 일시적 데이터 크기에 따라, 우선적으로 메모리에 저장된다.(공간 크기는 **cubrid.conf** 에 지정된 시스템 파라미터 **temp_file_memory_size_in_pages** 에 의해 결정됨). 이를 초과하는 데이터는 디스크에 저장한다.

데이터베이스에서는 일시적 데이터를 위한 디스크 공간 할당을 위해 일시적 볼륨을 생성해 사용한다. 그러나 사용자는 **cubrid addvoldb -p temp** 명령을 통해 일시적 데이터를 저장하기 위한 용도로 영구적 볼륨을 할당할 수도 있다. 이러한 영구적 볼륨이 있는 경우 임시 데이터를 디스크 공간에 저장할 때 일시적 볼륨보다 우선 사용한다.

일시적 데이터를 사용할 수 있는 질의의 예는 다음과 같다.:

- **SELECT** 문 등 결과가 생성되는 질의
- **GROUP BY** 나 **ORDER BY** 가 포함된 질의
- 부질의(subquery)가 포함된 질의
- 정렬 병합(sort-merge) 조인이 수행되는 질의
- **CREATE INDEX** 문이 포함된 질의

일시적 데이터에 의해 시스템의 디스크 공간이 모두 사용되는 것을 방지하려면 다음과 같이 조치할 것을 권장한다.

- 영구적 볼륨을 미리 생성해 일시적 데이터에 필요한 공간을 확보한다.
- **cubrid.conf**에서 **temp_file_max_size_in_pages** 파라미터를 설정해 질의를 수행할 때 일시적 볼륨에 사용되는 공간의 크기를 제한한다(기본적으로는 제한 없음).

일시적 볼륨(temporary temp volume)이 한번 생성되면 데이터베이스가 재시작될 때까지 유지되며 크기를 줄일 수 없다. 크기가 너무 큰 경우 데이터베이스를 재시작해 일시적 볼륨이 자동으로 삭제되도록 해야한다.

- **일시적 볼륨의 파일명**: 일시적 볼륨의 파일명 형식은 **db_name_tnum** 이며, 여기서 **db_name** 은 데이터베이스명을, **num** 은 볼륨 식별자를 나타낸다. 볼륨 식별자는 32766부터 1씩 감소한다.
- **일시적 볼륨의 크기 설정**: 생성될 일시적 볼륨의 수는 트랜잭션 처리에 필요한 공간 크기에 따라 시스템에서 결정한다. 그러나 사용자가 시스템 파라미터 설정 파일(**cubrid.conf**)에서 **temp_file_max_size_in_pages** 파라미터 값을 설정해서 총 일시적 볼륨 크기를 제한할 수도 있다. 기본값은 여유 공간이 있는 한 일시적 볼륨을 무제한으로 생성할 수 있음을 나타내는 -1이다. **temp_file_max_size_in_pages** 파라미터 값이 0으로 설정된 경우 일시적 볼륨이 생성되지 않고, 시스템은 일시적 데이터에 할당된 영구적 볼륨만 사용한다.

- **일시적 볼륨의 저장 위치 설정**: 기본적으로 일시적 볼륨은 최초 데이터베이스 볼륨이 생성된 위치에 생성되거나 사용자가 **temp_volume_path** 파라미터 값을 설정해 일시적 볼륨을 저장할 다른 디렉터리를 지정할 수도 있다.
- **일시적 볼륨 삭제**: 일시적 볼륨은 데이터베이스가 실행 중에만 존재하므로 서버가 실행 중일 때 일시적 볼륨을 삭제해서는 안 된다. 일시적 볼륨은 데이터베이스 서버가 정상적으로 종료될 때 삭제되며, 데이터베이스 서버가 비정상적으로 종료될 경우 서버가 재시작될 때 삭제된다.

참고

일반적으로 영구적 볼륨은 영구적 데이터를 저장하는 데 사용되고, 일시적 볼륨은 일시적 데이터를 저장하는 데 사용된다. 일시적 데이터 저장을 위해 영구적 볼륨을 할당할 수는 있으나 일시적 볼륨에는 절대 영구적 데이터가 저장되지 않는다.

보관 로그(Archive Log)

보관 로그는 최근의 변경 사항을 포함하고 있는 활성 로그(active log) 공간이 모두 사용된 후에 지속적으로 생성되는 로그를 보관하기 위한 볼륨이다. 시스템 파라미터 **log_max_archives**의 값이 0보다 크게 설정된 경우 활성 로그 볼륨의 공간이 소진된 후에 보관 로그 볼륨이 추가된다. 제품 설치 시에는 0으로 설정되어 있다. 보관 로그 볼륨은 **log_max_archives**의 설정 값만큼 볼륨 파일이 유지된다. 디스크 공간 확보를 위해 불필요한 보관 로그는 시스템의 설정에 의해 삭제되어야 하지만, 데이터베이스 복구에 사용하려면 이 값을 적절하게 설정해야 한다.

이에 대한 자세한 내용은 [보관 로그 관리](#)를 참고한다.

백그라운드 보관 로그(Background Archive Log)

백그라운드 보관 로그(background archive log)는 백그라운드에서 로그 보관 작업(log archiving)을 수행할 때 사용하는 볼륨이다.

백업 볼륨

백업 볼륨은 데이터베이스에 대한 스냅샷으로서, 이러한 백업 볼륨과 로그 볼륨을 기반으로 특정 시점까지 발생한 트랜잭션을 복구할 수 있다.

사용자는 **cubrid backupdb** 유틸리티를 통해 데이터베이스 복구를 위해 필요한 모든 데이터를 복사할 수 있으며, 데이터베이스 환경 설정 파일(**cubrid.conf**)의 **backup_volume_max_size_bytes** 파라미터 값을 설정하여 백업 볼륨의 분할 크기를 조정할 수 있다.

데이터베이스 서버

DB 서버 프로세스

각 데이터베이스에는 한 개의 서버 프로세스가 존재한다. 서버 프로세스는 CUBRID 데이터베이스 서버를 구성하는 핵심 프로세스로 데이터베이스 파일 및 로그 파일 등에 직접 접근하여, 사용자의 요청을 처리한다. 클라이언트 프로세스는 서버 프로세스와 TCP/IP 통신을 통해 접속하며, 하나의 서버 프로세스는 스레드를 생성해서 다수의 클라이언트 프로세스의 요청 작업을 처리한다. 데이터베이스 별, 즉 서버 프로세스별로 시스템 파라미터 설정을 지정할 수 있으며 서버 프로세스는 **max_clients** 파라미터 값으로 지정된 수만큼 클라이언트 프로세스의 접속이 가능하다.

마스터 프로세스

마스터 프로세스는 클라이언트 프로세스가 서버 프로세스에 접속하여 통신할 수 있게 하는 중개 프로세스로서, 호스트별로 한 개씩 동작한다. (정확히는 시스템 파라미터 파일인 **cubrid.conf**에 지정되는 접속 포트 번호별로 하나씩의 마스터 프로세스가 존재한다.) 마스터 프로세스는 지정된 TCP/IP 포트에 대기하고 있고, 클라이언트 프로세스는 해당 TCP/IP 포트에 마스터 프로세스에 접속한 후 마스터 프로세스가 지정된 데이터베이스 이름에 따라 해당 서버 프로세스로 소켓 포트를 변경하여 접속을 처리한다.

실행 모드

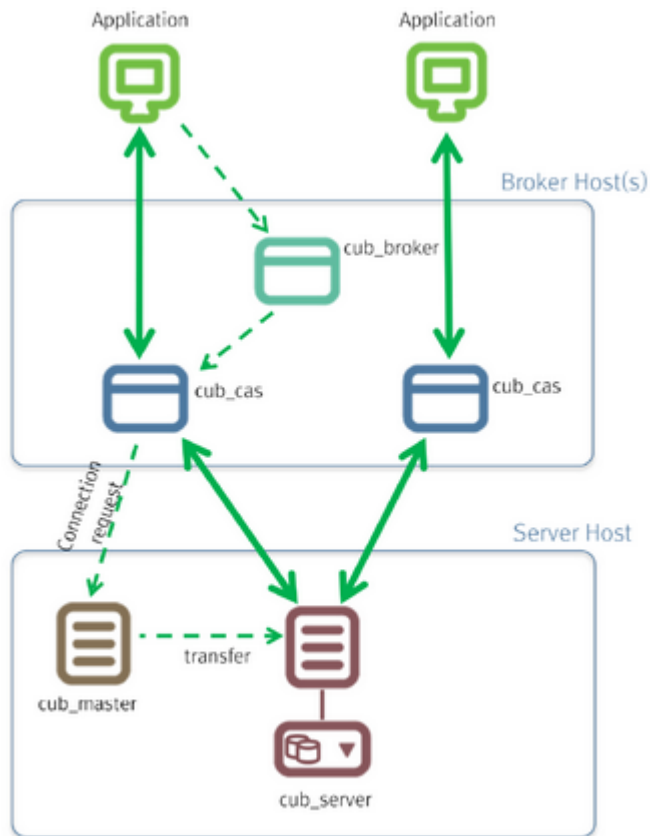
서버 프로세스를 제외한 CUBRID의 프로그램들은 종류에 따라 두 가지 실행 모드가 있다. 실행 모드는 클라이언트/서버 모드(client/server mode)와 독립 모드(standalone mode)로 나뉜다.

- 클라이언트/서버 모드는 해당 프로그램이 클라이언트 프로세스로서 동작하여 서버 프로세스에 접속하는 방식이다.
- 독립 모드는 해당 프로그램이 서버 프로세스의 기능을 포함하고 있어 직접 데이터베이스 파일에 접근하여 수행하는 방식이다.

예를 들어, 데이터베이스 생성 유틸리티나 복구 유틸리티 등은 다수 사용자가 데이터베이스에 접근하는 것을 막고 해당 프로그램만이 온전히 점유해서 작업할 수 있도록 독립 모드로 실행된다. 또 다른 예로, CSQL 인터프리터는 클라이언트/서버 모드로 동작하여 서버 프로세스에 접속할 수도 있고, 독립 모드로 동작하여 데이터베이스에 접근하여 SQL 문을 실행할 수도 있다. 참고로, 하나의 데이터베이스에 서버 프로세스와 독립 모드로 실행되는 프로그램이 동시에 접근할 수는 없다.

브로커

브로커는 다양한 응용 클라이언트가 데이터베이스 서버에 연결할 수 있도록 중계하는 미들웨어이다. 브로커를 포함하는 큐브리드 시스템은 아래 그림과 같이, 응용 클라이언트(application), cub_broker, cub_cas, 데이터베이스 서버(cub_server)를 포함한 다중 계층 구조를 가진다.



응용 클라이언트

응용 클라이언트에서 사용할 수 있는 인터페이스는 C-API(CCI, CUBRID Call Interface), ODBC, JDBC, PHP, Python, Ruby, OLEDB, ADO.NET, Node.js 등이 있다.

cub_cas

cub_cas(CUBRID Common Application Server, 브로커 응용 서버, 또는 줄여서 응용 서버, CAS라고도 함)는 연결을 요청하는 모든 종류의 응용 클라이언트가 사용하는 공용 응용 서버 역할을 한다. 또한, cub_cas는 데이터베이스 서버의 클라이언트로 동작하여 클라이언트의 요청에 의해 데이터베이스 서버와 연결을 제공한다. 서비스 풀(service pool) 내에서 구동되는 cub_cas의 개수는 **cubrid_broker.conf** 설정 파일에 지정할 수 있으며, cub_broker에 의해 동적으로 조정된다.

cub_cas는 CUBRID 데이터베이스 서버의 클라이언트 라이브러리와 링크되는 프로그램으로 데이터베이스 서버 프로세스(cub_server)에는 클라이언트 모듈로 동작하며, 쿼리 파싱이나 최적화, 실행 계획 생성 등의 작업이 클라이언트 모듈에서 수행된다.

cub_broker

cub_broker는 응용 클라이언트와 cub_cas 사이의 연결을 중계하는 기능을 수행한다. 즉, 응용 클라이언트가 접근을 요청하면, cub_broker는 공유 메모리(shared memory)를 통해 cub_cas의 상태를 파악하여 접근 가능한 cub_cas에게 요청을 전달하고, 해당 cub_cas로부터 전달 받은 요청에 대한 처리 결과를 응용 클라이언트에게 반환한다.

또한, cub_broker는 서비스 풀 내의 cub_cas 개수를 조정하여 서버 부하를 관리하고, cub_cas의 구동 상태를 모니터링 및 관리한다. 만약, 응용 클라이언트의 요청을 cub_cas 1에게 전달하였는데, 비

정상적인 종료로 인해 cub_cas 1과의 연결이 실패하면, cub_broker는 응용 클라이언트에게 연결 실패에 관한 에러 메시지를 전송하고 cub_cas 1을 재구동한다. 새롭게 구동된 cub_cas 1은 정상적인 대기 상태가 되어, 새로운 응용 클라이언트의 요청에 의해 재연결된다.

공유 메모리

공유 메모리에는 cub_cas의 상태 정보가 저장되며, cub_broker는 공유 메모리에 저장된 cub_cas의 상태 정보를 참조하여 응용 클라이언트와의 연결을 중계한다. 공유 메모리에 저장된 cub_cas의 상태 정보를 통해 시스템 관리자는 어떤 cub_cas가 현재 작업을 수행중인지, 어떤 응용 클라이언트의 요청이 처리 중인지를 확인할 수 있다.

인터페이스 모듈

CUBRID는 다양한 응용 프로그래밍 인터페이스(API: Application Programming Interface)를 제공한다. 지원되는 API는 다음과 같다.

- JDBC: Java 환경에서 데이터베이스 응용 프로그램을 작성하는 표준 API
- ODBC: Windows 환경에서 데이터베이스 응용 프로그램을 작성하는 표준 API. ODBC 드라이버는 CCI 라이브러리를 기반으로 작성되었다.
- OLE DB: Windows 환경에서 COM 방식으로 데이터베이스 응용 프로그램을 작성하는 API. OLE DB 프로바이더는 CCI 라이브러리를 기반으로 작성되었다.
- PHP: PHP 환경에서 데이터베이스 응용 프로그램을 작성하는 API. PHP 드라이버는 CCI 라이브러리를 기반으로 작성되었다.
- CCI: CUBRID에서 제공하는 C 언어 인터페이스. C 라이브러리 형태로 제공된다.

각 인터페이스 모듈들은 모두 브로커를 통해서 데이터베이스 서버에 접근하게 된다. 브로커는 다양한 응용 클라이언트가 데이터베이스 서버에 연결할 수 있도록 중계하는 미들웨어로, 각 인터페이스 모듈의 요청을 받아서 데이터베이스 서버의 클라이언트 라이브러리에서 제공하는 native-C API를 호출하게 된다.

CUBRID의 특징

완벽한 트랜잭션 지원

트랜잭션의 원자성(atomicity), 일관성(consistency), 격리성(isolation), 지속성(durability)을 완벽하게 보장하기 위해 CUBRID는 다음의 기능을 충실하게 지원한다.

- 트랜잭션 단위의 commit, rollback, savepoint 지원
- 시스템이나 데이터베이스의 장애 시 트랜잭션 일관성 보장
- 복제 간 트랜잭션 일관성 보장

- 데이터베이스, 테이블, 레코드 등 다중 단위 잠금(multiple granularity locking) 지원
- 교착 상태(deadlock) 자동 해결

데이터베이스 백업 및 복구

데이터베이스 백업은 CUBRID 데이터베이스 볼륨, 제어 파일, 로그 파일을 저장하는 작업이고, 데이터베이스 복구는 백업 작업에 의해 생성된 백업 파일, 활성 로그, 보관 로그를 이용하여 특정 시점의 데이터베이스로 복구하는 작업이다. 이 때, 복구 환경은 백업 환경과 동일한 운영체제 및 동일 버전의 CUBRID가 설치되어야 한다. CUBRID가 지원하는 백업 방식으로는 온라인 백업, 오프라인 백업, 증분 백업이 있고, 복구 방식으로는 증분 백업에 의한 복구, 부분 복구, 전체 복구가 있다.

테이블 분할 - 파티션

분할 기법(partitioning)은 하나의 테이블을 여러 개의 독립적인 논리적 단위로 분할하는 기법을 가리킨다. 각 논리적 단위를 분할(partition)이라 부르며, 각 분할을 서로 다른 물리적 공간에 나누어 저장하도록 하여 레코드를 검색할 때 해당 분할만 접근할 수 있도록 하여 성능 향상을 기대할 수 있다. CUBRID가 제공하는 분할 기법은 다음과 같다.

- 레인지 분할 기법 : 칼럼 값의 범위를 기준으로 테이블을 분할하는 기법
- 해시 분할 기법 : 칼럼의 해시값을 기준으로 분할하는 기법
- 리스트 분할 기법 : 칼럼 값의 목록을 기준으로 분할하는 기법

다양한 인덱스 기능 지원

CUBRID는 다양한 조건 질의를 수행할 때 가급적 인덱스를 활용할 수 있도록 다음과 같은 인덱스 기능을 지원한다.

- 내림차순 인덱스 스캔(Descending Index Scan): 별도의 내림차순 인덱스를 생성하지 않아도 오름차순 인덱스만으로 내림차순 인덱스 스캔 가능
- 커버링 인덱스(Covering Index): **SELECT** 리스트의 칼럼이 인덱스에 포함된 경우 인덱스 스캔만으로 요구하는 데이터를 가져올 수 있음
- **ORDER BY** 절 최적화: 요구하는 레코드의 정렬 순서가 인덱스의 순서와 같다면 별도의 정렬 작업이 필요 없음(Skip ORDER BY)
- **GROUP BY** 절 최적화: **GROUP BY** 절에 있는 모든 칼럼이 인덱스에 포함된다면 질의 수행 시 인덱스를 사용할 수 있어 별도의 정렬 작업이 필요 없음(Skip GROUP BY)

HA 기능

CUBRID는 하드웨어, 소프트웨어, 네트워크 등에 장애가 발생해도 지속적인 서비스가 가능하게 하는 HA(High Availability) 기능을 제공한다. CUBRID의 HA 기능은 shared-nothing 구조이며, CUBRID Heartbeat을 이용하여 시스템과 CUBRID의 상태를 실시간으로 감시하고 장애 발생 시 절체(failover)

를 수행한다. CUBRID HA 환경에서 마스터 데이터베이스 서버로부터 슬레이브 데이터베이스 서버로의 데이터 동기화를 위해 다음 두 단계를 수행한다.

- 마스터 데이터베이스 서버에서 생성되는 트랜잭션 로그를 실시간으로 다른 노드에 복제하는 트랜잭션 로그 다중화 단계
- 실시간으로 복제되는 트랜잭션 로그를 분석하여 슬레이브 데이터베이스 서버로 데이터를 반영하는 트랜잭션 로그 반영 단계

저장 프로시저

저장 프로시저는 미들웨어에서 실행되는 로직과 데이터베이스에서 실행되는 로직을 분리하여 응용 프로그램의 복잡성을 줄이고, 재사용성, 보안성, 성능을 향상시킬 수 있는 기법이다. CUBRID는 SQL 확장 언어인 PL/CSQL을 지원하고 범용 언어인 Java로도 작성할 수 있다. 저장 프로시저 기능을 사용하기 위해 프로시저 언어 서버가 구동된다. (CUBRID [프로시저 언어 서버](#)를 참고) 저장 프로시저에 대한 자세한 내용은 [CUBRID 절차적 언어](#) 를 참고한다.

클릭 카운터

인터넷 환경에서 데이터 검색 시 보통 검색 이력을 남기기 위해 조회수와 같은 카운터를 데이터베이스에 유지한다.

일반적으로 위의 시나리오는 **SELECT** 문을 이용하여 데이터를 검색하고, 검색한 질의에 대한 조회수를 증가 시키기 위해 다시 **UPDATE** 문을 통해 구현하는 것이 일반적인 방식이었다.

이 방식은 한 데이터에 **SELECT** 가 집중될 때 **UPDATE** 에 대한 잠금(Lock) 경쟁이 가중되어 급격한 성능 저하가 발생하는 단점이 존재한다.

이에 CUBRID는 인터넷 환경에서 사용자 편의성 및 성능 측면에서 최적화된 기능을 제공하기 위해 클릭 카운터(Click Counter) 라는 새로운 개념을 도입하고, 이를 위해 `INCR()` 함수 및 **WITH INCREMENT FOR** 구문을 제공한다.

관계형 데이터 모델 확장

· 컬렉션

관계형 데이터베이스에서는 한 칼럼이 여러 개의 값을 가지는 것을 허용하지 않지만, CUBRID는 한 칼럼이 여러 개의 값을 가지도록 정의할 수 있다. 이를 위해 CUBRID에서는 컬렉션(collection)이라는 데이터 타입을 제공하는데, 컬렉션 타입은 컬렉션 원소의 중복 허용 여부와 순서 유지 여부에 따라 크게 **SET**, **MULTISET**, **LIST** 의 세 종류로 구분할 수 있다.

- **SET**: 각 원소의 중복을 허용하지 않는 집합으로서, 원소의 나열 순서와 무관하게 중복 없이 정렬되어 저장된다.
- **MULTISET**: 각 원소의 중복을 허용하는 집합으로서, 원소의 나열 순서와 무관하다.

- **LIST**: 각 원소의 중복을 허용하는 집합으로서, **SET**, **MULTISET** 과 달리 원소의 순서를 유지한다.

· JSON

JavaScript Object Notation (JSON) 은 데이터 교환을 위한 사실상의 표준이 되었다. 관계형 데이터 모델에서는 반 정형 데이터 중 하나인 JSON을 가지는 것을 허용하지 않는다. 그러나 큐브리드에서는 **SQL JSON 함수**를 사용하여 JSON 문서를 생성하고 질의 할 수 있다. 또한 **JSON 데이터 타입** 컬럼을 정의하고 JSON 문서를 JSON 타입 컬럼에 저장할 수 있다.

· 상속

상속은 상위 클래스(테이블)에서 생성된 칼럼과 메서드들을 하위 클래스에서 재사용할 수 있게 하는 개념으로, CUBRID는 상속을 지원함으로써 재사용성을 제공한다. CUBRID에서 제공하는 상속 기능을 이용하여 공통의 칼럼을 가지는 상위 클래스를 생성하고, 상위 클래스를 상속받아 고유한 칼럼을 추가한 하위 클래스를 생성함으로써, 필요한 칼럼 수를 최소화한 데이터베이스 모델링이 가능해진다.

설치 및 업그레이드

CUBRID가 사용하는 포트에 대해서는 **포트 설정**을 참고한다. Linux 환경에서는 **APPL_SERVER_PORT**를 제외한 나머지 포트 설정이 Windows 환경에서의 포트 설정과 동일하다.

CUBRID의 다양한 도구 및 드라이버는 <https://www.cubrid.org/downloads>에서 다운로드할 수 있다.

설치와 실행

지원 플랫폼 및 설치 최소 사양

CUBRID가 지원하는 플랫폼과 설치를 위한 하드웨어/소프트웨어 요구 사항은 아래 표와 같다.

지원 플랫폼	요구되는 메모리	요구되는 디스크 공간
Windows 10 버전 2004 (빌드 19041) 이상 또는 Windows 11(with WSL)	2G 이상	2G 이상(*)
	1G 이상	2G 이상(*)

지원 플랫폼	요구되는 메모리	요구되는 디스크 공간
Windows 64 Bit Windows 7 이상 (native Windows)		
Linux 계열 64 Bit(Linux kernel 2.4 및 glibc 2.3.4 이상)	1G 이상	2G 이상(*)

(*): 처음 설치 시 약 500MB의 디스크 공간이 필요하며, 하나의 DB를 기본 옵션으로 생성할 경우 약 1.5GB의 디스크 공간이 필요하다.

CUBRID 패키지 설치 시 CUBRID Admin 클라이언트가 같이 설치되지 않는다. 따라서 CUBRID Admin 사용하려면 이를 추가로 설치해야 한다. CUBRID 설치 패키지는 <http://ftp.cubrid.org> 에서 받을 수 있다.

JDBC, ODBC, PHP 등의 드라이버들도 <http://ftp.cubrid.org> 에서 받을 수 있다.

CUBRID 엔진, 사용 도구 및 드라이버에 대한 자세한 정보는 <https://www.cubrid.org> 를 참고한다.

버전 호환성

응용 프로그램의 호환성

- 2008 R4.1 또는 그 이상 버전에서 JDBC, PHP, CCI API 등을 사용하는 응용 프로그램은 CUBRID 11.4 브로커에 접근할 수 있다. 다만, JDBC, PHP, CCI 인터페이스에 추가/개선된 기능을 사용하기 위해서는 CUBRID 11.4 버전의 라이브러리를 링크하거나 드라이버를 사용해야 한다. 10.0에서 추가된 **타입 존이 있는 날짜/시간 데이터 타입** 을 사용하기 위해서는 드라이버를 업그레이드 해야 한다.
- 10.2 이전 버전의 드라이버는 JSON 타입 컬럼을 Varchar 타입으로 인식한다.
- 새로운 예약어 추가 및 일부 질의에 대한 스펙 변경으로 인해 질의 결과가 과거 버전과 다를 수 있으므로 주의한다.
- 2008 R3.0 이하 버전에서 GLO 클래스를 이용하여 개발된 응용은 BLOB, CLOB 타입에 맞는 응용 및 스키마로 변환하여 사용해야 한다.

CUBRID Admin의 호환성

- CUBRID Admin는 CUBRID 10.0 이상 버전의 서버에 대해서 하위 호환성을 보장하며, 각 서버 버전과 일치하는 CUBRID JDBC 드라이버를 사용한다. 하지만 CUBRID Admin에서 제공하는 모든 기능을 제대로 사용하기 위해서는 CUBRID 서버 버전보다 높은 버전의 CUBRID Admin를 사용해야 한다. CUBRID JDBC 드라이버는 CUBRID 설치 시 \$CUBRID/jdbc 디렉터리에 포함되어 있다(Linux 환경에서 \$CUBRID는 Windows 환경에서는 %CUBRID% 형식으로 사용됨).

참고

과거 버전 사용자들은 드라이버, 브로커, DB 서버 모두를 반드시 업그레이드해야 하며, 11.3 이하 버전의 DB 볼륨이 11.4와 호환되지 않으므로 반드시 데이터 마이그레이션을 해야 한다. 업그레이드 및 데이터 마이그레이션은 [업그레이드](#)를 참고한다.

CUBRID DB 서버와 브로커 간 상호 운용성

- CUBRID DB 서버와 브로커를 분리하여 운영하는 경우, 두 장비 간 CUBRID 버전은 동일해야 한다. 그러나 패치 버전만 다른 경우는 호환된다.

예를 들어, 2008 R4.1 Patch1의 브로커는 2008 R4.1 Patch 10의 DB 서버와 호환되지만 2008 R4.3 DB 서버와는 호환되지 않는다. 9.1 Patch 1의 브로커는 9.1 Patch 10의 DB 서버와 호환되지만 9.2 DB 서버와는 호환되지 않는다.

- CUBRID DB 서버와 브로커 장비의 운영 체제가 서로 다르더라도 DB 서버의 Bit 버전과 브로커 서버의 Bit 버전이 서로 동일하면 상호 운용성을 보장한다.

예를 들어, Linux용 64Bit 버전 DB 서버는 Windows용 64Bit 버전 브로커 서버와 상호 운용이 가능하다.

DB 서버와 브로커 서버 사이의 관계에 대한 설명은 [CUBRID 소개](#)를 참고한다.

Linux에서의 설치와 실행

설치 전 확인 사항

Linux 버전의 CUBRID 데이터베이스를 설치하기 전에 다음 사항을 점검한다.

- glibc 버전

glibc 2.3.4 버전 이상만 지원한다. glibc 버전은 다음과 같은 방법으로 확인한다.

```
% rpm -q glibc
```

- 64비트

10.0 이후 CUBRID는 64비트 버전만 지원한다. Linux버전은 다음과 같은 방법으로 확인한다.

```
% uname -a
Linux host_name 2.6.32-696.20.1.el6.x86_64 #1 SMP Fri Jan 26
17:51:45 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
```

64비트 Linux에서는 CUBRID 64비트 버전을 설치한다.

· 추가로 설치할 라이브러리

- Curses Library (rpm -q ncurses)

CUBRID는 curses 라이브러리를 사용하여 구동되므로 시스템에 curses 라이브러리가 없는 경우 사전에 최신 ncurses-compat-libs 패키지를 설치해야 한다.

- gcrypt Library (rpm -q libgcrypt)

- stdc++ Library (rpm -q libstdc++)

· /etc/hosts 파일에 호스트 이름과 IP 주소 매핑이 정상인지 확인하기

호스트 이름과 이에 맞는 IP 주소가 비정상적으로 매핑되어 있으면 DB 서버를 구동할 수 없으므로, 정상적으로 매핑되어 있는지 확인한다.

CUBRID 설치

설치 프로그램은 바이너리를 포함한 쉘 스크립트로 되어 있어 자동으로 설치할 수 있다. 다음은 리눅스에서 “CUBRID-11.4-latest-Linux.x86_64.sh” 파일을 이용하여 CUBRID를 설치하는 예제이다.

```
$ sh CUBRID-11.4-latest-Linux.x86_64.sh
Do you agree to the above license terms? (yes or no) : yes
Do you want to install this software(CUBRID) to the default(/home1/
cub_user/CUBRID) directory? (yes or no) [Default: yes] : yes
Install CUBRID to '/home1/cub_user/CUBRID' ...
In case a different version of the CUBRID product is being used in
other machines,
please note that the CUBRID 11.4 servers are only compatible with
the CUBRID 11.4 clients and vice versa.
Do you want to continue? (yes or no) [Default: yes] : yes
Copying old .cubrid.sh to .cubrid.sh.bak ...

CUBRID has been successfully installed.

demodb has been successfully created.

If you want to use CUBRID, run the following commands
$ . /home1/cub_user/.cubrid.sh
$ cubrid service start
```

위의 예제와 같이 다운로드한 파일(CUBRID-11.4-latest-Linux.x86_64.sh)을 설치한 후, CUBRID 데이터 베이스를 사용하기 위해서는 사용자의 shell 환경에 맞게 .cubrid.sh (또는 .cubrid.csh) 수행하여 CUBRID 관련 환경 변수값을 설정해야 한다. 또한, 로그인시 자동 설정하려면 사용자의 shell 환경에 맞게 .cubrid.sh (또는 .cubrid.csh)를 .bashrc (또는 .cshrc) 등에 수동으로 등록해주어야 한다.

```
$ . /home1/cub_user/.cubrid.sh
```

CUBRID가 설치 완료되면 CUBRID 매니저 서버와 브로커를 다음과 같이 구동시킬 수 있다.

```
$ cubrid service start
```

cubrid service를 구동시킨 후 정상적으로 구동되었는지 확인하려면 Linux에서는 다음과 같이 grep으로 cub_* 프로세스들이 구동되어 있는지를 확인한다.

```
$ ps -ef | grep cub_
cub_user 15200 1 0 18:57 00:00:00 cub_master
cub_user 15205 1 0 18:57 pts/17 00:00:00 cub_broker
cub_user 15210 1 0 18:57 pts/17 00:00:00 query_editor_cub_cas_1
cub_user 15211 1 0 18:57 pts/17 00:00:00 query_editor_cub_cas_2
cub_user 15212 1 0 18:57 pts/17 00:00:00 query_editor_cub_cas_3
cub_user 15213 1 0 18:57 pts/17 00:00:00 query_editor_cub_cas_4
cub_user 15214 1 0 18:57 pts/17 00:00:00 query_editor_cub_cas_5
cub_user 15217 1 0 18:57 pts/17 00:00:00 cub_broker
cub_user 15222 1 0 18:57 pts/17 00:00:00 broker1_cub_cas_1
cub_user 15223 1 0 18:57 pts/17 00:00:00 broker1_cub_cas_2
cub_user 15224 1 0 18:57 pts/17 00:00:00 broker1_cub_cas_3
cub_user 15225 1 0 18:57 pts/17 00:00:00 broker1_cub_cas_4
cub_user 15226 1 0 18:57 pts/17 00:00:00 broker1_cub_cas_5
cub_user 15229 1 0 18:57 00:00:00 cub_auto start
cub_user 15232 1 0 18:57 00:00:00 cub_js start
```

RPM으로 CUBRID 설치

CentOS 6 환경에서 생성한 RPM 파일을 사용하여 CUBRID를 설치할 수 있으며, 일반적인 RPM 유틸리티와 동일한 방법으로 설치하고 삭제할 수 있다. 설치하면 새로운 시스템 그룹(cubrid) 및 사용자 계정(cubrid)이 생성되며, 설치 후에는 cubrid 사용자 계정으로 로그인하여 CUBRID 서비스를 시작해야 한다.

```
$ rpm -Uvh CUBRID-11.4-latest-Linux.x86_64.rpm
```

RPM을 실행하면 CUBRID는 “cubrid” 홈 디렉터리(/opt/cubrid)에 설치되고, CUBRID 관련 환경 설정 파일(cubrid.[c]sh)이 /etc/profile.d 디렉터리에 설치된다. 단, demodb는 자동으로 설치되지 않으므로 “cubrid” Linux 계정으로 로그인하여 /opt/cubrid/demo/make_cubrid_demo.sh를 실행하여야 한다. CUBRID가 설치 완료되면 “cubrid” Linux 계정으로 로그인하여 CUBRID 서비스를 다음과 같이 시작한다.

```
$ cubrid service start
```


참고

· RPM과 의존성

RPM으로 설치할 때에는 의존성을 꼭 확인해야 한다. 의존성을 무시(--nodeps)하고 설치하면 실행되지 않을 수 있다.

· RPM 삭제 후에도 계정 및 DB는 남아 있음

RPM을 삭제하더라도 cubrid 사용자 계정 및 설치 후 생성한 데이터베이스는 보관되므로, 더 이상 필요하지 않은 경우 수동으로 삭제해야 한다.

· Linux에서 시스템 구동 시 CUBRID 자동 구동하기

SH 패키지로 CUBRID를 설치했다면 \$CUBRID/share/init.d 디렉터리에 cubrid라는 스크립트가 포함되어 있다. 이 파일 안의 **CUBRID_USER** 라는 환경 변수 값을 CUBRID를 설치한 Linux 계정으로 변경한 후, /etc/init.d에 등록하면 service나 chkconfig 명령을 사용하여 Linux 시스템 구동 시 CUBRID를 자동으로 구동할 수 있다.

RPM 패키지로 CUBRID를 설치했다면 /etc/init.d 디렉터리에 cubrid 스크립트가 추가된다. 그러나 cubrid 스크립트 파일 안의 \$CUBRID_USER 환경 변수를 cubrid 계정으로 변경하는 작업이 필요하다.

· /etc/hosts 파일에 호스트 이름과 IP 주소 매핑이 정상인지 확인하기

호스트 이름과 이에 맞는 IP 주소가 비정상적으로 매핑되어 있으면 DB 서버를 구동할 수 없으므로, 정상적으로 매핑되어 있는지 확인한다.

CUBRID 업그레이드

다른 버전의 CUBRID가 설치된 디렉터리를 CUBRID를 설치할 디렉터리로 지정하면, 해당 디렉터리가 존재하는 것을 알리고 덮어쓸 것인지 확인한다. **no** 를 입력하면 설치가 중단된다.

```
Directory '/home1/cub_user/CUBRID' exist!
If a CUBRID service is running on this directory, it may be
terminated abnormally.
And if you don't have right access permission on this
directory(subdirectories or files), install operation will be failed.
Overwrite anyway? (yes or no) [Default: no] : yes
```

CUBRID를 설치하고 설정 파일을 구성할 때 기존의 설정 파일을 그대로 사용할 것인지, 새 설정 파일을 사용할 것인지 확인한다. **yes** 를 입력하면 기존의 설정 파일을 확장자가 .bak인 백업 파일로 보관한다.

```
The configuration file (.conf or .pass) already exists. Do you want
to overwrite it? (yes or no) : yes
```

과거 버전에서 새 버전으로 데이터베이스를 업그레이드하는 방법에 대한 보다 자세한 내용은 **업그레이드** 를 참고한다.

환경 설정

서비스 포트 등 사용자 환경에 맞춰 설정을 변경하려면 **\$CUBRID/conf** 디렉터리에서 설정 파일의 파라미터를 수정한다. 자세한 내용은 **Windows에서의 설치와 실행**의 환경 설정을 참고한다.

CUBRID 인터페이스 설치

CCI, JDBC, PHP, ODBC, OLE DB, ADO.NET, Ruby, Python, Node.js 등의 인터페이스 모듈은 <https://www.cubrid.org/downloads> 에서 최신 정보를 확인할 수 있고 관련 파일을 내려받아 설치할 수 있다.

각 드라이버에 대한 간단한 설명은 **API 레퍼런스** 를 참고한다.

CUBRID 도구 설치

CUBRID Admin, CUBRID Migration ToolKit 등의 도구는 <https://www.cubrid.org/downloads> 에서 최신 정보를 확인할 수 있고 관련 파일을 내려받아 설치할 수 있다.

Windows에서 WSL(Windows Subsystem for Linux) 환경을 이용한 설치와 실행

Windows의 WSL(Windows Subsystem for Linux) 환경에서 CUBRID를 설치하고 실행할 수 있는 설치 프로그램을 제공한다. (<https://www.cubrid.com/downloads>) WSL 환경에서 CUBRID를 설치하고 실행하려면, WSL이 설치되어 있어야 한다. WSL 설치 방법은 <https://learn.microsoft.com/ko-kr/windows/wsl/install> 에서 확인할 수 있다.

설치 전 확인 사항

Windows 버전의 CUBRID 데이터베이스를 설치하기 전에 다음 사항을 점검한다.

- Windows 버전 호환성 확인

Windows 10 버전 2004 이상(Build 19041 이상) 또는 Windows 11

- 가상화 기술 지원 확인 (Intel VT-x, AMD-V)

BIOS/UEFI에서 가상화 기술이 활성화되어 있는지 확인한다.

- WSL(Windows Subsystem for Linux) 설치 및 활성화

<https://github.com/microsoft/WSL/releases> 에서 최신 WSL2를 설치하고 Windows 기능에서 WSL을 활성화 한다. CUBRID 설치 프로그램은 WSL 설치를 자동으로 수행하지 않으므로, WSL 활성화 상태를 체크하고 활성화 하는 기능만 제공한다. WSL 기능을 설치 & 활성화 하는 경우 재부팅이 필요하며, 재부팅 후 설치 프로그램을 다시 실행해야 한다.

- 설치 프로그램은 관리자 권한이 필요하다.

설치 과정

제공 되는 EXE 설치 프로그램을 실행하면, 설치 프로그램이 WSL 환경을 체크하고, 환경을 만족하지 않은 경우 안내 메시지를 보여주고 설치를 중단한다. WSL이 활성화 되어 있으면, 설치 프로그램은 CUBRID가 포함되어 있는 Linux 배포판과 CUBRID Server 관리를 위한 Tray 애플리케이션을 설치 한다.

1단계: 설치 옵션 선택

- 배포판 이름 설정
- 자동 실행 목록에 Tray 애플리케이션 등록 여부 선택
- 바탕화면 바로가기 생성 여부
- 샘플 데이터베이스 생성 여부
- WSL2 모드 선택 (체크시 WSL2 모드로 설치, 체크 해제시 WSL1 모드로 설치)

2단계: 설치 디렉터리 지정

3단계: 설치 완료

- 설치 완료 후 Tray 애플리케이션을 실행할 지 선택 할 수 있다.
- 설치 프로그램은 설치와 제거 기능만 제공된다. 제거 시에는 WSL 배포판과 Tray 애플리케이션을 제거되어, 배포판 내부의 데이터베이스 및 설정 파일은 모두 삭제되므로, 제거 전에 데이터베이스를 백업해야 한다.
- 업그레이드 및 기타 사용 방법은 Linux 와 동일하게 진행하면 된다.

참고

설치 프로그램을 통해 설치완료되면 시스템 재구동 시 CUBRID WSL 배포판과 CUBRID Server가 자동으로 실행하게 되어 있다. 시스템 재구동 시 자동 실행을 중단하려면 “설정 > 앱 > 시작 프로그램 > **cubrid_starter.exe**” 를 끄므로 설정 한다. 마찬가지로 지동 실행 목록에 Tray 애플리케이션을 등록한 경우 중단하려면 “설정 > 앱 > 시작 프로그램 > **cubrid_tray_app.exe**” 를 끄므로 설정 한다.

배포판 관련

- 설치 프로그램은 배포판 이름을 설정할 수 있으며, 배포판 이름은 Windows 명령 프롬프트에서 `wsl -l` 명령으로 확인 가능하다.
- 배포판의 CUBRID는 cubrid 사용자 계정으로 설치되며, cubrid 사용자의 홈 디렉터리는 `/home/cubrid` 이다.
- 배포판의 CUBRID 설치 경로는 `/home/cubrid/CUBRID` 이며, CUBRID 환경 변수는 `/home/cubrid/.cubrid.sh` 에 설정되어 있다.
- 배포판 안으로 접속하려면 Windows 명령 프롬프트에서 `wsl -d <배포판 이름> -u cubrid` 명령으로 접속 가능하다.

WSL 관련 사항

- WSL 관련 사항은 <https://learn.microsoft.com/ko-kr/windows/wsl/> 에서 확인 할 수 있다.

Windows에서의 설치와 실행

설치 전 확인 사항

Windows 버전의 CUBRID 데이터베이스를 설치하기 전에 다음 사항을 점검한다.

- 64비트

CUBRID는 64비트 버전만 지원한다. [내 컴퓨터] > [시스템 등록 정보] 창을 활성화하여 Windows 버전 비트를 확인할 수 있다. 64비트 Windows에 CUBRID 64비트 버전을 설치한다.

⚠ 경고

10.10이 32비트 Windows의 마지막 릴리스이다.

설치 과정

1단계: 설치 디렉터리 지정

2단계: 샘플 데이터베이스 생성

샘플 데이터베이스를 생성하려면 약 1.5GB의 디스크 공간이 필요하다.

3단계: 설치 완료

우측 하단에 CUBRID Service Tray가 나타난다.

참고

CUBRID는 설치하고 나면 시스템 재구동 시 자동으로 실행하게 되어 있다. 시스템 재구동 시 자동 실행을 중단하려면 “제어판 > 시스템 및 보안 > 관리 도구 > 서비스 > CUBRIDService” 에서 더블클릭한 후 나타난 팝업 창에서 시작 유형을 수동으로 변경한다.

설치 후 확인 사항

- CUBRID Service Tray 구동 여부

시스템을 시작할 때 CUBRID Service Tray가 자동으로 구동되지 않는다면 다음 사항을 확인하도록 한다.

- [시작 버튼] > [제어판] > [관리 도구] > [서비스]의 Task Scheduler가 시작되어 있는지 확인하고, 그렇지 않으면 Task Scheduler를 시작한다.
- [시작 버튼] > [모든 프로그램] > [시작프로그램]에 CUBRID Service Tray가 등록되어 있는지 확인하고, 그렇지 않으면 CUBRID Service Tray를 등록한다.

CUBRID 업그레이드

과거 버전의 CUBRID가 이미 설치된 환경에 새로운 버전의 CUBRID를 설치하는 경우, 시스템 트레이에서 [CUBRID Service Tray] > [Exit]를 선택하여 운영 중인 서비스를 종료한 후 과거 버전의 CUBRID를 제거해야 한다. “데이터베이스와 설정 파일을 모두 삭제하겠습니까?”라고 묻는 대화 상자가 나타나면, 과거 버전의 데이터베이스가 삭제되지 않도록[아니오]를 클릭한다.

과거 버전에서 새 버전으로 데이터베이스를 업그레이드하는 방법에 대한 보다 자세한 내용은 [업그레이드](#)를 참고한다.

환경 설정

서비스 포트 등 사용자 환경에 맞춰 설정을 변경하려면 **%CUBRID%\conf** 디렉터리에서 다음 설정 파일의 파라미터 값을 변경한다. 방화벽이 설정되어 있다면 CUBRID에서 사용하는 포트들을 열어두어야(open) 한다. CUBRID가 사용하는 포트에 대한 자세한 내용은 [포트 설정](#)을 참고한다.

- **cm.conf**

CUBRID 매니저용 설정 파일이다. **cm_port** 는 매니저 서버 프로세스, 매니저 서버 프로세스가 사용하는 포트 기본값은 **8001** 이다.

- **cubrid.conf**

서버 설정용 파일로, 운영하려는 데이터베이스의 메모리, 동시 사용자 수에 따른 스레드 수, 브로커와 서버 사이의 통신 포트 등을 설정한다. **cubrid_port_id** 는 마스터 프로세스가 사용하는 포트 기본값은 **1523** 이다. 자세한 내용은 [cubrid.conf 설정 파일과 기본 제공 파라미터](#)를 참조한다.

· cubrid_broker.conf

브로커 설정용 파일로, 운영하려는 브로커가 사용하는 포트, 응용서버(CAS) 수, SQL LOG 등을 설정한다. **BROKER_PORT** 는 브로커가 사용하는 포트이며, 실제 JDBC와 같은 드라이버에서 보는 포트는 해당 브로커의 포트이다. **APPL_SERVER_PORT** 는 Windows에서만 추가하는 파라미터로, 브로커 응용 서버(CAS)가 사용하는 포트이다. 기본값은 **BROKER_PORT + 1**이다. **APPL_SERVER_PORT** 값을 기준으로 1씩 더한 포트들이 CAS 개수만큼 사용된다. 예를 들어 **APPL_SERVER_PORT** 값이 35000이고 **MAX_NUM_APPL_SERVER** 값에 의한 CAS의 최대 개수가 50이면 CAS에서 listen하는 포트는 35000, 35001, ..., 35049이다. 자세한 내용은 [브로커별 파라미터](#)를 참조한다.

CCI_DEFAULT_AUTOCOMMIT 브로커 파라미터는 2008 R4.0부터 지원하기 시작했고, 이때 기본값은 **OFF** 였다가 2008 R4.1부터는 기본값이 **ON** 으로 바뀌었다. 따라서 2008 R4.0에서 2008 R4.1 이상 버전으로 업그레이드하는 사용자는 이 값을 OFF로 바꾸거나, 응용 프로그램의 함수에서 자동 커밋 모드를 OFF로 설정해야 한다.

CUBRID 인터페이스 설치

<https://www.cubrid.org/downloads> 에서 CCI, JDBC, PHP, ODBC, OLE DB, ADO.NET, Ruby, Python 및 Node.js와 같은 인터페이스 모듈을 다운로드할 수 있다.

각 드라이버에 대한 간단한 설명은 [API 레퍼런스](#)를 참고한다.

CUBRID 도구 설치

<https://www.cubrid.org/downloads> 에서 CUBRID Manager 및 CUBRID Migration Toolkit을 비롯한 다양한 도구를 다운로드할 수 있다.

압축 파일로 설치하기

Linux에서 tar.gz 파일로 CUBRID 설치

설치 전 확인 사항

Linux 버전의 CUBRID 데이터베이스를 설치하기 전에 다음 사항을 점검한다.

· glibc 버전

glibc 2.3.4 버전 이상만 지원한다. glibc 버전은 다음과 같은 방법으로 확인한다.

```
% rpm -q glibc
```

· 64비트 여부

10.0 이후 CUBRID는 64비트 버전만 지원한다. Linux 버전은 다음과 같은 방법으로 확인한다.

```
% uname -a
Linux host_name 2.6.18-53.1.14.el5xen #1 SMP Wed Mar 5 12:08:17 EST
2008 x86_64 x86_64 x86_64 GNU/Linux
```

64비트 Linux에서는 CUBRID 64비트 버전을 설치한다.

· 추가로 설치할 라이브러리

- Curses Library (rpm -q ncurses)

CUBRID는 Curses 라이브러리 버전 5와 함께 패키징된다. 시스템에 최신 버전이 있고 다운 그레이드할 수 없는 경우 ncurses-compat-libs 패키지를 설치해야할 수 있다.

- gcrypt Library (rpm -q libgcrypt)
- stdc++ Library (rpm -q libstdc++)

· /etc/hosts 파일에 호스트 이름과 IP 주소 매핑이 정상인지 확인하기

호스트 이름과 이에 맞는 IP 주소가 비정상적으로 매핑되어 있으면 DB 서버를 구동할 수 없으므로, 정상적으로 매핑되어 있는지 확인한다.

설치 과정

설치 디렉터리 지정

- 압축 파일을 설치하려는 경로에 풀어 놓는다.

```
tar xvfz CUBRID-11.4-latest-Linux.x86_64.tar.gz /home1/
cub_user/
```

/home1/cub_user/ 이하에 CUBRID 디렉터리가 생기고 그 이하에 파일이 생성된다.

환경 변수 설정

1. 사용자의 홈 디렉터리(/home1/cub_user) 이하에서 자동으로 실행되는 쉘 스크립트에 아래의 환경 변수를 추가한다.

\$CUBRID_DATABASES 변수에 설정된 디렉터리 생성이 필요하다. 적절한 권한이 있는 임의의 디렉터리를 지정할 수 있다.

다음은 bash 쉘로 수행하는 경우 .bash_profile에 다음을 추가하는 예이다.

```
export CUBRID=/home1/cub_user/CUBRID
export CUBRID_DATABASES=$CUBRID/databases
```

2. **CLASSPATH** 환경 변수에 CUBRID JDBC 라이브러리 파일 이름을 추가한다.

```
export CLASSPATH=$CUBRID/jdbc/cubrid_jdbc.jar:$CLASSPATH
```

3. **PATH** 환경 변수에 CUBRID bin 디렉터리를 추가한다.

```
export PATH=$CUBRID/bin:$PATH
```

DB 생성

- 콘솔 창에서 DB를 생성할 디렉터리로 이동해서 DB를 직접 생성한다.

```
cd $CUBRID_DATABASES
mkdir testdb
cd testdb
cubrid createdb --db-volume-size=128M --log-volume-size=128M testdb en_US
```

부팅 시 자동 시작

- **\$CUBRID/share/init.d** 디렉터리에 cubrid라는 스크립트가 포함되어 있다. 이 파일 안의 **CUBRID_USER** 환경 변수 값을 CUBRID를 설치한 Linux 계정으로 변경한 후, **/etc/init.d**에 등록하면 **service**나 **chkconfig** 명령을 사용하여 Linux 시스템 구동 시 CUBRID를 자동으로 구동할 수 있다.

DB 자동 구동

- 부팅 시 생성한 DB가 구동되게 하려면 **\$CUBRID/conf/cubrid.conf** 에서 다음을 수정한다.

```
[service]
service=server, broker, manager
server=testdb
```

- **service** 파라미터에는 자동으로 구동할 프로세스들을 지정한다.
- **server** 파라미터에는 자동으로 구동할 DB 이름을 지정한다.

CUBRID 설치 이후 환경 설정, 도구 설치, 인터페이스 설치 등은 [Linux에서의 설치와 실행](#)을 확인하도록 한다.

Windows에서 zip 파일로 CUBRID 설치 설치 전 확인 사항

Windows 버전의 CUBRID 데이터베이스를 설치하기 전에 다음 사항을 점검한다.

- 64비트 여부

CUBRID는 64비트 버전만 지원한다. [내 컴퓨터] > [시스템 등록 정보] 창을 활성화하여 Windows 버전 비트를 확인할 수 있다. 64비트 Windows에 CUBRID 64비트 버전을 설치한다.

⚠ 경고

10.10이 32비트 Windows의 마지막 릴리스이다.

설치 과정

설치 디렉터리 지정

- 압축 파일을 설치하려는 경로에 풀어 놓는다.

```
C:\CUBRID
```

- **\$CUBRID_DATABASES** 변수에 설정된 디렉터리 생성이 필요하다. 적절한 권한이 있는 임의의 디렉터리를 지정할 수 있다.

환경 변수 설정

1. [시작 버튼] > [컴퓨터] > (오른쪽 마우스 버튼 클릭) > [속성] -> [고급 시스템 설정] > [환경변수]를 선택한다.
2. 시스템 변수 항목에 [새로 만들기]를 클릭한 후 아래와 같이 시스템 변수를 추가한다.

```
CUBRID = C:\CUBRID
CUBRID_DATABASES = %CUBRID%\databases
```

3. **CLASSPATH** 시스템 변수에 CUBRID JDBC 라이브러리 파일 이름을 추가한다.

```
%CUBRID%\jdbc\cubrid_jdbc.jar
```

4. **Path** 시스템 변수에 CUBRID bin 과 CUBRID ccibin 디렉터리를 추가한다.

```
%CUBRID%\bin;%CUBRID%\cci\bin
```

참고

CUBRID를 사용하기 위한 환경변수 설정은 아래의 Batch file(cubrid_env.bat)을 이용하여 설정을 편리하게 할 수 있다. C:\CUBRID\shard\windows_scripts\cubrid_env.bat

DB 생성

- cmd 명령으로 콘솔 창을 띄운 후 DB를 생성할 디렉터리로 이동해서 DB를 직접 생성한다.

```
cd C:\CUBRID\databases
md testdb
cd testdb
c:\CUBRID\databases\testdb>cubrid createdb --db-volume-size=128M --log-volume-size=128M testdb en_US
```

부팅 시 자동 시작

- 설치한 CUBRID가 Windows 시스템 부팅 시 자동으로 시작되게 하려면 CUBRID 서비스가 먼저 Windows 서비스에 등록되어야 한다.

1. CUBRID 서비스를 Windows 서비스에 등록한다.

```
C:\CUBRID\bin\ctrlService.exe -i C:\CUBRID\bin
```

2. CUBRID 서비스를 구동/정지하는 방법은 아래와 같다.

```
C:\CUBRID\bin\ctrlService.exe -start/-stop
```

DB 자동 구동

- Windows 부팅 시 DB가 구동되게 하려면 C:\CUBRIDconf\cubrid.conf에서 다음을 수정한다.

```
[service]
service=server, broker, manager
server=testdb
```

- service 파라미터에는 자동으로 구동할 프로세스들을 지정한다.
- server 파라미터에는 자동으로 구동할 DB 이름을 지정한다.

서비스에서 제거

- 등록된 CUBRID Service를 제거하려면 다음을 수행한다.

```
C:\CUBRID\bin\ctrlService.exe -u
```

CUBRID Service Tray 등록

zip 파일로 CUBRID를 설치하는 경우 CUBRID Service Tray가 자동으로 등록되지 않으므로, 이를 사용하려면 수동으로 등록하는 절차가 필요하다.

1. C:\CUBRID\bin\CUBRID_Service_Tray.exe 파일의 바로 가기를 시작 > 모든프로그램 > 시작프로그램에 생성한다.
2. 시작 > 보조 프로그램 > 실행 창에서 regedit를 입력하면 레지스트리 편집기가 실행된다.
3. 컴퓨터 > HKEY_LOCAL_MACHINE > SOFTWARE에 CUBRID 폴더를 생성한다.
4. 생성한 CUBRID 폴더에 cmclient 폴더를 생성(새로 만들기 > 키)하고 아래의 항목을 추가(새로 만들기 > 문자열 값)한다.

이름	종류	데이터
ROOT_PATH	REG_SZ	C:\CUBRID\cubridmanager

5. 생성한 CUBRID 폴더에 cmserver 폴더를 생성(새로 만들기 > 키)하고 아래의 항목을 추가(새로 만들기 > 문자열 값)한다.

이름	종류	데이터
ROOT_PATH	REG_SZ	C:\CUBRID

6. 생성한 CUBRID 폴더에 CUBRID 폴더를 생성(새로 만들기 > 키)하고 아래의 항목을 추가(새로 만들기 > 문자열 값)한다.

이름	종류	데이터
ROOT_PATH	REG_SZ	C:\CUBRID

7. Windows를 재부팅하면 CUBRID Service Tray가 오른쪽 하단에 생긴다.

설치 후 확인 사항

- CUBRID Service Tray 구동 여부

시스템을 시작할 때 CUBRID Service Tray가 자동으로 구동되지 않는다면 다음 사항을 확인하도록 한다.

- [시작 버튼] > [제어판] > [관리 도구] > [서비스]의 Task Scheduler가 시작되어 있는지 확인하고, 그렇지 않으면 Task Scheduler를 시작한다.
- [시작 버튼] > [모든 프로그램] > [시작프로그램]에 CUBRID Service Tray가 등록되어 있는지 확인하고, 그렇지 않으면 CUBRID Service Tray를 등록한다.

CUBRID 설치 이후 환경 설정, 도구 설치, 인터페이스 설치 등은 **Windows에서의 설치와 실행**을 확인하도록 한다.

참고

zip 파일로 CUBRID를 설치한 경우, CUBRID 실행에 필요한 DLL 이 없어 실행이 되지 않을 수 있다. 이런 경우 Microsoft Visual C++ Redistributable 를 설치해야 한다.

· Microsoft Visual C++ Redistributable

<https://visualstudio.microsoft.com/ko/downloads/>

환경 변수 설정

CUBRID를 사용하기 위해서는 다음의 환경 변수들이 설정되어 있어야 한다. 필요한 환경 변수들은 CUBRID 시스템을 설치하면 자동으로 설정되나 필요에 의해서 사용자가 적절히 변경할 수도 있다.

CUBRID 환경 변수

- **CUBRID**: CUBRID 시스템이 설치된 위치를 지정하는 기본 환경 변수이다. CUBRID 시스템에 포함된 모든 프로그램은 이 환경 변수를 참조하므로 정확히 설정되어 있어야 한다.
- **CUBRID_DATABASES**: **databases.txt** 파일의 위치를 지정하는 환경 변수이다. CUBRID 시스템은 **\$CUBRID_DATABASES/databases.txt** 파일에 데이터베이스 볼륨들의 절대 경로를 저장 관리한다. **databases.txt** 파일을 참고한다.
- **CUBRID_MSG_LANG**: CUBRID 시스템이 명령어 사용법 메시지와 오류 메시지를 출력할 때 사용할 언어를 지정하는 환경 변수이다. 제품 설치 시 초기 설정 값은 없으며, 설정 값이 없으면 **en_US**로 설정된다. 환경변수로 설정할 수 있는 언어는 en_US, en_US.utf8, ko_KR.euckr 및 ko_KR.utf8 이다.

참고

- CUBRID Manager 사용자는 DB 서버 노드의 **CUBRID_MSG_LANG** 환경 변수를 **en_US** 로 설정해야만 데이터베이스 관련 작업 이후 출력되는 메시지를 정상적으로 확인할 수 있다. **CUBRID_MSG_LANG** 환경 변수가 **en_US** 가 아닌 경우 메시지는 비정상적으로 출력되지만 데이터베이스의 작업은 정상적으로 실행된다.
- 변경한 **CUBRID_MSG_LANG** 을 적용하려면 DB 서버 노드의 CUBRID 시스템이 반드시 재시작 (cubrid service stop; cubrid service start)되어야 한다.

- **CUBRID_TMP**: CUBRID DBMS를 구동에 필요한 임시 파일을 저장하기 위한 디렉터리를 설정하는 환경 변수이다. 임시 파일은 텍스트형과 프로세스간 통신을 위한 UNIX 도메인 소켓의 두 종류가 있다. 이 환경 변수가 설정된 경우 임시 파일은 환경 변수에 지정된 디렉터리에 생성된다.
 - 텍스트형 임시 파일은 데이터베이스 서버, csql 이외에도 **plandump** 등과 같은 유틸리티에서 생성하며 이 환경 변수가 설정되지 않으면 다음의 디렉터리를 사용한다.
 - Linux: /tmp
 - Windows: C:\Windows\TEMP
 - **UNIX 도메인 소켓 파일** 은 이 환경 변수가 설정되지 않은 경우 프로세스 별로 아래의 경로에 생성된다 (이 소켓 파일은 **Linux에만** 해당 되며 Windows에는 적용되지 않는다).
 - cub_master 프로세스: **/tmp** 디렉터리
 - cub_broker 프로세스: **\$CUBRID/var/CUBRID SOCK** 디렉터리
 - cub_pl 프로세스: **\$CUBRID/var/CUBRID SOCK** 디렉터리

참고

- **CUBRID_TMP** 환경 변수가 설정된 경우, cub_pl 프로세스(java VM 내장) 구동시 사용하는 **stored_procedure_vm_options** 중 임시 파일 저장 경로를 지정하는 **java.io.tmpdir** 의 설정값은 무시된다.
- Windows에서 이 환경 변수를 설정하기 위해서는 registry에 CUBRID_TMP key를 추가해야한다 (%CUBRID%\share\windows_scripts\cubrid_env.bat 참조).

CUBRID_TMP 의 값에는 다음과 같은 제약 사항이 있다.

- unix socket의 path의 최대 크기가 108이므로 다음과 같이 **\$CUBRID_TMP** 에 108보다 긴 경로를 입력하면 에러를 출력한다.

```
$ export CUBRID_TMP=/home1/testusr/cubrid=/tmp/
12345678901234567890123456789012345678901234567890123456789012345678
9012345678901234567890123456789
$ cubrid server start apricot

The $CUBRID_TMP is too long. (/home1/testusr/cubrid=/tmp/
12345678901234567890123456789012345678901234567890123456789012345678
9012345678901234567890123456789)
```

- 상대 경로를 입력하면 에러를 출력한다.

```
$ export CUBRID_TMP=./var
$ cubrid server start testdb

The $CUBRID_TMP should be an absolute path. (./var)
```

CUBRID_TMP 는 CUBRID가 사용하는 유닉스 도메인 소켓의 기본 경로에서 발생할 수 있는 다음 문제를 회피하기 위해 사용할 수 있다.

- **/tmp** 는 주로 Linux에서 임시 파일을 저장하는 공간으로, 시스템 관리자가 이 공간을 주기적으로 임의 삭제하는 경우 유닉스 도메인 소켓까지 삭제될 수 있다. 이러한 경우 **\$CUBRID_TMP** 를 **/tmp** 가 아닌 다른 경로로 설정한다.
- 유닉스 도메인 소켓 파일의 경로 최대 길이는 108인데, CUBRID의 설치 경로가 길어서 cub_broker 용 유닉스 도메인 소켓 파일을 저장하는 **\$CUBRID/var/CUBRID SOCK** 경로의 길이가 108을 넘는 경우 브로커를 구동할 수 없다. 따라서 **\$CUBRID_TMP** 를 108이 넘지 않는 경로로 설정해야 한다.

이들 환경변수는 CUBRID를 설치하면서 이미 설정되었으나, 설정을 확인하기 위해서는 다음 명령을 사용할 수 있다.

- Linux

```
% printenv CUBRID
% printenv CUBRID_DATABASES
% printenv CUBRID_MSG_LANG
% printenv CUBRID_TMP
```

- Windows

```
C:\> set CUBRID
```

OS 환경 변수

- PATH: Linux 환경에서 PATH 환경 변수에는 CUBRID 시스템의 실행 파일이 있는 디렉터리인 **\$CUBRID/bin** 이 포함되어 있어야 한다.
- LD_LIBRARY_PATH: Linux 환경에서는 **LD_LIBRARY_PATH** (혹은 **SHLIB_PATH** 나 **LIBPATH**) 환경 변수에 CUBRID 시스템의 동적 라이브러리 파일(libjvm.so)이 있는 디렉터리인 **\$CUBRID/lib** 과 **\$CUBRID/cci/lib** 이 포함되어 있어야 한다.
- Path: Windows 환경에서 Path 환경 변수에는 CUBRID 시스템의 실행 파일이 있는 디렉터리인 **%CUBRID%\bin**과 **%CUBRID%\cci\bin**이 포함되어 있어야 한다.

참고

- Linux 환경에서 다음과 같은 환경 변수 설정들을 통해 **CUBRID의 메모리 관리자(GLIB 메모리 관리자)**의 힙 메모리 할당 동작을 제어할 수 있다. 자세한 내용은 https://www.gnu.org/software/libc/manual/html_node/Malloc-Tunable-Parameters 참조

환경 변수	설 명	기본값
MALLOC_MMAP_MAX_	mmap() 시스템 호출을 사용하여 할당할 수 있는 최대 메모리 블록 수의 임계값 설정	65536
MALLOC_MMAP_THRESH_OLD_	mmap() 시스템 호출을 사용할지 여부를 결정하는 크기 기준 (임계값) 설정	128 KiB
MALLOC_TOP_PAD_	힙 메모리 상단에 유지해야 할 추가 메모리 여유 공간(패딩) 크기 설정	0
MALLOC_TRIM_THRESHOLD_	해제되는 메모리를 커널에 반환할지 여부를	128 KiB

환경 변수	설 명	기본값
	결정하는 임계값 설정, 임계값 초과시 반환	
MALLOC_ARENA_MAX	프로세스에서 생성할 수 있는 최대 아레나 (arena) 개수를 제한하는 설정	core 수의 8배

- 설치 패키지의 .cubrid.sh(또는 .cubrid.csh)에는 **MALLOC_TRIM_THRESHOLD_**값이 0으로 설정되어 있다. 그 외의 환경 변수들은 주석 처리되어 있음.

참고

- Linux 환경에서 **LD_PRELOAD** 환경 변수 설정을 통해 CUBRID의 메모리 관리자를 변경할 수 있다.
- 단, 메모리 관리자 교체에 의해 CUBRID 운영시 오동작이 발생할 수 있으므로 교체시 신중하게 검토해야 한다.
- 설치 패키지의 .cubrid.sh(또는 .cubrid.csh)에는 LD_PRELOAD 환경 변수가 주석 처리되어 있음.

```
예) export LD_PRELOAD=/usr/lib64/jemalloc.so.1
```

- **THP (Transparent Huge Pages)** 는 리눅스 커널 기능(2.6.38 이상) 환경에서 메모리 관리 효율을 높이기 위해 기본적으로 활성화되어 있는 기능이다. 작은 메모리 페이지(4KB)를 큰 페이지(보통 2MB)로 자동 결합하여 TLB(Translation Lookaside Buffer) 오버헤드를 줄여준다. 하지만 CUBRID와 같은 DBMS 환경에서는 THP의 자동 메모리 재배치 과정에서 **CPU 점유율 급증 (sys cpu)** 이나 **메모리 할당 지연** 이 발생하여 성능 저하의 원인이 될 수 있으므로, 안정적인 서비스를 위해 **THP 비활성화(never)** 를 권고한다.
- THP 활성화 상태 확인

```
예) [root]# cat /sys/kernel/mm/transparent_hugepage/enabled
```



```
[always] madvise never
비활성방법) [root]# echo never > /sys/kernel/mm/
transparent_hugepage/enabled
```

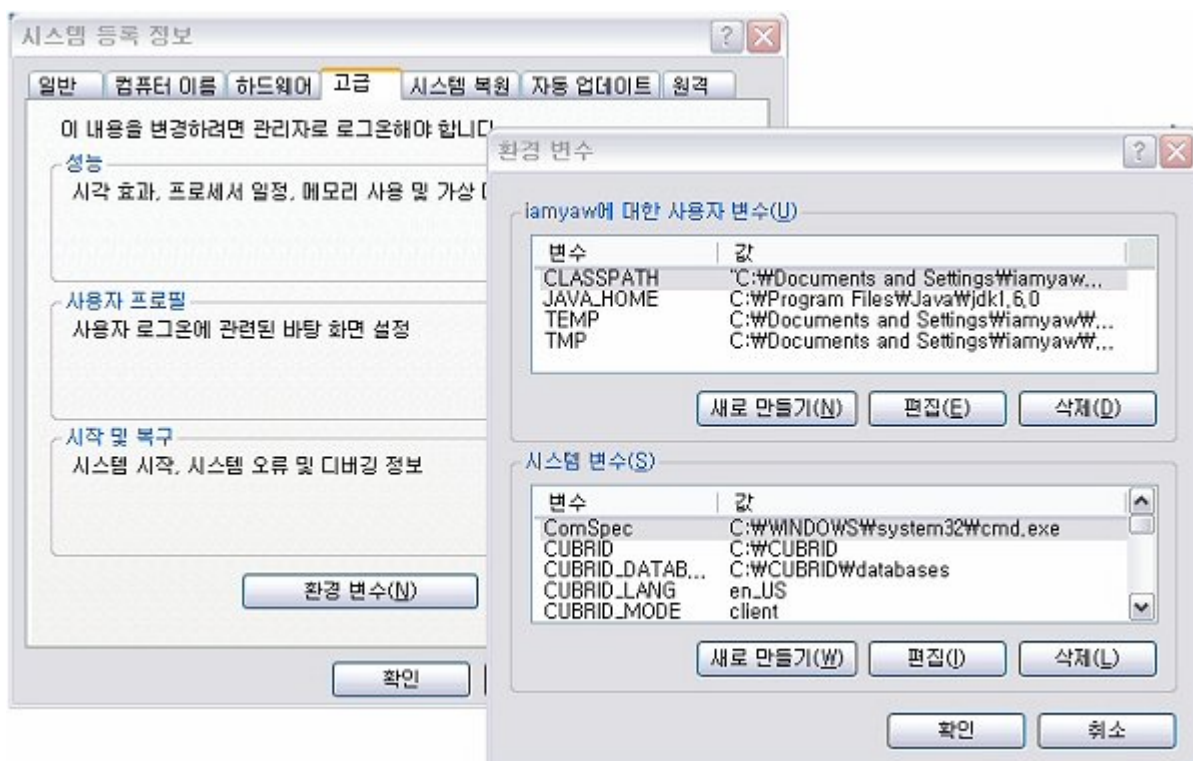
- THP 조각 모음(defrag) 상태 확인

```
예) [root]# cat /sys/kernel/mm/transparent_hugepage/defrag
always madvise [never]
비활성방법) [root]# echo never > /sys/kernel/mm/
transparent_hugepage/defrag
```

환경 변수 설정

Windows 환경인 경우

Windows 환경에서 CUBRID 시스템을 설치한 경우는 설치 프로그램이 필요한 환경 변수를 자동으로 설정한다. [시스템 등록 정보] 대화 상자의 [고급] 탭에서 [환경 변수]를 클릭하면 나타나는 [환경 변수] 대화 상자에서 확인할 수 있으며, [편집] 버튼을 통해 변경할 수 있다. Windows 환경에서 환경 변수를 변경하는 방법에 대한 상세한 정보는 Windows 도움말을 참고한다.



Linux 환경인 경우

Linux 환경에서 CUBRID 시스템을 설치한 경우는 설치 프로그램이 **.cubrid.sh** 혹은 **.cubrid.csh** 파일을 자동으로 생성하고 설치 계정의 셸 로그인 스크립트에서 자동으로 호출하도록 구성한다. 다음은 sh이나 bash 등을 사용하는 환경에서 생성된 **.cubrid.sh** 파일의 환경 변수 설정 내용이다.

```

CUBRID=/home1/cub_user/CUBRID
CUBRID_DATABASES=/home1/cub_user/CUBRID/databases
ld_lib_path=`printenv LD_LIBRARY_PATH`

if [ "$ld_lib_path" = "" ]
then
    LD_LIBRARY_PATH=$CUBRID/lib:$CUBRID/cci/lib
else
    LD_LIBRARY_PATH=$CUBRID/lib:$CUBRID/cci/lib:$LD_LIBRARY_PATH
fi

SHLIB_PATH=$LD_LIBRARY_PATH
LIBPATH=$LD_LIBRARY_PATH
PATH=$CUBRID/bin:$CUBRID/cubridmanager:$PATH

export CUBRID
export CUBRID_DATABASES
export LD_LIBRARY_PATH
export SHLIB_PATH
export LIBPATH
export PATH

```

언어 및 문자셋 설정

CUBRID 데이터베이스 관리 시스템은 사용할 언어와 문자셋을 DB 생성 시 DB 이름 뒤에 지정한다 (예: cubrid createdb testdb ko_KR.utf8). 현재 언어와 문자셋으로 설정될 수 있는 값은 다음과 같다.

- **en_US.iso88591**: 영어 ISO-8859-1 인코딩 (.iso88591 생략 가능)
- **ko_KR.euckr**: 한국어 EUC-KR 인코딩
- **ko_KR.utf8**: 한국어 UTF-8 인코딩(.utf8 생략 가능)
- **de_DE.utf8**: 독일어 UTF-8 인코딩
- **es_ES.utf8**: 스페인어 UTF-8 인코딩
- **fr_FR.utf8**: 프랑스어 UTF-8 인코딩
- **it_IT.utf8**: 이탈리아어 UTF-8 인코딩
- **ja_JP.utf8**: 일본어 UTF-8 인코딩
- **km_KH.utf8**: 캄보디아어 UTF-8 인코딩
- **tr_TR.utf8**: 터키어 UTF-8 인코딩(.utf8 생략 가능)
- **vi_VN.utf8**: 베트남어 UTF-8 인코딩

- **zh_CN.utf8**: 중국어 UTF-8 인코딩
- **ro_RO.utf8**: 루마니아어 UTF-8 인코딩

CUBRID의 언어와 문자셋 설정은 데이터를 쓰거나 읽을 때 영향을 미치며, 프로그램들이 출력하는 메시지에도 해당 언어가 사용된다.

문자셋, 로캘 및 콜레이션 설정과 관련된 자세한 내용은 **다국어 개요** 을 참고한다.

포트 설정

포트가 개방되어 있지 않은 환경에서 사용하는 경우, CUBRID가 사용하는 포트들을 개방해야 한다.

다음은 CUBRID가 사용하는 포트에 대해 하나의 표로 정리한 것이다. 각 포트는 상대방의 접속을 대기하는 listener 쪽에서 개방되어야 한다.

Linux 방화벽에서 특정 프로세스에 대한 포트를 개방하려면 해당 방화벽 프로그램의 설명을 따른다.

Windows에서 임의의 가용 포트를 사용하는 경우는 어떤 포트를 개방할 지 알 수 없으므로 Windows 메뉴의 “제어판” 검색창에서 “방화벽”을 입력한 후, “Windows 방화벽 > Windows 방화벽을 통해 프로그램 또는 기능 허용”에서 포트 개방을 원하는 프로그램을 추가한다.

Windows에서 특정 포트를 지정하기 번거로운 경우에도 이 방법을 사용할 수 있다. 일반적으로 Windows 방화벽에서 특정 프로그램을 지정하지 않고 포트를 여는 것보다 허용되는 프로그램 목록에 프로그램을 추가하는 것이 보다 안전하므로 이 방식을 권장한다.

- cub_broker에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_broker.exe”를 추가한다.
- CAS에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_cas.exe”를 추가한다.
- cub_master에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_master.exe”를 추가한다.
- cub_server에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_server.exe”를 추가한다.
- CUBRID 매니저에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_cmserver.exe”를 추가한다.
- CUBRID 프로시저 언어 서버에 대한 모든 포트를 개방하려면 “%CUBRID%\bin\cub_pl.exe”를 추가한다.

브로커 장비 또는 DB 서버 장비에서 Linux용 CUBRID를 사용한다면 Linux 포트가 모두 개방되어 있어야 한다. 브로커 장비 또는 DB 서버 장비에서 Windows용 CUBRID를 사용한다면 Windows 포트가 모두 개방되어 있거나, 관련 프로세스들이 모두 Windows 방화벽에서 허용되는 목록에 추가되어 있어야 한다.

구분	listener	requester	Linux 포트	Windows 포트	방화벽 포트 설정	설명
기본 사용	cub_broker	application	BROKER_PORT	BROKER_PORT	개방 (open)	일회성 연결
	CAS	application	BROKER_PORT	APPL_SERVER_PORT ~ (APP_SERVER_PORT + CAS 개수 - 1)	개방	연결 유지
	cub_master	CAS	cubrid_port_id	cubrid_port_id	개방	일회성 연결
	cub_server	CAS	cubrid_port_id	임의의 가용 포트	Linux: 개방 Windows: 프로그램	연결 유지
	클라이언트 장비(*)	cub_server	ECHO(7)	ECHO(7)	개방	주기적 연결
HA 사용	서버 장비(**)	CAS, CSQL	ECHO(7)	ECHO(7)	개방	주기적 연결
	cub_broker	application	BROKER_PORT	미지원	개방	일회성 연결
	CAS	application	BROKER_PORT	미지원	개방	연결 유지
	cub_master	CAS		미지원	개방	

구분	listener	requester	Linux 포트	Windows 포트	방화벽 포트 설정	설명
			cubrid_port_id			일회성 연결
	cub_master (slave)	cub_master (master)	ha_port_id	미지원	개방	주기적 연결, heartbeat 확인
	cub_master (master)	cub_master (slave)	ha_port_id	미지원	개방	주기적 연결, heartbeat 확인
	cub_server	CAS	cubrid_port_id	미지원	개방	연결 유지
	클라이언트 장비(*)	cub_server	ECHO(7)	미지원	개방	주기적 연결
	서버 장비(**)	CAS, CSQL, copylogdb, applylogdb	ECHO(7)	미지원	개방	주기적 연결
Manager 사용	Manager 서버	application	8001	8001	개방	
PL 서버 사용	cub_pl	cub_server	stored_procedure_port	stored_procedure_port	개방	연결 유지

(*): CAS, CSQL, copylogdb, 또는 applylogdb 프로세스가 존재하는 장비

(**): cub_server가 존재하는 장비

각 구분 별 상세 설명은 아래와 같다.

CUBRID 기본 사용 포트

접속 요청을 기다리는(listening) 프로세스들을 기준으로 각 OS 별로 필요한 포트를 정리하면 다음과 같으며, 각 포트는 listener 쪽에서 개방되어야 한다.

listener	requester	Linux port	Windows port	방화벽 포트 설정	설명
cub_broker	application	BROKER_PORT	BROKER_PORT	개방(open)	일회성 연결
CAS	application	BROKER_PORT	APPL_SERVER_PORT ~ (APPL_SERVER_PORT + CAS 개수 - 1)	개방	연결 유지
cub_master	CAS	cubrid_port_id	cubrid_port_id	개방	일회성 연결
cub_server	CAS	cubrid_port_id	임의의 가용 포트	Linux: 개방 Windows: 프로그램	연결 유지
클라이언트 장비(*)	cub_server	ECHO(7)	ECHO(7)	개방	주기적 연결
서버 장비(**)	CAS, CSQL	ECHO(7)	ECHO(7)	개방	주기적 연결

(*): CAS 또는 CSQL 프로세스가 존재하는 장비

(**): cub_server가 존재하는 장비

참고

Windows에서는 CAS가 cub_server에 접근할 때 사용할 포트를 임의로 정하므로 개방할 포트를 정할 수 없다. 따라서 “Windows 방화벽 > 허용되는 프로그램”에 “%CUBRID%\bin\cub_server.exe”을 추가해야 한다.

서버 프로세스(cub_server)와 이에 접속하는 클라이언트 프로세스들(CAS, CSQL) 사이에서 상대 노드가 정상 동작하는지 ECHO(7) 포트를 통해 서로 확인하므로, 방화벽 존재 시 ECHO(7) 포트를 개방해야 한다. ECHO 포트를 서버와 클라이언트 양쪽 다 개방할 수 없는 상황이라면 cubrid.conf의 `check_peer_alive` 파라미터 값을 none으로 설정한다.

다음은 각 프로세스 간 연결 관계를 나타낸 것이다.

```
application - cub_broker
             -> CAS   - cub_master
                   -> cub_server
```

- application: 응용 프로세스
- cub_broker: 브로커 서버 프로세스. application이 연결할 CAS를 선택하는 역할을 수행.
- CAS: 브로커 응용 서버 프로세스. application과 cub_server를 중계.
- cub_master: 마스터 프로세스. CAS가 연결할 cub_server를 선택하는 역할을 수행.
- cub_server: DB 서버 프로세스

프로세스 간 관계 기호 및 의미는 다음과 같다.

- - 기호: 최초 한 번만 연결됨을 나타낸다.
- ->, <- 기호: 연결이 유지됨을 나타내며, -> 의 오른쪽 또는 <-의 왼쪽이 화살을 받는 쪽이다. 화살을 받는 쪽이 처음에 상대 프로세스의 접속을 기다리는(listening) 쪽을 나타낸다.
- (master): HA 구성에서 master 노드를 나타낸다.
- (slave): HA 구성에서 slave 노드를 나타낸다.

다음은 응용 프로그램과 DB 사이의 연결 과정을 순서대로 나열한 것이다.

1. application이 cubrid_broker.conf에 설정된 브로커 포트(BROKER_PORT)를 통해 cub_broker와 연결을 시도한다.
2. cub_broker는 연결 가능한 CAS를 선택한다.
3. application과 CAS가 연결된다.

Linux에서는 application과 브로커의 네트워크 연결을 브로커가 CAS에 그대로 전달한다. 따라서 application이 CAS와 연결을 위한 별도의 네트워크 포트는 필요하지 않다. Windows에서는 Application이 브로커에 연결되면 브로커가 가용한 CAS에 접속하기 위한 네트워크 포트 번호를 application에 전달한다. Application은 브로커와의 연결을 종료한 후, 전달 받은 네트워크 포트 번호로 CAS와 접속하게 된다. APPL_SERVER_PORT의 값이 설정되지 않으면 첫번째 CAS가 사용하는 네트워크 포트는 BROKER_PORT + 1이 된다.

예를 들어 Windows에서 BROKER_PORT가 33000이고 APPL_SERVER_PORT가 설정되지 않았으면 application과 CAS 사이에 사용되는 포트는 다음과 같다.

- application이 CAS(1)과 접속하는 포트 : 33001
- application이 CAS(2)와 접속하는 포트 : 33002
- application이 CAS(3)와 접속하는 포트 : 33003

4. CAS는 cubrid.conf에 설정된 cubrid_port_id 포트를 통해 cub_master에게 cub_server로의 연결을 요청한다.

5. CAS와 cub_server가 연결된다.

Linux에서는 CAS가 유닉스 도메인 소켓을 통해 cub_server와 연결되므로 cubrid_port_id 포트를 사용한다. Windows에서는 유닉스 도메인 소켓을 사용할 수 없으므로 임의의 가용 포트를 통해 cub_server와 연결된다. Windows에서 DB server를 운용한다면 브로커 장비와 DB 서버 장비 사이에서는 임의의 가용 포트를 사용하므로, 두 장비 사이에서 방화벽이 해당 프로세스에 대한 포트를 막게 되면 정상 동작을 보장할 수 없게 된다는 점에 주의한다.

6. 이후 CAS는 application이 종료되어도 CAS가 재시작되지 않는 한 cub_server와 연결을 유지한다.

CUBRID HA 사용 포트

CUBRID HA는 Linux 환경에서만 지원한다.

접속 요청을 기다리는(listening) 프로세스들을 기준으로 각 OS 별로 필요한 포트를 정리하면 다음과 같으며, 각 포트는 listener 쪽에서 개방되어야 한다.

listener	requester	Linux port	방화벽 포트 설정	설명
cub_broker	application	BROKER_PORT	개방(open)	일회성 연결
CAS	application	BROKER_PORT	개방	연결 유지

listener	requester	Linux port	방화벽 포트 설정	설명
cub_master	CAS	cubrid_port_id	개방	일회성 연결
cub_master (slave)	cub_master (master)	ha_port_id	개방	주기적 연결, heartbeat 확인
cub_master (master)	cub_master (slave)	ha_port_id	개방	주기적 연결, heartbeat 확인
cub_server	CAS	cubrid_port_id	개방	연결 유지
클라이언트 장비 (*)	cub_server	ECHO(7)	개방	주기적 연결
서버 장비(**)	CAS, CSQL, copylogdb, applylogdb	ECHO(7)	개방	주기적 연결

(*): CAS, CSQL, copylogdb, 또는 applylogdb 프로세스가 존재하는 장비

(**): cub_server가 존재하는 장비

서버 프로세스(cub_server)와 이에 접속하는 클라이언트 프로세스들(CAS, CSQL, copylogdb, applylogdb 등) 사이에서 상대 노드가 정상 동작하는지 ECHO(7) 포트를 통해 서로 확인하므로, 방화벽 존재 시 ECHO(7) 포트를 개방해야 한다. ECHO 포트를 서버와 클라이언트 양쪽 다 개방할 수 없는 상황이라면 cubrid.conf의 `ref:check_peer_alive <check_peer_alive>` 파라미터 값을 none으로 설정한다.

다음은 각 프로세스 간 연결 관계를 나타낸 것이다.

```
application - cub_broker
              -> CAS    - cub_master(master) <-> cub_master(slave)
                        -> cub_server(master)   cub_server(slave) <-
applylogdb(slave)
```

copylogdb(slave)

- cub_master(master): CUBRID HA 구성에서 master 노드에 있는 마스터 프로세스. 상대 노드가 살아있는지 확인하는 역할을 수행.
- cub_master(slave): CUBRID HA 구성에서 slave 노드에 있는 마스터 프로세스. 상대 노드가 살아있는지 확인하는 역할을 수행.
- copylogdb(slave): CUBRID HA 구성에서 slave 노드에 있는 복제 로그 복사 프로세스
- applylogdb(slave): CUBRID HA 구성에서 slave 노드에 있는 복제 로그 반영 프로세스

master 노드에서 slave 노드로의 복제 과정 파악이 용이하게 하기 위해 위에서 master 노드의 applylogdb, copylogdb와 slave 노드의 CAS는 생략했다.

프로세스 간 관계 기호 및 의미는 다음과 같다.

- - 기호: 최초 한 번만 연결됨을 나타낸다.
- ->, <- 기호: 연결이 유지됨을 나타내며, -> 의 오른쪽 또는 <-의 왼쪽이 화살을 받는 쪽이다. 화살을 받는 쪽이 처음에 상대 프로세스의 접속을 기다리는(listening) 쪽을 나타낸다.
- (master): HA 구성에서 master 노드를 나타낸다.
- (slave): HA 구성에서 slave 노드를 나타낸다.

응용 프로그램과 DB 사이의 연결 과정은 **CUBRID 기본 사용 포트**와 동일하다. 여기에서는 CUBRID HA에 의해 1:1로 master DB와 slave DB를 구성할 때 master 노드와 slave 노드 사이의 연결 과정에 대해서만 설명한다.

1. cub_master(master)와 cub_master(slave) 사이에는 cubrid_ha.conf에 설정된 ha_port_id를 사용한다.
2. copylogdb(slave)는 slave 노드에 있는 cubrid.conf의 cubrid_port_id에 설정된 포트를 통해 cub_master(master)에게 master DB로의 연결을 요청하여, 최종적으로 cub_server(master)와 연결하게 된다.
3. applylogdb(slave)는 slave 노드에 있는 cubrid.conf의 cubrid_port_id에 설정된 포트를 통해 cub_master(slave)에게 slave DB로의 연결을 요청하여, 최종적으로 cub_server(slave)와 연결하게 된다.

master 노드에서도 applylogdb와 copylogdb가 동작하는데, master 노드가 절체(failover)로 인해 slave 노드로 변경될 때를 대비하기 위함이다.

CUBRID 매니저 서버 사용 포트

다음 표는 접속 요청을 기다리는(listening) 프로세스들을 기준으로 CUBRID Manager 서버가 사용하는 포트이며, 이것은 OS의 종류와 상관없이 동일하다.

listener	requester	port	방화벽 존재 시 포트 설정
Manager server	application	8001	개방(open)

- CUBRID 매니저 클라이언트가 CUBRID 매니저 서버 프로세스에 접속할 때 사용하는 포트는 cm.conf의 **cm_port**이며 기본값은 8001이다.

CUBRID 프로시저 언어 서버 사용 포트

다음 표는 CUBRID 프로시저 언어 서버 (cub_pl)이 데이터베이스 서버 (cub_server) 와 TCP 소켓으로 통신할 때 사용하는 포트이다.

Listener	Requester	Port	방화벽 존재 시 포트 설정
cub_pl	cub_server	stored_procedure_port	개방(open)

OS 종류와 **cubrid.conf**의 **stored_procedure_uds**의 설정값에 따른 동작

- Linux
 - **stored_procedure_uds**가 **yes**인 경우 유닉스 도메인 소켓을 사용하며 TCP 소켓을 사용하지 않는다.
 - **stored_procedure_uds**가 **no**인 경우 TCP 소켓을 사용하며 포트는 **stored_procedure_port**의 설정값을 사용한다.
- Windows
 - Windows에서는 유닉스 도메인 소켓을 지원하지 않는다.
 - 따라서 **stored_procedure_uds**에 상관없이 TCP 소켓을 사용하며 포트는 **stored_procedure_port**의 설정값을 사용한다.

참고

CUBRID 프로시저 언어 서버 (cub_pl)가 cub_server 와 통신할 때 사용하는 포트는 **cubrid.conf**의 **stored_procedure_port**이며 기본값은 0으로, 사용 가능한 임의의 가용 포트가 할당됨을 의미한다.

업그레이드

업그레이드 시 주의 사항

변경

11.3 버전과의 동작 차이에 대해서는 릴리스 노트의 **11.4 변경사항**을 반드시 참고한다.

기존 환경 설정 파일 보관

- 기존 버전의 **\$CUBRID/conf** 디렉터리의 환경 설정 파일(**cubrid.conf**, **cubrid_broker.conf**, **cm.conf**)과 **\$CUBRID/DATABASES** 디렉터리의 DB 위치 정보 파일(**databases.txt**)을 보관한다.

새로 추가된 예약어 검사

- CUBRID 설치 패키지에 포함 또는 http://ftp.cubrid.org/CUBRID_Engine/11.4/에서 배포되는 CUBRID 11.4 버전용 예약어 검출 스크립트인 **check_reserved.sql**을 이용하여 예약어 사용 여부를 검사할 수 있으며, 예약어로 지정된 식별자를 사용하고 있을 경우 식별자를 수정해야 한다. **식별자** 절을 참고한다.

환경 변수 CUBRID_MSG_LANG 설정

- **CUBRID_LANG** 과 **CUBRID_CHARSET** 환경 변수는 더 이상 사용되지 않으며, 데이터베이스를 생성할 때 언어 및 문자셋을 반드시 지정해야 한다. 유틸리티 메시지 및 오류 메시지를 출력할 때는 **CUBRID_MSG_LANG** 환경 변수를 사용하며 설정하지 않으면 데이터베이스를 생성할 때 지정한 언어 및 문자셋을 따른다.

스키마 변환

- 9.0 Beta 미만 버전에서 ISO-8859-1이 아닌 EUC-KR, UTF-8 문자셋을 사용하던 사용자는 스키마를 반드시 변경해야 한다. 9.0 Beta 미만 버전에서는 **CHAR**, **VARCHAR** 의 자릿수(precision)를 바이트 크기로 지정했으나 9.0 Beta 버전부터는 글자의 개수로 지정한다.

로캘 추가/유지

- 추가하고 싶은 로캘이 있는 경우 **\$CUBRID/conf/cubrid_locales.txt** 파일에 해당 로캘을 추가한 후 **make_locale** 스크립트를 실행한다. **로캘 설정**을 참고한다.

- 기존 버전의 로캘을 유지하고 싶은 경우에도 **\$CUBRID/conf/cubrid_locales.txt** 파일에 기존 버전의 로캘을 추가하고 **make_locale** 스크립트를 실행한 후 **cubrid createdb** 명령 실행 시 기존 로캘을 지정한다.

DB 마이그레이션

- CUBRID 11.4는 CUBRID 11.3 및 이전 버전들과 DB 볼륨이 호환되지 않으므로, cubrid unloaddb/loaddb 유틸리티를 이용해서 마이그레이션을 해야 한다. 자세한 절차는 **DB 마이그레이션** 을 참고하면 된다.
- CUBRID 2008 R3.1부터 GLO를 지원하지 않으며 LOB 타입이 GLO 기능을 대체하게 되었으므로, GLO를 이용한 응용 및 스키마는 LOB 타입에 맞게 수정해야 한다.

참고

2008 R4.0 이하 버전에서 TIMESTAMP '1970-01-01 00:00:00'(GMT)는 TIMESTAMP의 최소값이지만, 2008 4.1 이상 버전에서는 zerodate로 인식되며 TIMESTAMP '1970-01-01 00:00:01'(GMT)이 TIMESTAMP의 최소값이다.

볼륨 호환성 표

Vesion	11.4	11.3	11.2	10.0	10.2	10.1	10.0
11.4	O						
11.3		O	O				
11.2		O	O				
11.0				O			
10.2					O		
10.1						O	
10.0							O

복제 또는 HA 환경 재구성

- CUBRID 2008 R4.0부터는 복제 기능을 더 이상 지원하지 않으므로, 예전의 복제 기능을 사용하는 시스템에서는 DB 마이그레이션 이후 HA 환경으로 재구성할 것을 권장한다. 또한, CUBRID 2008 R2.0 및 R2.1에서 제공된 Linux Heartbeat 기반의 HA 기능을 사용하는 시스템도 보다 안정적인 운영을 위해 DB 마이그레이션 이후 CUBRID Heartbeat 기반의 HA 환경으로 재구성해야 한다. (아래의 **HA 환경에서 DB 마이그레이션** 참고)
- HA 환경 구성은 매뉴얼의 **CUBRID HA**를 참고하여 재설정해야 한다.

저장 함수/프로시저

- 저장 함수/프로시저를 사용하기 위해 CUBRID 프로시저 언어 서버(cub_pl)가 기본적으로 구동되며, 데이터베이스 서버가 구동/관리하도록 변경되었다. 따라서 프로시저 언어 서버가 종료되어도 데이터베이스 서버에 의해 자동으로 재구동된다.
- CUBRID 프로시저 언어 서버를 구동하지 않으려면 **cubrid.conf**의 **stored_procedure**파라미터를 **no**로 설정후 재시작해야 한다. 이 경우, PL/CSQL과 Java 저장 프로시저 기능을 사용할 수 없다.

CUBRID 9.2/9.3/10.0/10.1/10.2/11.0/11.2/11.3 에서 CUBRID 11.4 로 업그레이드하기

CUBRID 9.2/9.3/10.0/10.1/10.2/11.0/11.2/11.3 버전을 사용 중인 사용자는 다른 디렉터리에 11.4를 설치하고 데이터베이스를 11.4로 마이그레이션한 후 이전 환경 설정 파일에서 파라미터 값을 수정해야 한다.

DB 마이그레이션

다음 표는 http://ftp.cubrid.org/CUBRID_Engine/11.4/ 에서 제공되는 예약어 검출 스크립트인 check_reserved.sql와 cubrid unloaddb/loaddb 유틸리티를 사용하여 마이그레이션을 수행하는 방법을 보여준다. (unloaddb 및 loaddb 참고)

단계	Linux 환경	Windows 환경
Step C1: CUBRID Service 종료	% cubrid service stop	CUBRID Service Tray 를 종료한다
Step C2: 예약어 검출 스크립트 실행	예약어 검출 스크립트가 위치하는 디렉토리에서 아래 명령을 실행한다. 검출 결과를 확인하여 마이그레이션 진행 또는 식별자 수정 작업을 진행한다. (허가된 식별자에 대해)	

단계	Linux 환경	Windows 환경
	% csql -S -u dba -i check_reserved.sql testdb	
Step C3: 이전 버전 DB 언로드	이전 버전의 databases.txt 및 설정 파일을 별도의 디렉토리에 보관한다. (C3a) cubrid unload 유틸리티를 실행하고 이 때 생성된 파일을 별도 디렉토리에 보관한다.(C3b) % cubrid unloaddb -S -u dba testdb 이전 DB 를 제거 한다. (C3c) % cubrid deletedb testdb	
		이전 버전의 큐브리드를 언인스톨한다.
Step C4: 새 버전을 인스톨한다.	다음을 참고한다. 설치와 실행	
Step C5: DB 생성 및 데이터 로딩	원하는 DB 생성 디렉토리로 이동 후 DB 생성한다. 이 때 로컬 세팅에 주의한다.(*). (C5a) % cd \$CUBRID/databases/testdb % cubrid createdb testdb en_US (C3b)에서 생성한 파일을 가지고cubrid loaddb 유틸리티를 실행한다. (C5b) % cubrid loaddb -u dba -s testdb_schema -d testdb_objects -i testdb_indexes testdb	
Step C6: 새 버전의 DB 백업	% cubrid backupdb -S testdb	
Step C7: CUBRID 환경 설정 및 CUBRID Service 구동	환경 설정 파일을 수정한다. 이 때 (C3a)에서 보관한 이전 버전의 환경 설정 파일을 새 버전에 맞게 수정한다.	CUBRID Service Tray > [Service Start] 를 선택하여 서비스를 시작한다.

단계	Linux 환경	Windows 환경
	(시스템 파라미터 설정은 다음을 참조 파라미터 설정 과 시스템 설정)	명령 프롬프트 창에서 DB 서버를 구동한다.
	% cubrid service start	% cubrid server start testdb
	% cubrid server start testdb	

파라미터 설정

cubrid.conf

- **log_buffer_size** 의 최소값이 48KB(3*1page, 16KB=1page)에서 2MB(128*1page, 16KB=1page)로 변경 되었으므로, 이 값을 설정한 경우 변경된 최소값보다 크게 설정해야 한다.

CUBRID 9.1에서 CUBRID 11.4로 업그레이드하기

CUBRID 9.1 버전을 사용 중인 사용자는 다른 디렉터리에 11.4를 설치하고 데이터베이스를 11.4로 마이그레이션한 후 이전 환경 설정 파일에서 파라미터 값을 수정해야 한다.

DB 마이그레이션

DB 마이그레이션 를 참고한다.

파라미터 설정

cubrid.conf

- **log_buffer_size** 의 최소값이 48KB(3*1page, 16KB=1page)에서 2MB(128*1page, 16KB=1page)로 변경 되었으므로, 이 값을 설정한 경우 변경된 최소값보다 크게 설정해야 한다.
- **sort_buffer_size** 의 최대값을 2G로 제한되므로 **sort_buffer_size** 의 값은 2G 보다 크게 설정하지 않아야 한다.
- 다음 표의 기존 파라미터는 더 이상 지원하지 않을 예정이며 신규 파라미터 사용을 권장한다. 괄호 안의 값이 생략된 경우 기본 적용되는 단위이며, 신규 파라미터는 단위 지정이 가능하다. 자세한 내용은 **시스템 설정** 에서 각 파라미터 설명을 참고한다.

기존 파라미터(단위)	신규 파라미터(단위)
lock_timeout_in_secs(sec)	lock_timeout(msec)
checkpoint_every_npages(page_count)	checkpoint_every_size(byte)
checkpoint_interval_in_mins(min)	checkpoint_interval(msec)
max_flush_pages_per_second(page_count)	max_flush_size_per_second(byte)
sync_on_nflush(page_count)	sync_on_flush_size(byte)
sql_trace_slow_msecs(msec)	sql_trace_slow(msecs)

cubrid_broker.conf

- **KEEP_CONNECTION** 에서 OFF 설정이 제거되었으므로 기존 버전에서 OFF로 설정한 경우 ON 또는 AUTO로 변경해야 한다.
- **SELECT_AUTO_COMMIT** 이 제거되었으므로 기존 버전에서 이 파라미터의 설정을 제거해야 한다.
- **APPL_SERVER_MAX_SIZE_HARD_LIMIT** 의 최대값을 2,097,151으로 제한되므로 이 값보다 크게 설정하지 않아야 한다.

환경 변수

- **CUBRID_CHARSET** 이 제거되고, DB 생성 시 데이터베이스의 언어 및 문자셋을, **CUBRID_MSG_LANG** 으로 유틸리티 메시지 및 오류 메시지의 언어 및 문자셋을 설정하게 되었다.

경고

데이터베이스를 생성할 때 언어 및 문자셋을 반드시 지정해야 하며, 문자셋에 따라 문자열 타입의 크기, 문자열 비교 연산 등에 영향을 끼친다. 데이터베이스 생성 시 지정된 문자셋은 변경할 수 없으므로 지정에 주의해야 한다.

문자셋, 로캘 및 콜레이션 설정과 관련된 자세한 내용은 [다국어 개요](#)를 참고한다.

CUBRID 2008 R4.1/R4.3/R4.4에서 CUBRID 11.4로 업그레이드하기

CUBRID 2008 R4.1, R4.3 또는 R4.4 버전을 사용 중인 사용자는 다른 디렉터리에 11.4를 설치하고 데이터베이스를 11.4로 마이그레이션한 후 기존 환경 설정 파일에서 파라미터 값을 수정해야 한다.

DB 마이그레이션

DB 마이그레이션 를 참고하여 마이그레이션을 수행한다.

(*): CUBRID 2008 R4.x 이하 버전 사용자는 로캘(언어와 문자셋) 결정에 특히 주의해야 한다. 예를 들어 언어는 ko_KR(한국어)이고 문자셋은 utf8을 사용하던 2008 R4.3 사용자가 10.0 으로 마이그레이션을 진행하는 경우, “cubrid createdb testdb ko_KR.utf8”과 같이 로캘을 지정해야 한다. 지정하려는 로캘이 시스템 내장 로캘이 아닌 경우, 먼저 make_locale(.sh) 명령을 실행해야 한다. **로캘 설정** 을 참고한다.

- 멀티바이트 문자에 대한 저장 공간 변화에 주의해야 한다. 예를 들어 2008 R4.3에서 **CHAR(6)** 은 6바이트 CHAR 타입을 의미하지만 9.3에서 **CHAR(6)** 은 6글자 **CHAR** 타입을 의미한다. utf8 문자셋에서 한글은 한 글자 당 3바이트를 차지하므로, **CHAR(6)** 은 18바이트를 차지한다. 따라서 이전 버전보다 더 많은 디스크 공간을 필요로 한다.
- CUBRID 2008 R4.x 이하 버전에서 utf8 문자셋을 사용했다면, “cubrid createdb” 수행 시 반드시 utf8 문자셋으로 지정해야 한다. 그렇지 않을 경우 검색 또는 문자열 함수가 제대로 동작하지 않는다.

파라미터 설정

cubrid.conf

- **log_buffer_size** 최소값이 48KB(3*1page, 16KB=1page)에서 2MB(128*1page, 16KB=1page)로 변경되었으므로, 이 값을 설정한 경우 변경된 최소값보다 크게 설정해야 한다.
- **sort_buffer_size** 의 최대 크기를 2G로 제한했으므로 이 값보다 크게 설정하지 않아야 한다.
- **single_byte_compare** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- **intl_mbs_support** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- **lock_timeout_message_type** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- 다음 표의 기존 파라미터들은 더 이상 지원하지 않을 예정이며, 앞으로 신규 파라미터의 사용을 권장한다. 괄호 안의 값이 단위 생략된 경우 기본 적용되는 단위이며, 신규 파라미터들은 단위 지정이 가능하다. 자세한 내용은 **시스템 설정** 의 각 파라미터 설명을 참고한다.

기존 파라미터(단위)	신규 파라미터(단위)
lock_timeout_in_secs(sec)	lock_timeout(msec)
checkpoint_every_npages(page_count)	checkpoint_every_size(byte)
checkpoint_interval_in_mins(min)	checkpoint_interval(msec)
max_flush_pages_per_second(page_count)	max_flush_size_per_second(byte)
sync_on_nflush(page_count)	sync_on_flush_size(byte)
sql_trace_slow_msecs(msec)	sql_trace_slow(msecs)

cubrid_broker.conf

- **KEEP_CONNECTION** 에서 OFF 설정이 제거되었으므로 기존 버전에서 OFF로 설정한 경우 ON 또는 AUTO로 변경해야 한다.
- **SELECT_AUTO_COMMIT** 이 제거되었으므로 기존 버전에서 이 파라미터의 설정을 제거해야 한다.
- **APPL_SERVER_MAX_SIZE_HARD_LIMIT** 의 최대값을 2,097,151으로 제한했으므로 이 값보다 크게 설정하지 않아야 한다.

cubrid_ha.conf

- **ha_apply_max_mem_size** 파라미터의 값을 500보다 크게 설정한 사용자는 이 값을 500 이하로 설정해야 한다.

환경 변수

- **CUBRID_LANG** 이 제거되고, DB 생성 시 데이터베이스의 언어 및 문자셋을, **CUBRID_MSG_LANG** 으로 유틸리티 메시지 및 오류 메시지의 언어 및 문자셋을 설정하게 되었다.

⚠ 경고

데이터베이스를 생성할 때 언어 및 문자셋을 반드시 지정해야 하며, 문자셋에 따라 문자열 타입의 크기, 문자열 비교 연산 등에 영향을 끼친다. 데이터베이스 생성 시 지정된 문자셋은 변경할 수 없으므로 지정에 주의해야 한다.

문자셋, 로컬 및 콜레이션 설정과 관련된 자세한 내용은 **다국어 개요**를 참고한다.

CUBRID 2008 R4.0 이하 버전에서 CUBRID 11.4로 업그레이드하기

CUBRID 2008 R4.0 이하 버전 사용자는 CUBRID 11.4 버전을 별도의 디렉터리에 설치한 후 데이터베이스를 11.4로 마이그레이션한 후 기존의 환경 설정 파일에서 파라미터들의 값을 변경해야 한다.

DB 마이그레이션

DB 마이그레이션과 동일한 절차대로 수행한다. 단, CUBRID 2008 3.1 이하 버전의 GLO 클래스 사용자가 마이그레이션하는 경우, CUBRID 2008 R3.1부터는 GLO 클래스를 지원하지 않으므로 **BLOB** 또는 **CLOB** 타입을 사용하도록 응용과 스키마를 변경해야 한다. 변경 작업이 용이하지 않다면 마이그레이션을 보류할 것을 권장한다.

파라미터 설정

cubrid.conf

- **log_buffer_size** 최소값이 48KB(3*1page, 16KB=1page)에서 2MB(128*1page, 16KB=1page)로 변경되었으므로, 이 값을 설정한 경우 변경된 최소값보다 크게 설정해야 한다.
- **sort_buffer_size** 의 최대 크기를 2G로 제한했으므로 이 값보다 크게 설정하지 않아야 한다.
- **single_byte_compare** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- **intl_mbs_support** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- **lock_timeout_message_type** 파라미터는 더 이상 사용하지 않으므로 삭제해야 한다.
- **thread_stacksize** 의 기본값이 100K에서 1M으로 변경되었으므로, 이 값을 설정하지 않은 사용자는 CUBRID 관련 프로세스들의 메모리 사용량을 살펴볼 것을 권장한다.
- **data_buffer_size** 의 최소값이 64K에서 16M으로 변경되었으므로, 이 값을 16M 미만으로 설정한 사용자는 16M 이상으로 설정해야 한다.
- 다음 파라미터 중 기존 파라미터들은 더 이상 사용하지 않을 예정(deprecated)이며, 앞으로 신규 파라미터의 사용을 권장한다. 괄호 안의 값은 단위 생략 시 기본 적용되는 단위이며, 신규 파라미터들은 단위 지정이 가능하다. 자세한 내용은 **시스템 설정** 의 각 파라미터 설명을 참고한다.

기존 파라미터(단위)	신규 파라미터(단위)
lock_timeout_in_secs(sec)	lock_timeout(msec)
checkpoint_every_npages(page_count)	checkpoint_every_size(byte)
checkpoint_interval_in_mins(min)	checkpoint_interval(msec)
max_flush_pages_per_second(page_count)	max_flush_pages_per_second(page_count)
sync_on_nflush(page_count)	sync_on_flush_size(byte)

cubrid_broker.conf

- **KEEP_CONNECTION** 에서 OFF 설정값이 제거되었으므로 기존 버전에서 **OFF** 로 설정한 경우 **ON** 또는 **AUTO** 로 변경해야 한다.
- **SELECT_AUTO_COMMIT** 이 제거되었으므로 기존 버전에서 이 파라미터를 설정한 경우 제거해야 한다.
- **APPL_SERVER_MAX_SIZE_HARD_LIMIT** 의 최대값을 2,097,151으로 제한했으므로 이 값보다 크게 설정하지 않아야 한다.
- **APPL_SERVER_MAX_SIZE_HARD_LIMIT** 의 최소값이 1024M이다. **APPL_SERVER_MAX_SIZE** 의 값을 설정하는 사용자는 **APPL_SERVER_MAX_SIZE_HARD_LIMIT** 의 값보다 작게 설정할 것을 권장한다.
- **CCI_DEFAULT_AUTOCOMMIT** 의 기본값이 ON으로 변경되었으므로, 이를 설정하지 않은 응용 프로그램 사용자가 기존과 같은 자동 커밋 모드를 유지하고 싶다면 **OFF** 로 설정해야 한다.

cubrid_ha.conf

- **ha_apply_max_mem_size** 파라미터의 값을 500 이상으로 설정한 사용자는 이 값을 500 이하로 설정해야 한다.

환경 변수

- **CUBRID_LANG** 이 제거되고, DB 생성 시 데이터베이스의 언어 및 문자셋을, **CUBRID_MSG_LANG** 으로 유틸리티 메시지 및 오류 메시지의 언어 및 문자셋을 설정하게 되었다.

⚠ 경고

데이터베이스를 생성할 때 언어 및 문자셋을 반드시 지정해야 하며, 문자셋에 따라 문자열 타입의 크기, 문자열 비교 연산 등에 영향을 끼친다. 데이터베이스 생성 시 지정된 문자셋은 변경할 수 없으므로 지정에 주의해야 한다.

문자셋, 로캘 및 콜레이션 설정과 관련된 자세한 내용은 **다국어 개요**를 참고한다.

HA 환경에서 DB 마이그레이션

CUBRID 2008 R2.2 이상 버전에서 CUBRID 11.4로 HA 마이그레이션

아래는 브로커, 마스터 DB, 슬레이브 DB를 각각 별도 서버에 구축한 환경에서 현재 서비스를 중지하고 업그레이드를 수행하기 위한 절차이다.

단계	설명
Steps C1-C6: DB 마이그레이션 실행	마스터 노드에서 CUBRID 업그레이드 및 DB 마이그레이션을 수행하고 새 버전의 DB를 백업한다.
Step C7: 슬레이브 노드에 CUBRID 새 버전 설치	슬레이브 노드에서 이전 버전의 DB는 삭제하고, 새 버전을 설치한다. 상세 정보는 다음을 참고한다. 설치와 실행.
Step C8: 마스터 노드 백업본을 슬레이브 노드에서 복구	C6 단계에서 생성된 마스터 노드의 새 버전 DB 백업본(예: testdb_bk*)을 슬레이브 노드에서 복구한다. <pre> % scp user1@master:\$CUBRID/ databases/databases.txt \$CUBRID/databases/. % cd ~/DB/testdb % scp user1@master:~/DB/ testdb/testdb_bk0v000 . </pre>

단계	설명
	<pre>% scp user1@master:~/DB/ testdb/testdb_bkvinf . % cubrid restoredb testdb</pre>
Step C9: HA 환경 재구성 후 HA모드 구동	마스터 및 슬레이브 노드에서 CUBRID 환경 설정 파일(cubrid.conf) 및 HA 환경 설정 파일(cubrid_ha.conf)을 설정한다. 다음을 참고한다. 데이터베이스 생성 및 서버 설정.
Step C10: 브로커 서버에 새 버전 설치 및 브로커 구동	설치 상세 정보는 다음을 참고한다. 설치와 실행. 브로커 서버에 있는 브로커를 시작한다. 다음을 참고한다. 브로커 설정, 시작 및 확인. <pre>% cubrid broker start</pre>

CUBRID 2008 R2.0 또는 R2.1에서 CUBRID 11.4로 HA 마이그레이션

CUBRID 2008 R2.0 또는 R2.1의 HA 기능을 사용하는 경우, 서버 버전 업그레이드, DB 마이그레이션을 수행하고 HA 환경을 새롭게 구축한 후 해당 버전에서 사용되었던 Linux Heartbeat 자동 시작 설정을 변경해야 한다. (Linux Heartbeat 패키지가 불필요한 경우 삭제한다.)

위의 C1~C10 단계를 수행한 후, 아래의 C11 단계를 수행한다.

단계	설명
C11 단계: 기존 Linux heartbeat 자동 시작 설정 변경	이하의 작업은 마스터 및 슬레이브 서버에서 root 계정으로 수행한다. <pre>[root@master ~]# chkconfig --del heartbeat // 슬레이브 서버에서 동일 작업 수행</pre>

CUBRID 제거

Linux에서 CUBRID 제거

SH 패키지 또는 압축 패키지의 CUBRID 제거

SH 패키지(설치 패키지의 파일 확장자 명이 .sh) 또는 압축 패키지(설치 패키지의 파일 확장자 명이 .tar.gz)를 제거하려면 다음의 절차대로 수행한다.

- 서비스 종료 후 데이터베이스 제거

CUBRID 서비스를 종료한 후 기존에 생성된 데이터베이스를 모두 제거한다.

예를 들어 기존에 생성된 데이터베이스가 testdb1, testdb2라면 다음과 같이 명령을 수행한다.

```
$ cubrid service stop
$ cubrid deletedb testdb1
$ cubrid deletedb testdb2
```

- CUBRID 엔진 제거

\$CUBRID 디렉터리 이하를 모두 제거한다.

```
$ echo $CUBRID
/home1/user1/CUBRID

$ rm -rf /home1/user1/CUBRID
```

- 자동 구동 스크립트 제거

/etc/init.d 디렉터리 이하에 cubrid 스크립트를 저장했다면 이 파일을 제거한다.

```
cd /etc/init.d
rm cubrid
```

RPM 패키지의 CUBRID 제거

RPM 패키지로 설치했다면 rpm 명령을 통해 제거가 가능하다.

- 서비스 종료 후 데이터베이스 제거

CUBRID 서비스를 종료한 후 기존에 생성된 데이터베이스를 모두 제거한다.

예를 들어 기존에 생성된 데이터베이스가 testdb1, testdb2라면 다음과 같이 명령을 수행한다.


```
$ cubrid service stop
$ cubrid deletedb testdb1
$ cubrid deletedb testdb2
```

• CUBRID 엔진 제거

```
$ rpm -q cubrid
cubrid-10.1.0.7663-1ca0ab8-Linux.x86_64

$ rpm -e cubrid
warning: /opt/cubrid/conf/cubrid.conf saved as /opt/cubrid/conf/
cubrid.conf.rpmsave
```

• 자동 구동 스크립트 제거

/etc/init.d 디렉터리 이하에 cubrid 스크립트를 저장했다면 이 파일을 제거한다.

```
cd /etc/init.d
rm cubrid
```

Windows에서 CUBRID 제거

“제어판 > 프로그램 제거”에서 CUBRID를 선택하여 제거한다.

시작하기

CUBRID를 처음 사용하는데 참고할 수 있는 간략한 사용법을 설명한다. CUBRID 서비스 시작하기, CSQL 인터프리터 및 GUI 도구를 이용한 질의 실행 방법을 찾아볼 수 있다.

CUBRID의 다양한 도구 및 드라이버는 <https://www.cubrid.org/downloads>에서 다운로드할 수 있다.

CUBRID가 제공하는 다음 도구를 사용하면 GUI를 통해 편리하게 CUBRID의 기능을 이용할 수 있다.

- CUBRID 매니저
- CUBRID 마이그레이션 툴킷

CUBRID는 JDBC, CCI, PHP, PDO, ODBC, OLE DB, ADO.NET, Perl, Python, Ruby 등 다양한 드라이버를 제공한다. 드라이버에 대한 자세한 내용은 API 레퍼런스를 참고한다.

CUBRID 서비스 시작

환경 변수 및 언어 설정을 완료한 후, CUBRID 서비스를 시작한다. 이에 대한 자세한 설명은 [CUBRID 서비스](#)를 참고한다.

셸 명령어

Linux 환경 또는 Windows 환경에서 아래와 같은 셸 명령어로 CUBRID 서비스를 시작하고, 설치 패키지에 포함된 demodb를 구동할 수 있다.

```
% cubrid service start

@ cubrid master start
++ cubrid master start: success

@ cubrid broker start
++ cubrid broker start: success

@ cubrid manager server start
++ cubrid manager server start: success

% cubrid server start demodb

@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success

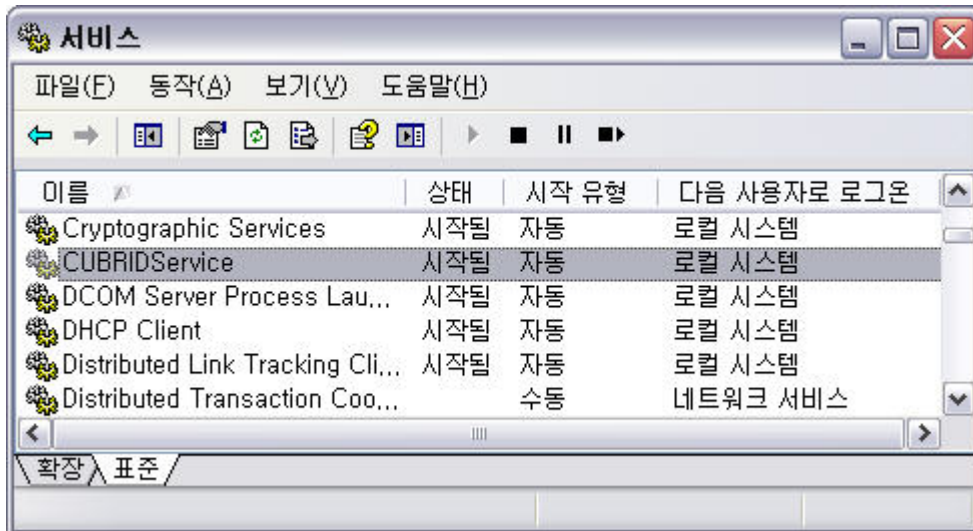
@ cubrid server status

Server demodb (rel 11.4, pid 31322)
```

CUBRIDService 또는 CUBRID Service Tray

Windows 환경에서는 다음과 같은 방법으로 CUBRID 서비스를 시작하거나 중지할 수 있다.

- [제어판] > [성능 및 유지 관리] > [관리도구] > [서비스]에 등록된 CUBRIDService를 선택하여 시작하거나 중지한다.



- 시스템 트레이에서 CUBRID Service Tray를 마우스 오른쪽 버튼으로 클릭한 후, CUBRID를 시작하려면 [Service Start]를 선택하고 중지하려면 [Service Stop]을 선택한다.

시스템 트레이에서 [Service Start]/[Service Stop] 메뉴를 선택하면, 명령어 프롬프트 창에서 **cubrid service start** / **cubrid service stop** 을 실행했을 때와 같은 동작을 수행하며, **cubrid.conf**의 **service** 파라미터에 설정한 프로세스들을 구동/중지한다.

- CUBRID가 실행 중일 때 CUBRID 서비스 트레이에서 [Exit]를 선택하면, 해당 서버에서 실행 중인 모든 서비스와 프로세스가 중지되므로 주의한다.

참고

CUBRID 서비스 트레이를 통해 CUBRID 관련 프로세스를 시작/종료하는 작업은 관리자 권한 (SYSTEM)으로 수행되고, 셸 명령어로 시작/종료하는 작업은 로그인한 사용자 권한으로 수행된다. Windows Vista 이상 버전의 환경에서 셸 명령어로 CUBRID 프로세스가 제어되지 않는 경우, 명령 프롬프트 창을 관리자 권한으로 실행([시작] > [모든 프로그램] > [보조 프로그램] > [명령 프롬프트])를 마우스 오른쪽 버튼으로 클릭하여 [관리자 권한으로 실행] 선택)하거나 CUBRID 서비스 트레이를 이용해서 해당 작업을 수행할 수 있다. CUBRID 서버 프로세스가 모두 중단되면, CUBRID Service Tray 아이콘이 회색으로 변한다.

데이터베이스 생성

데이터베이스 볼륨 및 로그 볼륨이 위치할 디렉터리에서 **cubrid createdb** 유틸리티를 실행하여 데이터베이스를 생성할 수 있다. **--db-volume-size** 또는 **--log-volume-size** 와 같은 추가 옵션을 지정하지 않으면 기본적으로 1.5GB 크기의 볼륨 파일이 생성된다(데이터 볼륨 512MB, 활성 로그 512MB, 백그라운드 보관 로그 512MB로 설정됨).

```
% cd testdb
% cubrid createdb testdb en_US
% ls -l
```

```
-rw----- 1 cubrid dbms 536870912 Jan 11 15:04 testdb
-rw----- 1 cubrid dbms 536870912 Jan 11 15:04 testdb_lgar_t
-rw----- 1 cubrid dbms 536870912 Jan 11 15:04 testdb_lgat
-rw----- 1 cubrid dbms      176 Jan 11 15:04 testdb_lginf
-rw----- 1 cubrid dbms      183 Jan 11 15:04 testdb_vinf
```

위에서 *testdb* 는 데이터 볼륨 파일을, *testdb_lgar_t*는 백그라운드 보관 로그 파일을, *testdb_lgat*는 활성 로그 파일을, *testdb_lginf*는 로그 정보 파일을, 그리고 *testdb_vinf*는 볼륨 정보 파일을 나타낸다.

볼륨에 대한 자세한 내용은 [데이터베이스 볼륨 구조](#) 를, 볼륨 생성에 대한 자세한 내용은 [createdb](#) 를 참고한다. **cubrid addvoldb** 유틸리티를 사용해 용도에 따라 볼륨을 분류해 추가하도록 권장한다. 자세한 내용은 [addvoldb](#) 을 참고한다.

데이터베이스 시작

데이터베이스 프로세스를 시작하려면 **cubrid** 명령어를 이용한다.

```
% cubrid server start testdb
```

앞에서 설명한 CUBRID 서비스 시작(**cubrid service start**) 시 *testdb* 가 같이 시작되게 하려면, **cubrid.conf** 파일의 **server** 파라미터에 *testdb* 를 설정한다.

```
% vi cubrid.conf

[service]

service=server,broker,manager
server=testdb

...
```

질의 도구

CSQL 인터프리터

CSQL 인터프리터는 CUBRID에서 명령어 방식으로 SQL 질의를 수행하고 수행 결과를 조회할 수 있는 프로그램이다. 입력된 SQL 문장과 그 결과는 나중에 사용하기 위해서 파일에 저장될 수도 있다. 자세한 내용은 [CSQL 인터프리터 소개](#) 및 [CSQL 실행 모드](#)를 참고한다.

본 장에서는 Linux 환경에서 CSQL 인터프리터를 사용하는 경우를 설명한다.

CSQL 인터프리터는 셸에서 다음과 같이 시작할 수 있다. 처음 설치한 상태에서는 **PUBLIC** 과 **DBA** 사용자가 제공되며, 이들의 비밀번호는 설정되어 있지 않다. CSQL 인터프리터 실행 시 사용자를 지정하지 않으면 **PUBLIC** 으로 로그인된다.

```
% csql demodb
```

CUBRID SQL Interpreter

Type `;help' for help messages.

```
csql> ;help
```

=== <Help: Session Command Summary> ===

All session commands should be prefixed by `;' and only blanks/
tabs
can precede the prefix. Capitalized characters represent the
minimum
abbreviation that you need to enter to execute the specified
command.

;REAd	[<file-name>]	- read a file into command buffer.
;Write	[<file-name>]	- (over)write command buffer into a file.
;Append	[<file-name>]	- append command buffer into a file.
;PRINT		- print command buffer.
;SHELL		- invoke shell.
;CD		- change current working directory.
;EXit		- exit program.
;Clear		- clear command buffer.
;EDIT	[format/fmt]	- invoke system editor [after
formatter]		with command buffer.
;LIST		- display the content of command
buffer.		
;RUn		- execute sql in command buffer.
;Xrun		- execute sql in command buffer, and clear the command buffer.
;COMmit		- commit the current transaction.
;ROllback		- roll back the current transaction.

```

;AUTocommit [ON|OFF]      - enable/disable auto commit mode.
;REStart                  - reconnect to the current database
in a CSQL session.
;CHECKpoint               - execute the checkpoint(CSQL with --
sysadm only).
;Killtran                 - check transaction status
information or end a specific transaction.(CSQL with --sysadm only).

;SHELL_Cmd [shell-cmd]    - set default shell, editor, print,
pager and formatter
;EDITOR_Cmd [editor-cmd]  command to new one, or display the
current
;PRINT_Cmd [print-cmd]    one, respectively.
;PAGer_cmd [pager-cmd]
;FOrmatter_cmd [formatter-cmd]

;DATE                     - display the local time, date.
;DATAbase                 - display the name of database being
accessed.
;SCHEMA class-name        - display schema information of a
class.
;TRIGGER ['*'|trigger-name] - display trigger definition.
;Get system_parameter     - get the value of a system parameter.
;SET system_parameter=value - set the value of a system parameter.
;STring-width [width]     - set width that each column which is
a string type is displayed.
;COlumn-width [name]=[width] - set width that a specific column is
displayed.
;PLAN [simple/detail/off]  - show query execution plan.
;Info <command>           - display internal information.
;TIme [ON/OFF]            - enable/disable to display the query
execution time.

;LINE-output [ON/OFF]     - enable/disable to display each
value in a line
;HISTORYList              - display list of the executed
queries.
;HISTORYRead <history_num> - read entry on the history number
into command buffer.
;TRACE [ON/OFF] [text/json] - enable/disable sql auto trace.
;SIngleline [ON|OFF]      - enable/disable single-line mode.
;CONnect username [dbname | dbname@hostname]
- connect to the current or other
databases as a username.
;.Hist [ON/OFF]           - start/stop collecting statistics
information in CSQL(available DBA only).
;.Clear_hist              - clear the CSQL statistics
information in the buffer.
;.Dump_hist               - display the CSQL statistics
information in CSQL.
;.X_hist                  - display the CSQL statistics
information in CSQL with statistics data initialized.
;HELP                     - display this help message.

```

CSQL에서 SQL 실행

csql을 실행하고 난 후에는 csql> 프롬프트에서 원하는 SQL문을 입력해서 실행할 수 있다. 하나의 SQL 문은 세미콜론(;)으로 끝나도록 입력하며, 여러 개의 SQL문을 한 줄에 입력할 수도 있다. 세션 명령어는 ;help 명령으로 간단한 사용법을 찾아 볼 수 있으며 상세한 내용은 [세션 명령어](#)를 참고한다.

```
% csql demodb

csql> SELECT SUM(n) FROM (SELECT gold FROM participant WHERE
nation_code='KOR'
csql> UNION ALL SELECT silver FROM participant WHERE
nation_code='JPN') AS t(n);

=== <Result of SELECT Command in Line 2> ===

      sum(n)
=====
          82

1 rows selected. (0.106504 sec) Committed.

csql> ;exit
```

관리 도구

	특징 요약	최신 파일 다운로드
CUBRID Manager	SQL 실행 및 DB 운영을 위한 Java 클라이언트 도구이다 1. JAVA 기반 관리 도구(JRE 1.6 이상 요구) 2. 최초 다운로드 후 이후 버전 업데이트는 자동 실행 3. 멀티 호스트 관리에 적합 4. CUBRID Manager 서버를 통해 DB 접속	CUBRID Manager Download
CUBRID Migration Toolkit	소스 DB(MySQL, Oracle, CUBRID)에서 CUBRID로 데이	

특징 요약	최신 파일 다운로드
터 및 스키마를 이전하는 Java 기반 클라이언트 도구이다. 1. JAVA 기반 관리 도구(JRE 1.6 이상 요구) 2. 최초 다운로드 후 이후 버전 업데이트는 자동 실행 3. 다중 SQL문 실행 결과만 이전 가능, 작업 시나리오 재사용 가능하여 배치 작업에 유리 4. JDBC로 DB에 직접 접속	CUBRID Migration Toolkit Download

CUBRID 매니저로 SQL 실행하기

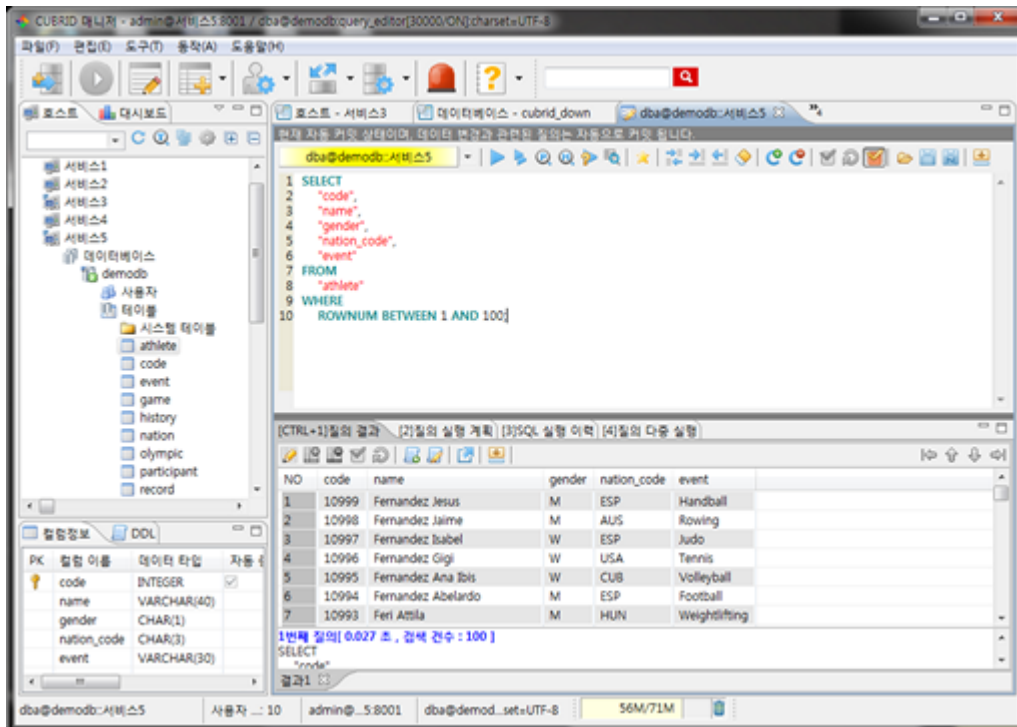
CUBRID 매니저는 별도로 다운로드 한 후 실행해야 하는 클라이언트 도구이며, JRE 혹은 JDK 1.6 이상 버전에서 실행되는 Java 애플리케이션이다.

1. CUBRID 매니저 최신 파일을 다운로드한 후 설치한다. CUBRID 매니저는 CUBRID 엔진 버전 2008 R2.2 이상부터 호환된다. 또한, 자동 업데이트 기능을 지원하므로 주기적으로 최신 버전을 유지하는 것이 좋다. (CUBRID FTP: http://ftp.cubrid.org/CUBRID_Tools/CUBRID_Manager)
2. 서버에서 CUBRID Service를 시작한다. CUBRID 매니저 서버가 구동되어야 CUBRID 매니저 클라이언트가 접속할 수 있다. CUBRID 매니저 서버의 실행 및 설정에 대한 자세한 내용은 **CUBRID 매니저 서버** 를 참고한다.

```
C:\CUBRID>cubrid service start
++ cubrid service is running.
```

3. CUBRID 매니저를 설치한 후 [파일 > 호스트 추가] 메뉴에서 호스트 정보를 등록한다. 호스트 등록 시에는 호스트 주소, 연결 포트(기본: 8001), CM 사용자 및 비밀번호를 입력해야 하며, 해당 서버의 엔진과 버전이 동일한 JDBC 드라이버를 설치해야 한다(자동 드라이버 검색/자동 업데이트 지원).
4. 왼쪽에 노드 트리에서 호스트를 선택하고 CM 사용자(=호스트 사용자) 인증을 수행한다. 기본 사용자 계정은 admin/admin이다.

5. 데이터베이스 노드에서 마우스 우클릭을 하여 새로운 데이터베이스를 생성하거나, 호스트 노드 하위에 있는 기존 데이터베이스를 선택하여 접속을 시도한다. 이때에는 DB 사용자 인증을 수행한다. 기본 사용자 이름은 dba이며 암호는 없다.
6. 접속한 DB에서 SQL을 실행하고, 결과를 확인한다. 왼쪽에는 호스트, 데이터베이스, 테이블 목록이 출력되고, 오른쪽에는 질의 편집기와 질의 결과 창이 있다. [SQL 실행 이력] 탭에서는 DB별로 실행 성공한 SQL 리스트를 재사용할 수 있으며, [질의 다중 실행] 탭에서 결과 비교를 위한 DB를 추가하여 여러 데이터베이스에서 결과값을 쉽게 비교할 수 있다.



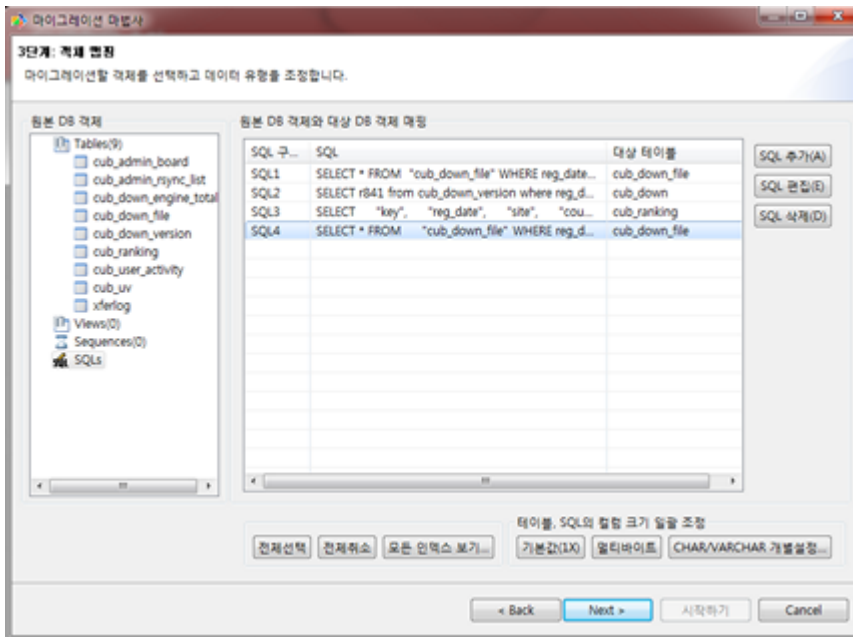
CUBRID 마이그레이션 툴킷으로 스키마/데이터 이전하기

CUBRID 마이그레이션 툴킷은 소스 데이터베이스(MySQL, Oracle, CUBRID)에서 타겟 데이터베이스(CUBRID)로 데이터 및 스키마를 이전하는 도구이다. 역시 JRE 혹은 JDK 1.6 이상 버전에서 실행되는 Java 애플리케이션이며, 별도로 다운받아야 한다. (CUBRID FTP: http://ftp.cubrid.org/CUBRID_Tools/CUBRID_Migration_Toolkit)

DB를 CUBRID로 전환하는 경우, 장비를 이전하는 경우, 운영 DB로부터 일부 스키마와 일부 데이터를 이전하고자 하는 경우, CUBRID 버전 업그레이드를 하는 경우, 배치 작업을 수행하는 경우 유용하다. 주요 기능은 다음과 같다.

- 전체/일부 스키마 및 데이터 마이그레이션 지원
- 여러 개의 SQL을 실행하여 원하는 결과 데이터만 마이그레이션 가능
- JDBC를 통한 온라인 마이그레이션 지원하여 운영 중단 없이 실행 가능
- CSV, SQL, CUBRID loaddb 포맷으로 출력 후 오프라인 마이그레이션 가능

- 마이그레이션 실행 스크립트를 추출하여 타겟 서버에서 직접 실행 가능
- 마이그레이션 실행 스크립트를 재사용할 수 있어 배치 작업 시간 단축



드라이버

CUBRID가 지원하는 드라이버는 다음과 같다.

- CUBRID JDBC driver (Downloads JDBC)
- CUBRID CCI driver (Downloads CCI)
- CUBRID PHP driver (Downloads PHP)
- CUBRID PDO driver (Downloads PDO)
- CUBRID ODBC driver (Downloads ODBC)
- CUBRID ADO.NET driver (Downloads ADO.NET)
- CUBRID Perl driver (Downloads Perl)
- CUBRID Python driver (Downloads Python)
- CUBRID Ruby driver (Downloads Ruby)
- CUBRID Node.js driver (Downloads Node.js)

위 드라이버 중 JDBC, CCI 드라이버는 CUBRID를 설치할 때 자동으로 다운로드되므로 따로 다운로드하지 않아도 된다.

CSQL 인터프리터

CUBRID에서 SQL 질의문을 수행하려면 그래픽 사용자 인터페이스(GUI) 기반 CUBRID Manager나 콘솔 기반 CSQL 인터프리터를 사용해야 한다.

CSQL은 CUBRID에서 명령어 방식으로 SQL 문을 사용할 수 있는 프로그램이다. 여기에서는 CSQL 인터프리터의 간단한 사용법과 관련 명령어를 설명한다.

CSQL 인터프리터 소개

SQL 사용을 위한 도구

CSQL 인터프리터는 CUBRID와 함께 설치되며, 대화형(interactive) 방식과 일괄 수행(batch) 방식으로 SQL 질의를 수행하고 수행 결과를 조회할 수 있는 프로그램이다. CSQL 인터프리터는 명령어 라인 입력 방식의 인터페이스를 제공하며, 입력된 SQL 문장과 그 결과는 나중에 사용하기 위해서 파일에 저장할 수도 있다.

CSQL 인터프리터는 CUBRID를 사용하는 가장 기본적이고 손쉬운 방법이다. CUBRID를 사용하는데 제공되는 다양한 API(JDBC, ODBC, PHP, CCI 등)를 활용하여 데이터베이스 응용 프로그램을 작성할 수 있다. 또한, CUBRID에서 제공하는 관리 및 질의 도구인 CUBRID 매니저를 사용할 수도 있다. 사용자는 CSQL 인터프리터가 제공하는 터미널 기반의 환경에서 SQL 질의를 생성하고, 수행 결과를 조회할 수 있다.

CSQL 인터프리터는 CUBRID 데이터베이스에 접속하여 SQL 문을 통해 다양한 작업을 수행한다. CSQL 인터프리터를 이용해 다음과 같은 작업을 수행할 수 있다.

- SQL 문을 이용하여 데이터베이스 조회, 갱신, 삭제 등의 작업
- 외부 셸 명령 실행
- 조회 결과의 저장 혹은 출력
- SQL 스크립트 파일의 작성 및 실행
- 테이블 스키마 조회
- 데이터베이스 서버 시스템 파라미터의 조회 및 변경
- 다양한 데이터베이스 정보(스키마, 트리거, 지연 트리거, workspace, 잠금, 통계) 조회

DBA를 위한 도구

DBA (Database Administrator)는 일상적인 많은 관리 업무를 수행하기 위해서 CUBRID가 설치된 시스템에 접속해서 CUBRID가 제공하는 다양한 관리 유틸리티를 이용해서 작업을 수행한다. 따라서, 터미널 기반의 인터페이스를 제공하는 CSQL 인터프리터는 **DBA**가 데이터베이스 관리 업무를 수행하

는데 유용하게 사용된다. 또한, CSQL 인터프리터는 **DBA**에게 필요한 다양한 데이터베이스 정보를 제공한다.

CSQL 인터프리터는 독립 모드(Standalone Mode)로 실행될 수도 있다. 독립 실행 모드는 CSQL 인터프리터가 서버 프로세스의 기능을 포함하여 직접 데이터베이스 파일에 접근하여 수행하는 방식이다. 즉 별도의 데이터베이스 서버 프로세스가 구동되어 있지 않은 상태에서 해당 데이터베이스를 대상으로 SQL 문을 실행할 수 있다. CSQL 인터프리터는 데이터베이스 서버나 브로커 등 어떠한 다른 프로그램의 도움 없이 **csql** 유틸리티 하나로 데이터베이스를 이용할 수 있는 강력한 수단이다.

CSQL 실행

CSQL 실행 모드

대화형 모드(Interactive Mode)

CSQL 인터프리터는 데이터베이스에서 스키마 또는 데이터를 다루기 위한 SQL 문을 입력하고 수행할 수 있다. **csql** 유틸리티를 실행하면 나타나는 프롬프트에 사용자는 구문을 입력한다. 구문을 입력한 후 실행하면 다음 라인에 결과가 표시되는데, 이를 대화형 모드라고 한다.

일괄 수행 모드(Batch Mode)

사용자는 원하는 SQL 문을 임의의 파일에 저장한 후 **csql** 유틸리티가 해당 파일을 읽도록 구문을 실행할 수 있다. 이를 일괄 수행(배치형) 모드라고 한다.

독립 모드(Standalone Mode)

독립 실행 모드는 CSQL 인터프리터가 서버 프로세스의 기능을 포함하여 직접 데이터베이스 파일에 접근하여 수행하는 방식이다. 즉 별도의 데이터베이스 서버 프로세스가 구동되어 있지 않은 상태에서 해당 데이터베이스를 대상으로 SQL 문을 실행할 수 있다. 독립 모드는 동시에 한 사용자만이 접근이 가능하므로 **DBA** (Database Administrator)가 관리 작업을 위해 수행하는데 적합한 모드이다.

클라이언트/서버 모드(Client/Server Mode)

클라이언트/서버 모드는 일반적으로 해당 CSQL 인터프리터가 클라이언트 프로세스로 동작하여 데이터베이스 서버 프로세스에 접속하는 방식으로 사용되는 모드이다.

시스템 관리자 모드

시스템 관리자 모드는 CSQL 인터프리터를 통해 특별한 관리 작업을 수행하기 위해 사용되는 모드이다. 서버 접속 개수가 시스템 파라미터 **max_clients**의 값을 초과하더라도 CSQL 인터프리터에서 시스템 관리자 모드로 접속하면 추가로 단 하나의 연결을 허용한다. 체크 포인트를 수행하거나 트랜잭션 모니터링을 종료 등의 작업을 수행할 수 있다.

```
csql -u dba --sysadm demodb
```

CSQL 사용 방법

로컬 호스트 접속

csql 유틸리티를 사용하여 CSQL 인터프리터를 실행한다. 이 때, 필요에 따라 옵션을 설정할 수 있으며, 옵션을 설정하려면 접속하려는 데이터베이스 이름을 인수로 지정한다. 다음은 로컬 서버에 위치한 데이터베이스에 접속하는 **csql** 유틸리티 구문이다.

```
csql [options] database_name
```

원격 호스트 접속

다음은 원격 호스트에 위치한 데이터베이스에 접속하는 **csql** 유틸리티 구문이다.

```
csql [options] database_name@remote_host_name
```

단, 원격 호스트에서 CSQL 인터프리터를 실행하려면 다음 조건을 만족해야 한다.

- 원격 호스트와 로컬 호스트에 설치된 CUBRID는 동일한 버전이어야 한다.
- 원격 호스트와 로컬 호스트의 마스터 프로세스가 사용하는 포트 번호가 동일해야 한다.
- **-C** 옵션을 사용하여 클라이언트/서버 모드로 원격 호스트에 접속해야 한다.

예제

다음은 192.168.1.3 위치의 원격 호스트에 존재하는 **demodb** 에 접속하여 **csql** 유틸리티를 호출하는 예제이다.

```
csql -C demodb@192.168.1.3
```

CSQL 시작 옵션

프롬프트 상에서 옵션 목록을 보려면, 다음과 같이 옵션을 적용할 데이터베이스를 지정하지 않고 **csql** 유틸리티를 실행한다.

```
$ csql
A database-name is missing.
interactive SQL utility, version 11.4
usage: csql [OPTION] database-name[@host]

valid options:
  -S, --SA-mode          standalone mode execution
  -C, --CS-mode          client-server mode execution
  -u, --user=ARG         alternate user name
```

```

-p, --password=ARG      password string, give "" for none
-e, --error-continue    don't exit on statement error
-i, --input-file=ARG    input-file-name
-o, --output-file=ARG    output-file-name
-s, --single-line        single line oriented execution
-c, --command=ARG       CSQL-commands
-l, --line-output        display each value in a line
-r, --read-only          read-only mode
-t, --plain-output       display results in a script-friendly
format (only works with -c and -i)
-q, --query-output       display results in a query-friendly
format (only work with -c and -i)
-d, --loaddb-output      display results in a loaddb-friendly
format (only work with -c and -i)
-N, --skip-column-names  do not display column names in
results (only works with -c and -i)
--string-width           display each column which is a string
type in this width
--no-auto-commit         disable auto commit mode execution
--no-pager               do not use pager
--no-single-line         turn off single line oriented
execution
--no-trigger-action      disable trigger action
--delimiter=ARG          delimiter between columns (only work
with -q)
--enclosure=ARG          enclosure for a result string (only
work with -q)

For additional information, see http://www.cubrid.org

```

옵션

-S, --SA-mode

-S 옵션을 이용하여 독립 모드로 데이터베이스에 접속하여 **csql**을 실행한다. 데이터베이스를 독립적으로 사용하고자 할 때 **-S** 옵션을 이용한다. **csql**이 독립 모드로 실행중이면 또 다른 **csql** 또는 유틸리티의 사용이 불가능하다. **-S** 옵션과 **-C** 옵션을 둘 다 생략하면 **-C** 옵션으로 동작한다.

```
csql -S demodb
```

-C, --CS-mode

-C 옵션을 이용하여 클라이언트/서버 모드로 데이터베이스에 접속하여 **csql** 유틸리티를 실행한다. 데이터베이스에 여러 클라이언트가 동시 접속하는 환경에서 **-C** 옵션을 이용한다. 만약 클라이언트/서버 모드로 원격 호스트의 데이터베이스에 접속한 경우라도 **csql** 유틸리티를 실행하는 도중에 발생한 에러 로그는 로컬 호스트의 **csql.err** 파일에 기록된다.

```
csql -C demodb
```

-i, --input-file=ARG

-i 옵션을 이용하여 배치 모드에서 사용할 입력 파일의 이름을 지정한다. **infile** 파일에는 하나 이상의 SQL 문이 저장되어 있으며, **-i** 옵션이 지정되지 않으면 CSQL 인터프리터는 대화형 모드로 실행된다.

```
csql -i infile demodb
```

-o, --output-file=ARG

-o 옵션을 이용하여 질의 수행 결과를 화면에 출력하지 않고 지정된 파일에 저장한다. 이는 CSQL 인터프리터에 의한 질의 수행 결과를 추후 조회하고자 할 때 유용하게 사용될 수 있다.

```
csql -o outfile demodb
```

-u, --user=ARG

-u 옵션을 이용하여 지정된 데이터베이스에 접속하려는 사용자 이름을 지정한다. 만약 **-u** 옵션이 지정되지 않으면 가장 낮은 사용자 권한을 가지는 **PUBLIC** 이 사용자로 지정된다. 또한 사용자 이름이 유효하지 않은 경우에는 오류가 출력되고 **csql** 유틸리티는 종료된다. 암호가 설정된 사용자 이름이 지정된 경우에는 암호를 입력받기 위한 프롬프트가 출력된다.

```
csql -u DBA demodb
```

-p, --password=ARG

-p 옵션을 이용하여 지정된 사용자의 암호를 입력한다. 특히, 배치 모드에서는 지정한 사용자에 대한 암호 입력을 요청하는 프롬프트가 출력되지 않으므로 **-p** 옵션을 이용하여 암호를 입력해야 한다. 잘못된 암호를 입력하면, 오류가 출력되고 **csql** 유틸리티는 종료된다.

```
csql -u DBA -p *** demodb
```

-s, --single-line

-i 옵션과 함께 사용하는 옵션으로, **-s** 옵션을 지정하면 파일에 입력된 여러 개의 SQL 문을 하나씩 나누어 수행한다. 이 옵션은 질의 수행에 메모리를 적게 할당하고 싶을 때 유용하게 이용할 수 있다. 각 SQL 문은 세미콜론(;)으로 구분한다. 옵션을 생략하면 여러 개의 SQL 문을 한꺼번에 읽어들이고 후 수행한다.

```
csql -s -i infile demodb
```

-c, --command=ARG

-c 옵션을 이용하여 셸 상에서 하나 이상의 SQL 문을 직접 수행한다. 이 때, 각 문장은 세미콜론(;)으로 구분한다.

```
csql -c 'select * from olympic;select * from stadium' demodb
```

-l, --line-output

-l 옵션을 이용하여 SQL 문을 실행한 결과 레코드의 SELECT 리스트 값들을 라인 단위로 나누어서 출력한다. **-l** 옵션을 지정하지 않으면 결과 레코드의 모든 SELECT 리스트 값들을 한 라인에 출력한다.

```
csql -l demodb
```

-e, --error-continue

SQL 문 여러 개를 연속으로 나열하여 실행할 때 **-e** 옵션을 이용하면 SQL 문 중간에 의미상 (semantic) 오류 또는 런타임 에러가 발생하여도 이를 무시하고 계속 SQL 문을 실행한다. 이때 SQL 문에 문법상(syntax) 오류가 있다면 **-e** 옵션이 지정되어 있어도 오류가 발생한 후의 질의를 실행하지 않는다.

```
$ csql -e demodb

csql> SELECT * FROM aaa;SELECT * FROM athlete WHERE code=10000;

In line 1, column 1,

ERROR: before ' ;SELECT * FROM athlete WHERE code=10000; '
Unknown class "aaa".

=== <Result of SELECT Command in Line 1> ===

      code  name                      gender
nation_code      event
=====
10000  'Aardewijn Pepijn'      'M'
'NED'      'Rowing'

1 rows selected. (0.006433 sec) Committed.
```

-r, --read-only

-r 옵션을 이용하여 읽기 전용으로 데이터베이스에 접속한다. 데이터베이스에 읽기 전용으로 접속하면 테이블을 만들거나 데이터를 입력할 수 없고 데이터를 조회만 할 수 있다.

```
csql -r demodb
```

-t, --plain-output

컬럼명과 값만 표시되며 **-c** 또는 **-i** 옵션과 함께 작동된다. 각 컬럼과 값이 탭과 줄 바꿈으로 구분되며, 결과에 포함된 탭과 백슬래시는 'n', 't' 및 '\'으로 각각 대체된다. 이 옵션은 **-l** 옵션과 함께 지정된 경우에는 무시된다.

```
$ csql demodb -c "select * from athlete where code between 12762 and 12765" -t
```

code	name	gender	nation_code	event
12762	O'Brien Dan	M	USA	Athletics
12763	O'Brien Leah	W	USA	Softball
12764	O'Brien Shaun William	M	AUS	Cycling
12765	O'Brien-Amico Leah	W	USA	Softball

-q, --query-output

결과를 insert 질의에서 사용할 수 있게 출력하는 옵션으로 컬럼명과 값만 표시되며 **-c** 또는 **-i** 옵션과 같이 사용해야 한다. 각 컬럼명과 값은 콤마 또는 **--delimiter** 옵션의 문자로 구분되며, 숫자형 타입을 제외한 모든 결과들은 작은 따옴표 또는 **--enclosure** 옵션의 문자로 둘러싸여 출력된다. 엔클러저(enclosure)가 작은 따옴표인 경우 결과안의 작은 따옴표는 두개 작은 따옴표로 대체된다. 이 옵션은 **-l** 옵션과 함께 지정된 경우에는 무시된다.

```
$ csql demodb -c "select * from athlete where code between 12762 and 12765" -q
```

```
code,name,gender,nation_code,event
12762,'O'Brien Dan','M','USA','Athletics'
12763,'O'Brien Leah','W','USA','Softball'
12764,'O'Brien Shaun William','M','AUS','Cycling'
12765,'O'Brien-Amico Leah','W','USA','Softball'
```

```
$ csql demodb -c "select * from athlete where code between 12762 and 12765" -q --delimiter="," --enclosure="\"
```

```
code,name,gender,nation_code,event
12762,"O'Brien Dan","M","USA","Athletics"
12763,"O'Brien Leah","W","USA","Softball"
12764,"O'Brien Shaun William","M","AUS","Cycling"
12765,"O'Brien-Amico Leah","W","USA","Softball"
```

-d, --loaddb-output

결과를 loaddb 유틸리티에서 사용할 수 있게 출력하는 옵션으로 컬럼명과 값만 표시되며 **-c** 또는 **-i** 옵션과 같이 사용해야 한다. 각 컬럼명과 값은 공백으로 구분되며, 숫자형 타입을 제외한 모든 결과들은 작은 따옴표로 둘러싸여 출력된다. 결과안의 작은 따옴표는 두개 작은 따옴표로 대체되며, 열거형(ENUM)타입의 결과는 값 대신 색인값을 출력한다. 이 옵션은 **-l** 옵션과 함께 지정된 경우에는 무시된다.

```
$ csql demodb -c "select * from athlete where code between 12762
and 12765" -d

%class [ ] ([code] [name] [gender] [nation_code] [event])
12762 'O' 'Brien Dan' 'M' 'USA' 'Athletics'
12763 'O' 'Brien Leah' 'W' 'USA' 'Softball'
12764 'O' 'Brien Shaun William' 'M' 'AUS' 'Cycling'
12765 'O' 'Brien-Amico Leah' 'W' 'USA' 'Softball'
```

-N, --skip-column-names

결과에서 컬럼명을 숨긴다. **-c** 또는 **-i** 옵션을 사용하는 경우에만 작동하며 일반적으로 **-t,**-q***-d**** 옵션과 함께 사용된다. 이 옵션은 **-l** 옵션과 함께 지정된 경우에는 무시된다.

```
$ csql demodb -c "select * from athlete where code between 12762
and 12765" -d -N

12762 'O' 'Brien Dan' 'M' 'USA' 'Athletics'
12763 'O' 'Brien Leah' 'W' 'USA' 'Softball'
12764 'O' 'Brien Shaun William' 'M' 'AUS' 'Cycling'
12765 'O' 'Brien-Amico Leah' 'W' 'USA' 'Softball'
```

--no-auto-commit

--no-auto-commit 옵션을 이용하여 자동 커밋 모드를 중지한다. **--no-auto-commit** 옵션을 지정하지 않으면 기본적으로 CSQL 인터프리터는 자동 커밋 모드로 작동되고, 입력된 SQL 문이 실행될 때마다 자동으로 커밋된다. 또한, CSQL 인터프리터를 시작한 후 **;Autocommit** 세션 명령을 수행해도 동일한 결과를 얻을 수 있다.

```
csql --no-auto-commit demodb
```

--no-pager

--no-pager 옵션을 이용하여 CSQL 인터프리터에서 수행한 질의 결과를 페이지 단위로 출력하지 않고, 일괄적으로 출력한다. **--no-pager** 옵션을 지정하지 않으면 페이지 단위로 질의 수행 결과를 출력한다.

```
csql --no-pager demodb
```

--no-single-line

--no-single-line 옵션을 이용하면 SQL 문 여러 개를 저장해 두었다가 **;xr** 혹은 **;ru** 세션 명령어로 한꺼번에 수행한다. 이 옵션을 지정하지 않으면 **;xr** 혹은 **;ru** 세션 명령어 없이 SQL 문이 바로 실행된다. 또한, **;Singleline** 세션 명령을 통해 동일한 결과를 얻을 수 있다.

```
csql --no-single-line demodb
```

--sysadm

이 옵션은 **-u dba**와 같이 사용해야 하며, 시스템 관리자 모드로 실행하고자 할 때 지정한다.

```
csql -u dba --sysadm demodb
```

--write-on-standby

이 옵션은 시스템 관리자 모드 옵션(**--sysadm**)과 함께 사용해야 한다. 이 옵션으로 CSQL을 실행한 dba는 standby 상태의 DB 즉, 슬레이브 DB 또는 레플리카 DB에 직접 접속하여 쓰기 작업을 수행할 수 있다. 단, 레플리카에 직접 쓰는 데이터는 복제되지 않는다.

```
csql --sysadm --write-on-standby -u dba testdb@localhost
```

i 참고

레플리카에 직접 데이터를 쓰는 경우 복제 불일치가 발생함에 주의해야 한다.

--no-trigger-action

이 옵션을 지정하면 해당 CSQL에서 수행되는 질의문의 트리거는 동작하지 않는다.

--delimiter=ARG

이 옵션은 **-q**와 같이 사용해야 하며, 인자에는 컬럼명과 값을 구분하는 단일 문자를 지정한다. 만약 여러개의 문자를 지정하는 경우에는 오류를 발생하지 않고 첫번째 문자를 사용한다. (\t, \n 와 같은 특수 문자 지정 가능하며 단일 문자로 취급)

--enclosure=ARG

이 옵션은 **-q**와 같이 사용해야 하며, 인자에는 숫자형 타입을 제외한 모든 결과값을 둘러싸는 단일 문자를 지정한다. 만약 여러개의 문자를 지정하는 경우에는 오류를 발생하지 않고 첫번째 문자를 사용한다.

세션 명령어

CSQL 인터프리터에는 SQL 문 이외에 CSQL 인터프리터를 제어하는 특별한 명령어가 있으며 이를 세션 명령어라고 한다. 모든 세션 명령어는 반드시 세미콜론(;)으로 시작해야 한다.

;help 를 입력하여 CSQL 인터프리터에서 지원되는 세션 명령어를 확인할 수 있다. 세션 명령어를 전부 입력하지 않고 고유한 세션 명령어를 식별할 수 있을 만큼의 최소 문자(다른 명령어와 구별하기 위한 최소 문자)만 입력해도 CSQL 인터프리터는 세션 명령어를 인식한다. 세션 명령어는 대소문자를 구분하지 않는다.

CSQL은 SQL의 문자열 리터럴, 주석, 식별자를 (참고: [작성 규칙](#)) 인식하면서 동작한다. 특히,

- 작은 따옴표로 둘러싸인 문자열 리터럴
- -, //, /* */ 로 표시되는 주석
- 큰따옴표, 대괄호, 백틱 부호로 둘러싸인 식별자

안에 위치하는 문자열은 이들의 밖에서 가질 수 있는 고유의 의미를 잃고 오로지 문자열로만 인식되는데, 이것은 세션 명령어에 대해서도 마찬가지이다. 예를 들어, 아래 문자열 리터럴, 주석, 식별자 안의 'exit'은 세션 명령어로서 동작하지 않고 문자열로만 인식된다.

```
csql> select * from table_a where col = ';exit';
...
csql> select * from table_b /* ;exit on no results */;
...
csql> create table table_c([;exit] int);
...
```

예외적으로 대화형 모드 CSQL에서는 명령행의 첫문자가 세미콜론인 경우 문자열 리터럴, 주석, 식별자 처리중이여도 해당 문자열은 세션 명령어로 인식한다.

```
csql> select * from table_a where col = '
csql> ;exit      <-- CSQL 종료
...
csql> select * from table_b /*
csql> ;exit      <-- CSQL 종료
...
csql> create table table_c([
csql> ;exit      <-- CSQL 종료
...
```

이 예외는 CSQL을 종료한다거나 (;exit), 편집하던 내용을 에디터로 이어서 편집한다거나 (;edit), 질의 버퍼를 비운다거나 (;clear) 하는 등의 작업을 편집 위치에 상관없이 언제나 가능하도록 하기 위해서 필요하다.

“질의 버퍼”는 질의문을 실행하기 전까지 질의문을 저장하는 버퍼이다. **--no-single-line** 옵션을 부여하여 CSQL을 실행하는 경우 ;xr 명령으로 질의를 실행하기 전까지는 질의문을 버퍼에 유지한다.

세션 명령어 도움말(HElp);HElp 는 전체 세션 명령어에 대한 도움을 보여주는 명령어로, csql의 세션 명령어를 확인하는 용도로 사용할 수 있다.

```
csql> ;help
=== <Help: Session Command Summary> ===

All session commands should be prefixed by ';' and only blanks/tabs
can precede the prefix. Capitalized characters represent the minimum
```

abbreviation that you need to enter to execute the specified command.

```

;REAd    [<file-name>]      - read a file into command buffer.
;Write   [<file-name>]      - (over)write command buffer into a
file.
;APpend  [<file-name>]      - append command buffer into a file.
;PRINT   - print command buffer.
;SHELL   - invoke shell.
;CD       - change current working directory.
;EXit (or Ctrl+d)          - exit program.

;Clear    - clear command buffer.
;EDIT     [format/fmt]      - invoke system editor [after
formatter] with command buffer.
;LISt     - display the content of command buffer.

;RUn      - execute sql in command buffer.
;Xrun     - execute sql in command buffer,
and clear the command buffer.

;COMmit   - commit the current transaction.
;ROLLback - roll back the current transaction.
;AUTocommit [ON|OFF]       - enable/disable auto commit mode.
;REStart  - reconnect to the current database in
a CSQL session.
;CHECKpoint - execute the checkpoint(CSQL with --
sysadm only).
;Killtran - check transaction status information
or end a specific transaction.(CSQL with --sysadm only).

;SHELL_Cmd    [shell-cmd] - set default shell, editor, print,
pager and formatter
;EDITOR_Cmd   [editor-cmd] command to new one, or display the
current
;PRINT_Cmd    [print-cmd]  one, respectively.
;PAger_cmd    [pager-cmd]
;FOrmatter_cmd [formatter-cmd]

;DATE         - display the local time, date.
;DATAbase     - display the name of database being
accessed.
;SCHEMA class-name - display schema information of a class.
;TRIGGER [*'|trigger-name] - display trigger definition.
;GET system_parameter - get the value of a system parameter.
;SET system_parameter=value - set the value of a system parameter.
;STRING-width [width] - set width that each column which is a
string type is displayed.
;COLUMN-width [name]=[width] - set width that a specific column is
displayed.
;PLAN [simple/detail/off] - show query execution plan.
;INFO <command> - display internal information.
;TIME [ON/OFF] - enable/disable to display the query
execution time.

```

```

;Server-output [ON/OFF]      - enable/disable displaying server
messages stored in DBMS_OUTPUT buffer.
;Line-output [ON/OFF]        - enable/disable to display each value
in a line
;HistoryList                  - display list of the executed queries.
;HistoryRead <history_num>    - read entry on the history number into
command buffer.
;Trace [ON/OFF] [text/json]   - enable/disable sql auto trace.
;Singleline [ON|OFF]          - enable/disable single-line mode.
;Connect username [dbname | dbname@hostname]
                              - connect to the current or other
databases as a username.
;.Hist [ON/OFF]               - start/stop collecting statistics
information in CSQL(available DBA only).
;.Clear_hist                  - clear the CSQL statistics information
in the buffer.
;.Dump_hist                   - display the CSQL statistics
information in CSQL.
;.X_hist                      - display the CSQL statistics
information in CSQL with statistics data initialized.
;Help                          - display this help message.

```

파일에서 질의 읽기(;REAd)

;REAd 명령어는 파일의 내용을 질의 버퍼로 읽는 세션 명령어로, 지정된 입력 파일에 저장된 질의문들을 실행하는데 사용할 수 있다. 질의 버퍼에 올려진 파일 내용을 보기 위해서는 **;List** 명령어를 사용한다.

```

csql> ;read nation.sql
The file has been read into the command buffer.
csql> ;list
insert into "sport_event" ("event_code", "event_name",
"gender_type", "num_player") values
(20001, 'Archery Individual', 'M', 1);
insert into "sport_event" ("event_code", "event_name",
"gender_type", "num_player") values
20002, 'Archery Individual', 'W', 1);
....

```

파일에 질의 저장(;WRite)

;WRite 는 질의 버퍼의 내용을 파일에 저장하는 세션 명령어로 사용자가 CSQL 인터프리터에서 입력 혹은 수정한 질의문을 파일에 저장할 때 사용된다.

```

csql> ;write outfile
Command buffer has been saved.

```

파일에 덧붙이기(;APpend)

현재 질의 버퍼의 내용을 출력 파일인 **outfile**에 추가한다.

```
csql> ;append outfile
Command buffer has been saved.
```

셸 명령어를 실행(;SHELL)

;SHELL 세션 명령어로 외부 셸을 호출할 수 있다. CSQL 인터프리터가 실행된 환경에서 새로운 셸이 시작되고, 셸을 마치면 다시 CSQL 인터프리터로 돌아온다. 만약에 **;SHELL_Cmd** 명령어로 수행할 셸 명령어가 지정되어 있다면 셸을 구동하여 지정된 명령어를 실행하고 CSQL 인터프리터로 복귀하게 된다.

```
csql> ;shell
% ls -al
total 2088
drwxr-xr-x 16 DBA cubrid 4096 Jul 29 16:51 .
drwxr-xr-x 6 DBA cubrid 4096 Jul 29 16:17 ..
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 02:49 audit
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 16:17 bin
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 16:17 conf
drwxr-xr-x 4 DBA cubrid 4096 Jul 29 16:14 cubridmanager
% exit
csql>
```

셸 명령어 등록(;SHELL_Cmd)

;SHELL_Cmd를 사용하여 **;SHELL** 세션 명령어로 실행할 셸 명령어를 등록한다. 등록된 명령어를 실행하기 위해서는 예제와 같이 **;shell** 명령어를 입력한다.

```
csql> ;shell_c ls -la
csql> ;shell
total 2088
drwxr-xr-x 16 DBA cubrid 4096 Jul 29 16:51 .
drwxr-xr-x 6 DBA cubrid 4096 Jul 29 16:17 ..
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 02:49 audit
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 16:17 bin
drwxr-xr-x 2 DBA cubrid 4096 Jul 29 16:17 conf
drwxr-xr-x 4 DBA cubrid 4096 Jul 29 16:14 cubridmanager
csql>
```

OS 환경변수(CUBRID_CSQ_SHELL 또는 SHELL, 두 환경변수가 설정된 경우 CUBRID_CSQ_SHELL 사용)를 미리 등록하여 사용할 수도 있다.

```
$ export CUBRID_CSQL_SHELL='ls -la'
또는
$ export SHELL='ls -la'
```

페이저 명령어 등록(;PAGER_cmd)

;PAGER_cmd를 사용하여 질의 실행 결과를 출력하는 페이저 명령어를 등록한다. 등록되는 명령어에 따라 출력되는 방식이 결정된다. 기본 명령어는 **more** 이며, **cat**, **less** 등이 사용될 수 있다. 단, 이 명령어는 Linux에서만 정상 동작한다.

페이저 명령어를 **more** 로 등록하는 경우 질의 결과를 페이지 단위로 출력하고, 스페이스 키가 눌러질 때까지 다음 페이지의 출력을 대기한다.

```
csql>;pager more
```

페이저 명령어를 **cat**으로 등록하는 경우 페이지링 없이 질의 결과 전체를 출력한다.

```
csql>;pager cat
```

output.txt로 출력을 리다이렉션하면 질의 결과 전체를 output.txt에 기록한다.

```
csql>;pager cat > output.txt
```

페이저 명령어를 **less** 로 등록하는 경우 질의 결과에 대해 포워딩, 백워딩을 할 수 있고 패턴 검색도 할 수 있다.

```
csql>;pager less
```

less 에서 사용하는 키보드 명령은 다음과 같다.

- Page UP, b: 한 페이지 뒤로 가기(백워딩)
- Page Down, Space: 한 페이지 앞으로 가기(포워딩)
- /문자열: 질의 결과에서 문자열 찾기
- n: 다음 문자열 찾기
- N: 앞의 문자열 찾기
- q: 질의 결과 보기 종료하기

현재 작업 디렉터리 변경(;CD)

CSQL 인터프리터를 실행한 현재 작업 디렉터리를 지정된 디렉터리로 변경한다. 경로를 지정하지 않으면 홈 디렉터리로 변경된다.

```
csql> ;cd /home1/DBA/CUBRID
Current directory changed to /home1/DBA/CUBRID.
```

CSQL 인터프리터 종료(;Exit)

CSQL 인터프리터를 종료한다.

```
csql> ;ex
```

질의 버퍼 초기화(;Clear)

;Clear 세션 명령어는 질의 버퍼의 내용을 초기화한다.

```
csql> ;clear
csql> ;list
```

질의 버퍼의 내용 보여주기(;List)

현재까지 입력 수정된 질의 버퍼의 내용을 화면에 출력하기 위해서는 **;List** 세션 명령어를 사용한다. 질의 버퍼는 사용자의 SQL 입력, **;Read** 명령어, **;EDIT** 명령어 등으로 수정될 수 있다.

```
csql> ;list
```

SQL 문 실행(;Run)

질의 버퍼에 있는 SQL 문을 실행하는 명령어이다. 다음에서 설명하는 **;Xrun** 세션 명령어와 달리 질의 실행 후에도 버퍼는 초기화되지 않는다.

```
csql> ;run
```

SQL 문 실행 후 질의 버퍼 초기화(;Xrun)

질의 버퍼에 있는 SQL 문을 실행하는 명령어이다. 질의 실행 후 질의 버퍼는 초기화된다.

```
csql> ;xrun
```

트랜잭션 커밋(;Commit)

현재 수행되고 있는 트랜잭션을 커밋(commit)하는 세션 명령어이다. 자동 커밋(auto-commit) 모드가 아닌 경우, 명시적으로 커밋 명령어를 입력해야 CSQL 인터프리터에서 수행 중이던 트랜잭션이 커밋된다. 자동 커밋(auto-commit) 모드인 경우는 SQL을 실행할 때마다 트랜잭션이 자동으로 커밋된다.

```
csql> ;commit
Execute OK. (0.000192 sec)
```

트랜잭션 롤백(;Rollback)

현재 수행되고 있는 트랜잭션을 롤백(rollback)하는 세션 명령어이다. **;Commit** 과 마찬가지로 자동 커밋(auto-commit) 모드가 아닐 경우(OFF)에만 의미가 있다.

```
csql> ;rollback
Execute OK. (0.000166 sec)
```

자동 커밋 모드 설정(;Autocommit)

자동 커밋(auto-commit) 모드를 **ON** 또는 **OFF** 로 설정하는 명령어이다. 만약, **ON** 또는 **OFF** 를 지정하지 않으면 현재 설정된 값을 보여준다. 참고로 CSQL 인터프리터는 기본값이 **ON** 이다.

```
csql> ;autocommit off
AUTOCOMMIT IS OFF
```

체크포인트 수행(;Checkpoint)

CSQL 세션 내에서 체크포인트 수행을 지시하는 명령어이다. CSQL 인터프리터 접속 시 사용자 지정 옵션(**-u user_name**)에 **DBA** 그룹 멤버가 지정되고 시스템 관리자 모드(**--sysadm**)로 접속한 경우에만 수행할 수 있다.

체크포인트는 현재 데이터 버퍼에 존재하는 페이지 중 임시 페이지를 제외한 모든 더티 페이지를 디스크로 내려쓰기(flush)하는 작업이며, CSQL 세션 내에서 파라미터 값을 설정하는 명령어(**;set parameter_name value**)를 통해서도 체크포인트 주기를 변경할 수 있다. 체크포인트 수행 주기와 관련된 파라미터는 **checkpoint_interval**과 **checkpoint_every_size** 가 있다. 이에 대한 자세한 내용은 [로그 관련 파라미터](#) 를 참고한다.

```
sysadm> ;checkpoint
Checkpoint has been issued.
```

트랜잭션 모니터링 또는 종료(;Killtran)

CSQL 세션 내에서 트랜잭션 상태 정보를 확인하거나 특정 트랜잭션을 종료시키는 명령어이다. CSQL 인터프리터 접속 시 사용자 지정 옵션(**-u user_name**)에 **DBA** 그룹 멤버가 지정되고 시스템 관

리자 모드(--sysadm)로 접속한 경우에만 수행할 수 있다. 인자가 생략되면 모든 트랜잭션 상태 정보를 화면 출력하고, 인자로 특정 트랜잭션 ID가 지정되면 해당 트랜잭션을 종료시킨다.

```
sysadm> ;killtran
Tran index      User name      Host name      Process id
Program name
-----
1(+)           dba            myhost         664
cub_cas
2(+)           dba            myhost         6700
csql
3(+)           dba            myhost         2188
cub_cas
4(+)           dba            myhost         696
csql
5(+)           public         myhost         6944
csql

sysadm> ;killtran 3
The specified transaction has been killed.
```

데이터베이스 재접속(;REstart)

CSQL 세션 내에서 대상 데이터베이스에 재접속을 시도하는 명령어이다. CSQL 인터프리터를 클라이언트/서버 모드(CS 모드)로 수행하는 경우에는 서버와의 접속이 해제되므로 유의한다. 이 명령어는 HA 환경에서 장애로 인해 다른 서버로 절체가 이루어짐에 따라 도중에 서버와의 연결이 해제되는 경우, 세션을 유지하면서 절체된 서버로 재접속할 때 유용하게 사용할 수 있다.

```
csql> ;restart
The database has been restarted.
```

현재 날짜 출력(;DATE)

;DATE 는 CSQL 인터프리터에서 현재 날짜 및 시간 정보를 출력한다.

```
csql> ;date
Tue July 29 18:58:12 KST 2008
```

대상 데이터베이스 정보 출력(;DATAbase)

CSQL 인터프리터에서 작업 중인 데이터베이스 이름 및 호스트 이름을 출력한다. 만약, 대상 데이터베이스가 HA모드로 동작 중이라면 현재 HA모드(active, standby, 또는 maintenance)도 함께 출력될 것이다.

```
csql> ;database
demodb@cubridhost (active)
```

지정한 테이블의 스키마 정보 출력(;Schema)

;Schema 세션 명령어로 지정한 테이블의 스키마 정보를 확인할 수 있다. 해당 테이블의 이름, 컬럼명, 제약 사항 등의 정보가 출력된다.

```
csql> ;schema event
=== <Help: Schema of a Class> ===
<Class Name>
    event
<Attributes>
    code          INTEGER NOT NULL
    sports        CHARACTER VARYING(50)
    name          CHARACTER VARYING(50)
    gender        CHARACTER(1)
    players       INTEGER
<Constraints>
    PRIMARY KEY pk_event_event_code ON event (code)
```

트리거 출력(;Trigger)

지정한 트리거 이름을 검색하여 출력하는 명령어이다. 트리거 이름을 지정하지 않으면 정의된 모든 트리거를 보여준다.

```
csql> ;trigger
=== <Help: All Triggers> ===
    trig_delete_contents
```

파라미터 값 확인(;Get)

;Get 세션 명령어를 이용해 현재 CSQL 인터프리터에 설정된 파라미터 값을 확인할 수 있다. 지정된 파라미터 명이 정확하지 않으면 오류가 발생한다.

```
csql> ;get isolation_level
=== Get Param Input ===
isolation_level="tran_rep_class_commit_instance"
```

파라미터 값 설정(;SET)

특정 파라미터의 값을 설정하기 위해서는 **;SET** 세션 명령어를 사용한다. 동적 변경이 가능한 파라미터만 값을 변경할 수 있으며, 서버 파라미터는 DBA 권한이 있어야만 값을 변경할 수 있다. 동적 변경이 가능한 파라미터 목록은 [브로커 설정](#) 를 참고한다.

```

csql> ;set block_ddl_statement=1
=== Set Param Input ===
block_ddl_statement=1

-- dba 계정으로 실행한 csql에서 log_max_archives 값을 동적으로 변경
csql> ;set log_max_archives=5

```

문자열 타입과 비트 타입 칼럼의 출력 길이 지정(;String-width)

문자열 타입과 비트 타입 칼럼의 출력 길이를 제한하기 위해서 사용할 수 있다.

;string-width 뒤에 값을 주지 않으면 현재의 출력 길이를 보여준다. 값이 0이면, 해당 칼럼의 값을 모두 출력한다. 값이 0보다 크다면, 해당 길이만큼 칼럼의 값을 출력한다.

```

csql> SELECT name FROM NATION WHERE NAME LIKE 'Ar%';
'Arab Republic of Egypt'
'Aruba'
'Armenia'
'Argentina'

csql> ;string-width 5
csql> SELECT name FROM NATION WHERE NAME LIKE 'Ar%';
'Arab '
'Aruba'
'Armen'
'Argen'

csql> ;string-width
STRING-WIDTH : 5

```

지정한 칼럼의 출력 길이 지정(;COLumn-width)

타입과 상관없이 특정 칼럼의 출력 길이를 제한하기 위해서 사용할 수 있다. ;COL 뒤에 값을 주지 않으면 현재 설정된 칼럼의 출력 길이를 보여준다. 뒤에 값이 0이면 해당 칼럼의 값을 모두 출력하며, 값이 0보다 크다면 해당 길이만큼 칼럼의 값을 출력한다.

```

csql> CREATE TABLE tbl(a BIGINT, b BIGINT);
csql> INSERT INTO tbl VALUES(12345678890, 1234567890)
csql> ;column-width a=5
csql> SELECT * FROM tbl;
12345          1234567890
csql> ;column-width
COLUMN-WIDTH a : 5

```

질의 실행 계획 보기 수준 설정(PLan)

;Plan 세션 명령어는 질의 실행 계획 보기의 수준을 설정한다. 수준은 **simple**, **detail**, **off** 로 지정한다. 각 설정값의 의미는 다음과 같다.

- **off**: 질의 실행 계획을 출력하지 않음(OPT LEVEL=1)
- **simple**: 질의 실행 계획을 단순하게 출력함. (OPT LEVEL=257)
- **detail**: 질의 실행 계획을 자세하게 출력함. (OPT LEVEL=513)

OPT LEVEL의 상세한 내용은 [질의 실행 계획 보기](#)를 참고한다.

SQL 트레이스 설정(;trace)

질의 결과를 출력할 때 SQL 트레이스 결과를 항상 함께 출력할 것인지 여부를 설정한다. 이 명령을 사용하여 SQL 트레이스를 ON으로 설정하면, “**SHOW TRACE**” 구문을 실행하지 않아도 질의 결과를 출력한 다음에 질의 프로파일링(profiling) 결과를 자동으로 출력한다.

보다 자세한 설명은 [질의 프로파일링](#)를 참고한다.

명령 형식은 다음과 같다.

```
;trace {on | off} [{text | json}]
```

- **on**: SQL 트레이스를 on한다.
- **off**: SQL 트레이스를 off한다.
- **text**: 일반 TEXT 형식으로 출력한다. OUTPUT 이하 절을 생략하면 일반 TEXT 형식으로 출력한다.
- **json**: JSON 형식으로 출력한다.

참고

독립 모드(-S 옵션 사용)로 실행한 CSQL 인터프리터는 SQL 트레이스 기능을 지원하지 않는다.

정보 출력(;Info)

;Info 세션 명령어는 스키마, 트리거, 작업 환경, 잠금, 통계 등의 정보를 확인할 수 있는 명령어이다.

```
csql> ;info lock
*** Lock Table Dump ***
Lock Escalation at = 100000, Run Deadlock interval = 1
Transaction (index 0, unknown, unknown@unknown|-1)
Isolation COMMITTED READ
State TRAN_ACTIVE
```

```
Timeout_period -1
.....
```

CSQL 실행 통계 정보 출력(;.Hist)

;**Hist** 세션 명령어는 CSQL에서 질의 실행 통계 정보의 수집을 시작하기 위한 CSQL 세션 명령어로
서, 이 정보는 “;**Hist on**”이 입력된 이후부터 현재 연결된 CSQL에 대해서만 수집된다. 다음은 이 세
션 명령어의 ;**Hist** 세션 명령어의 옵션으로 **on**, **off**를 제공하며, 각 옵션의 의미는 다음과 같다.

- **on**: 해당 연결에 대한 서버 실행 통계 정보 수집을 시작.
- **off**: 서버 실행 통계 정보 수집을 종료.

단, **cubrid.conf** 파일에서 관련 파라미터(**communication_histogram**)를 **yes**로 설정하거나, **csql**에서 “;se communication_histogram=yes”를 실행해야만 이 명령어를 사용할 수 있다.

“;**Hist on**” 이후 서버 실행 통계 정보를 화면에 출력하기 위해서는 **;**dump_hist**** 또는 **;**x****와 같은 실행 명령어를 입력해야 한다. **;**dump_hist**** 또는 **;**x****를 수행할 때마다, 축적된 값이 출력된 후 모든 값이 초기화된다.

참고로, DB 서버의 모든 질의 실행 통계 정보를 확인하기 위해서는 **cubrid statdump** 유틸리티를 사용해야 한다.

다음 예제는 현재 연결에 대한 서버 실행 통계 정보를 확인하는 예제이다. 출력되는 통계 정보 항목 또는 **cubrid statdump**에 대한 설명은 **statdump**을 참고한다.

```
csql> ;set communication_histogram=yes
csql> ;.hist on
csql> SELECT name FROM nation LIMIT 5;
csql> ;.x
Histogram of client requests:
```

Name	Rcount	Sent size	Recv size ,
Server time			
LC_FETCHALL 0.000000 s	2 X	72+	0 b,
LC_DOESEXIST 0.000142 s	3 X	84+	36 b,
LC_FETCH_LOCKHINT_CLASSES 0.000000 s	1 X	56+	0 b,
BTREE_GET_KEY_TYPE 0.000111 s	5 X	180+	60 b,
QM_QUERY_DROP_ALL_PLANS 0.000080 s	1 X	224+	40 b,
QM_QUERY_DUMP_PLANS 0.000355 s	1 X	72+	308 b,

Totals: 0.001415 s	13 X	688+	444 b

Average server response time = 0.000109 secs
 Average time between client requests = 0.000000 secs

*** CLIENT EXECUTION STATISTICS ***

System CPU (sec)	=	0
User CPU (sec)	=	0
Elapsed (sec)	=	20

*** SERVER EXECUTION STATISTICS ***

Num_file_creates	=	0
Num_file_removes	=	0
Num_file_ioreads	=	0
Num_file_iowrites	=	0
Num_file_iosynches	=	0
The timer values for file_iosync_all are:		
Num_file_iosync_all	=	0
Total_time_file_iosync_all	=	0
Max_time_file_iosync_all	=	0
Avg_time_file_iosync_all	=	0
Num_file_page_allocs	=	0
Num_file_page_deallocs	=	0
Num_data_page_fetches	=	58
Num_data_page_dirties	=	0
Num_data_page_ioreads	=	0
Num_data_page_iowrites	=	0
Num_data_page_flushed	=	0
Num_data_page_private_quota	=	551
Num_data_page_private_count	=	8
Num_data_page_fixed	=	1
Num_data_page_dirty	=	0
Num_data_page_lru1	=	22
Num_data_page_lru2	=	14
Num_data_page_lru3	=	8
Num_data_page_victim_candidate	=	8
Num_log_page_fetches	=	0
Num_log_page_ioreads	=	0
Num_log_page_iowrites	=	0
Num_log_append_records	=	0
Num_log_archives	=	0
Num_log_start_checkpoints	=	0
Num_log_end_checkpoints	=	0
Num_log_wals	=	0
Num_log_page_iowrites_for_replacement	=	0
Num_log_page_replacements	=	0
Num_page_locks_acquired	=	0
Num_object_locks_acquired	=	4
Num_page_locks_converted	=	0
Num_object_locks_converted	=	2
Num_page_locks_re-requested	=	0
Num_object_locks_re-requested	=	11
Num_page_locks_waits	=	0


```

Num_object_locks_waits           =          0
Num_object_locks_time_waited_usec =          0
Num_tran_commits                 =          1
Num_tran_rollbacks               =          0
Num_tran_savepoints              =          0
Num_tran_start_topops            =          0
Num_tran_end_topops              =          0
Num_tran_interrupts              =          0
Num_tran_postpone_cache_hits     =          0
Num_tran_postpone_cache_miss     =          0
Num_tran_topop_postpone_cache_hits =          0
Num_tran_topop_postpone_cache_miss =          0
Num_btree_inserts                =          0
Num_btree_deletes                =          0
Num_btree_updates                =          0
Num_btree_covered                =          0
Num_btree_noncovered             =          0
Num_btree_resumes                =          0
Num_btree_multirange_optimization =          0
Num_btree_splits                 =          0
Num_btree_merges                 =          0
Num_btree_get_stats              =          0
The timer values for btree_online_load are:
Num_btree_online_load            =          0
Total_time_btree_online_load     =          0
Max_time_btree_online_load       =          0
Avg_time_btree_online_load       =          0
The timer values for btree_online_insert_task are:
Num_btree_online_insert_task     =          0
Total_time_btree_online_insert_task =          0
Max_time_btree_online_insert_task =          0
Avg_time_btree_online_insert_task =          0
The timer values for btree_online_prepare_task are:
Num_btree_online_prepare_task    =          0
Total_time_btree_online_prepare_task =          0
Max_time_btree_online_prepare_task =          0
Avg_time_btree_online_prepare_task =          0
The timer values for btree_online_insert_leaf are:
Num_btree_online_insert_leaf     =          0
Total_time_btree_online_insert_leaf =          0
Max_time_btree_online_insert_leaf =          0
Avg_time_btree_online_insert_leaf =          0
Num_btree_online_inserts         =          0
Num_btree_online_inserts_same_page_hold =          0
Num_btree_online_inserts_retry   =          0
Num_btree_online_inserts_retry_nice =          0
Num_query_selects                =          1
Num_query_inserts                =          0
Num_query_deletes                =          0
Num_query_updates                =          0
Num_query_sscans                 =          1
Num_query_iscans                 =          0

```

```

Num_query_lscans           = 0
Num_query_setscans         = 0
Num_query_methscans        = 0
Num_query_nljoins          = 0
Num_query_mjoins           = 0
Num_query_objfetches       = 0
Num_query_holdable_cursors = 0
Num_sort_io_pages          = 0
Num_sort_data_pages        = 0
Num_network_requests       = 14
Num_adaptive_flush_pages   = 0
Num_adaptive_flush_log_pages = 0
Num_adaptive_flush_max_pages = 0
Num_prior_lsa_list_size    = 0
Num_prior_lsa_list_maxed   = 0
Num_prior_lsa_list_removed = 0
Time_ha_replication_delay  = 0
Num_plan_cache_add         = 0
Num_plan_cache_lookup      = 2
Num_plan_cache_hit         = 2
Num_plan_cache_miss        = 0
Num_plan_cache_full        = 0
Num_plan_cache_delete      = 0
Num_plan_cache_invalid_xasl_id = 0
Num_plan_cache_entries     = 1
Num_vacuum_log_pages_vacuumed = 0
Num_vacuum_log_pages_to_vacuum = 0
Num_vacuum_prefetch_requests_log_pages = 0
Num_vacuum_prefetch_hits_log_pages = 0
Num_heap_home_inserts      = 0
Num_heap_big_inserts       = 0
Num_heap_assign_inserts    = 0
Num_heap_home_deletes      = 0
Num_heap_home_mvcc_deletes = 0
Num_heap_home_to_rel_deletes = 0
Num_heap_home_to_big_deletes = 0
Num_heap_rel_deletes       = 0
Num_heap_rel_mvcc_deletes  = 0
Num_heap_rel_to_home_deletes = 0
Num_heap_rel_to_big_deletes = 0
Num_heap_rel_to_rel_deletes = 0
Num_heap_big_deletes       = 0
Num_heap_big_mvcc_deletes  = 0
Num_heap_home_updates      = 0
Num_heap_home_to_rel_updates = 0
Num_heap_home_to_big_updates = 0
Num_heap_rel_updates       = 0
Num_heap_rel_to_home_updates = 0
Num_heap_rel_to_rel_updates = 0
Num_heap_rel_to_big_updates = 0
Num_heap_big_updates       = 0
Num_heap_home_vacuums      = 0

```

```

Num_heap_big_vacuums           = 0
Num_heap_rel_vacuums           = 0
Num_heap_insid_vacuums         = 0
Num_heap_remove_vacuums        = 0
The timer values for heap_insert_prepare are:
Num_heap_insert_prepare        = 0
Total_time_heap_insert_prepare = 0
Max_time_heap_insert_prepare   = 0
Avg_time_heap_insert_prepare   = 0
The timer values for heap_insert_execute are:
Num_heap_insert_execute         = 0
Total_time_heap_insert_execute = 0
Max_time_heap_insert_execute    = 0
Avg_time_heap_insert_execute    = 0
The timer values for heap_insert_log are:
Num_heap_insert_log             = 0
Total_time_heap_insert_log      = 0
Max_time_heap_insert_log        = 0
Avg_time_heap_insert_log        = 0
The timer values for heap_delete_prepare are:
Num_heap_delete_prepare         = 0
Total_time_heap_delete_prepare = 0
Max_time_heap_delete_prepare    = 0
Avg_time_heap_delete_prepare    = 0
The timer values for heap_delete_execute are:
Num_heap_delete_execute         = 0
Total_time_heap_delete_execute = 0
Max_time_heap_delete_execute    = 0
Avg_time_heap_delete_execute    = 0
The timer values for heap_delete_log are:
Num_heap_delete_log             = 0
Total_time_heap_delete_log      = 0
Max_time_heap_delete_log        = 0
Avg_time_heap_delete_log        = 0
The timer values for heap_update_prepare are:
Num_heap_update_prepare         = 0
Total_time_heap_update_prepare = 0
Max_time_heap_update_prepare    = 0
Avg_time_heap_update_prepare    = 0
The timer values for heap_update_execute are:
Num_heap_update_execute         = 0
Total_time_heap_update_execute = 0
Max_time_heap_update_execute    = 0
Avg_time_heap_update_execute    = 0
The timer values for heap_update_log are:
Num_heap_update_log             = 0
Total_time_heap_update_log      = 0
Max_time_heap_update_log        = 0
Avg_time_heap_update_log        = 0
The timer values for heap_vacuum_prepare are:
Num_heap_vacuum_prepare         = 0
Total_time_heap_vacuum_prepare = 0

```

```

Max_time_heap_vacuum_prepare = 0
Avg_time_heap_vacuum_prepare = 0
The timer values for heap_vacuum_execute are:
Num_heap_vacuum_execute = 0
Total_time_heap_vacuum_execute = 0
Max_time_heap_vacuum_execute = 0
Avg_time_heap_vacuum_execute = 0
The timer values for heap_vacuum_log are:
Num_heap_vacuum_log = 0
Total_time_heap_vacuum_log = 0
Max_time_heap_vacuum_log = 0
Avg_time_heap_vacuum_log = 0
The timer values for heap_stats_sync_bestspace are:
Num_heap_stats_sync_bestspace = 0
Total_time_heap_stats_sync_bestspace = 0
Max_time_heap_stats_sync_bestspace = 0
Avg_time_heap_stats_sync_bestspace = 0
Num_heap_stats_bestspace_entries = 0
Num_heap_stats_bestspace_maxed = 0
The timer values for bestspace_add are:
Num_bestspace_add = 0
Total_time_bestspace_add = 0
Max_time_bestspace_add = 0
Avg_time_bestspace_add = 0
The timer values for bestspace_del are:
Num_bestspace_del = 0
Total_time_bestspace_del = 0
Max_time_bestspace_del = 0
Avg_time_bestspace_del = 0
The timer values for bestspace_find are:
Num_bestspace_find = 0
Total_time_bestspace_find = 0
Max_time_bestspace_find = 0
Avg_time_bestspace_find = 0
The timer values for heap_find_page_bestspace are:
Num_heap_find_page_bestspace = 0
Total_time_heap_find_page_bestspace = 0
Max_time_heap_find_page_bestspace = 0
Avg_time_heap_find_page_bestspace = 0
The timer values for heap_find_best_page are:
Num_heap_find_best_page = 0
Total_time_heap_find_best_page = 0
Max_time_heap_find_best_page = 0
Avg_time_heap_find_best_page = 0
The timer values for bt_fix_ovf_oids are:
Num_bt_fix_ovf_oids = 0
Total_time_bt_fix_ovf_oids = 0
Max_time_bt_fix_ovf_oids = 0
Avg_time_bt_fix_ovf_oids = 0
The timer values for bt_unique_rlocks are:
Num_bt_unique_rlocks = 0
Total_time_bt_unique_rlocks = 0

```

```

Max_time_bt_unique_rlocks      =      0
Avg_time_bt_unique_rlocks      =      0
The timer values for bt_unique_wlocks are:
Num_bt_unique_wlocks           =      0
Total_time_bt_unique_wlocks     =      0
Max_time_bt_unique_wlocks       =      0
Avg_time_bt_unique_wlocks       =      0
The timer values for bt_leaf are:
Num_bt_leaf                     =      0
Total_time_bt_leaf              =      0
Max_time_bt_leaf                =      0
Avg_time_bt_leaf                =      0
The timer values for bt_traverse are:
Num_bt_traverse                 =      5
Total_time_bt_traverse          =     162
Max_time_bt_traverse            =      51
Avg_time_bt_traverse            =      32
The timer values for bt_find_unique are:
Num_bt_find_unique              =      0
Total_time_bt_find_unique       =      0
Max_time_bt_find_unique         =      0
Avg_time_bt_find_unique         =      0
The timer values for bt_find_unique_traverse are:
Num_bt_find_unique_traverse     =      5
Total_time_bt_find_unique_traverse =     165
Max_time_bt_find_unique_traverse =      53
Avg_time_bt_find_unique_traverse =      33
The timer values for bt_range_search are:
Num_bt_range_search             =      0
Total_time_bt_range_search       =      0
Max_time_bt_range_search         =      0
Avg_time_bt_range_search         =      0
The timer values for bt_range_search_traverse are:
Num_bt_range_search_traverse    =      0
Total_time_bt_range_search_traverse =      0
Max_time_bt_range_search_traverse =      0
Avg_time_bt_range_search_traverse =      0
The timer values for bt_insert are:
Num_bt_insert                   =      0
Total_time_bt_insert            =      0
Max_time_bt_insert              =      0
Avg_time_bt_insert              =      0
The timer values for bt_insert_traverse are:
Num_bt_insert_traverse          =      0
Total_time_bt_insert_traverse    =      0
Max_time_bt_insert_traverse      =      0
Avg_time_bt_insert_traverse      =      0
The timer values for bt_delete_obj are:
Num_bt_delete_obj               =      0
Total_time_bt_delete_obj         =      0
Max_time_bt_delete_obj           =      0
Avg_time_bt_delete_obj           =      0

```

```
The timer values for bt_delete_obj_traverse are:
Num_bt_delete_obj_traverse      =      0
Total_time_bt_delete_obj_traverse =      0
Max_time_bt_delete_obj_traverse =      0
Avg_time_bt_delete_obj_traverse =      0
The timer values for bt_mvcc_delete are:
Num_bt_mvcc_delete              =      0
Total_time_bt_mvcc_delete       =      0
Max_time_bt_mvcc_delete         =      0
Avg_time_bt_mvcc_delete         =      0
The timer values for bt_mvcc_delete_traverse are:
Num_bt_mvcc_delete_traverse     =      0
Total_time_bt_mvcc_delete_traverse =      0
Max_time_bt_mvcc_delete_traverse =      0
Avg_time_bt_mvcc_delete_traverse =      0
The timer values for bt_mark_delete are:
Num_bt_mark_delete              =      0
Total_time_bt_mark_delete       =      0
Max_time_bt_mark_delete         =      0
Avg_time_bt_mark_delete         =      0
The timer values for bt_mark_delete_traverse are:
Num_bt_mark_delete_traverse     =      0
Total_time_bt_mark_delete_traverse =      0
Max_time_bt_mark_delete_traverse =      0
Avg_time_bt_mark_delete_traverse =      0
The timer values for bt_undo_insert are:
Num_bt_undo_insert              =      0
Total_time_bt_undo_insert       =      0
Max_time_bt_undo_insert         =      0
Avg_time_bt_undo_insert         =      0
The timer values for bt_undo_insert_traverse are:
Num_bt_undo_insert_traverse     =      0
Total_time_bt_undo_insert_traverse =      0
Max_time_bt_undo_insert_traverse =      0
Avg_time_bt_undo_insert_traverse =      0
The timer values for bt_undo_delete are:
Num_bt_undo_delete              =      0
Total_time_bt_undo_delete       =      0
Max_time_bt_undo_delete         =      0
Avg_time_bt_undo_delete         =      0
The timer values for bt_undo_delete_traverse are:
Num_bt_undo_delete_traverse     =      0
Total_time_bt_undo_delete_traverse =      0
Max_time_bt_undo_delete_traverse =      0
Avg_time_bt_undo_delete_traverse =      0
The timer values for bt_undo_mvcc_delete are:
Num_bt_undo_mvcc_delete         =      0
Total_time_bt_undo_mvcc_delete =      0
Max_time_bt_undo_mvcc_delete    =      0
Avg_time_bt_undo_mvcc_delete    =      0
The timer values for bt_undo_mvcc_delete_traverse are:
Num_bt_undo_mvcc_delete_traverse =      0
```

```

Total_time_bt_undo_mvcc_delete_traverse =          0
Max_time_bt_undo_mvcc_delete_traverse =          0
Avg_time_bt_undo_mvcc_delete_traverse =          0
The timer values for bt_vacuum are:
Num_bt_vacuum =          0
Total_time_bt_vacuum =          0
Max_time_bt_vacuum =          0
Avg_time_bt_vacuum =          0
The timer values for bt_vacuum_traverse are:
Num_bt_vacuum_traverse =          0
Total_time_bt_vacuum_traverse =          0
Max_time_bt_vacuum_traverse =          0
Avg_time_bt_vacuum_traverse =          0
The timer values for bt_vacuum_insid are:
Num_bt_vacuum_insid =          0
Total_time_bt_vacuum_insid =          0
Max_time_bt_vacuum_insid =          0
Avg_time_bt_vacuum_insid =          0
The timer values for bt_vacuum_insid_traverse are:
Num_bt_vacuum_insid_traverse =          0
Total_time_bt_vacuum_insid_traverse =          0
Max_time_bt_vacuum_insid_traverse =          0
Avg_time_bt_vacuum_insid_traverse =          0
The timer values for vacuum_master are:
Num_vacuum_master =          0
Total_time_vacuum_master =          0
Max_time_vacuum_master =          0
Avg_time_vacuum_master =          0
The timer values for vacuum_job are:
Num_vacuum_job =          0
Total_time_vacuum_job =          0
Max_time_vacuum_job =          0
Avg_time_vacuum_job =          0
The timer values for vacuum_worker_process_log are:
Num_vacuum_worker_process_log =          0
Total_time_vacuum_worker_process_log =          0
Max_time_vacuum_worker_process_log =          0
Avg_time_vacuum_worker_process_log =          0
The timer values for vacuum_worker_execute are:
Num_vacuum_worker_execute =          0
Total_time_vacuum_worker_execute =          0
Max_time_vacuum_worker_execute =          0
Avg_time_vacuum_worker_execute =          0
Time_get_snapshot_acquire_time =          7
Count_get_snapshot_retry =          0
Time_tran_complete_time =          7
The timer values for compute_oldest_visible are:
Num_compute_oldest_visible =          0
Total_time_compute_oldest_visible =          0
Max_time_compute_oldest_visible =          0
Avg_time_compute_oldest_visible =          0
Count_get_oldest_mvcc_retry =          0

```

```

Data_page_buffer_hit_ratio      =      100.00
Log_page_buffer_hit_ratio      =       0.00
Vacuum_data_page_buffer_hit_ratio =       0.00
Vacuum_page_efficiency_ratio    =       0.00
Vacuum_page_fetch_ratio        =       0.00
Data_page_fix_lock_acquire_time_msec =       0.00
Data_page_fix_hold_acquire_time_msec =       0.00
Data_page_fix_acquire_time_msec =       0.25
Data_page_allocate_time_ratio   =      100.00
Data_page_total_promote_success =       0.00
Data_page_total_promote_fail   =       0.00
Data_page_total_promote_time_msec =       0.00
Num_unfix_void_to_private_top  =       0
Num_unfix_void_to_private_mid  =       0
Num_unfix_void_to_shared_mid   =       0
Num_unfix_lru1_private_to_shared_mid =       0
Num_unfix_lru2_private_to_shared_mid =       0
Num_unfix_lru3_private_to_shared_mid =       0
Num_unfix_lru2_private_keep    =       0
Num_unfix_lru2_shared_keep     =       0
Num_unfix_lru2_private_to_top  =       0
Num_unfix_lru2_shared_to_top   =       0
Num_unfix_lru3_private_to_top  =       4
Num_unfix_lru3_shared_to_top   =       0
Num_unfix_lru1_private_keep    =       3
Num_unfix_lru1_shared_keep     =      36
Num_unfix_void_to_private_mid_vacuum =       0
Num_unfix_lru1_any_keep_vacuum =       0
Num_unfix_lru2_any_keep_vacuum =       0
Num_unfix_lru3_any_keep_vacuum =       0
Num_unfix_void_aout_found      =       0
Num_unfix_void_aout_not_found  =       0
Num_unfix_void_aout_found_vacuum =       0
Num_unfix_void_aout_not_found_vacuum =       0
Num_data_page_hash_anchor_waits =       0
Time_data_page_hash_anchor_wait =       0
The timer values for flush_collect are:
Num_flush_collect              =       0
Total_time_flush_collect       =       0
Max_time_flush_collect         =       0
Avg_time_flush_collect         =       0
The timer values for flush_flush are:
Num_flush_flush                =       0
Total_time_flush_flush         =       0
Max_time_flush_flush           =       0
Avg_time_flush_flush           =       0
The timer values for flush_sleep are:
Num_flush_sleep                =       0
Total_time_flush_sleep         =       0
Max_time_flush_sleep           =       0
Avg_time_flush_sleep           =       0
The timer values for flush_collect_per_page are:

```



```

Num_flush_collect_per_page      =          0
Total_time_flush_collect_per_page =          0
Max_time_flush_collect_per_page  =          0
Avg_time_flush_collect_per_page  =          0
The timer values for flush_flush_per_page are:
Num_flush_flush_per_page        =          0
Total_time_flush_flush_per_page =          0
Max_time_flush_flush_per_page   =          0
Avg_time_flush_flush_per_page   =          0
Num_data_page_writes            =          0
Num_data_page_dirty_to_post_flush =          0
Num_data_page_skipped_flush     =          0
Num_data_page_skipped_flush_need_wal =          0
Num_data_page_skipped_flush_already_flushed =          0
Num_data_page_skipped_flush_fixed_or_hot =          0
The timer values for compensate_flush are:
Num_compensate_flush            =          0
Total_time_compensate_flush     =          0
Max_time_compensate_flush       =          0
Avg_time_compensate_flush       =          0
The timer values for assign_direct_bcb are:
Num_assign_direct_bcb          =          0
Total_time_assign_direct_bcb   =          0
Max_time_assign_direct_bcb     =          0
Avg_time_assign_direct_bcb     =          0
The timer values for wake_flush_waiter are:
Num_wake_flush_waiter          =          0
Total_time_wake_flush_waiter   =          0
Max_time_wake_flush_waiter     =          0
Avg_time_wake_flush_waiter     =          0
The timer values for alloc_bcb are:
Num_alloc_bcb                  =          0
Total_time_alloc_bcb           =          0
Max_time_alloc_bcb             =          0
Avg_time_alloc_bcb             =          0
The timer values for alloc_bcb_search_victim are:
Num_alloc_bcb_search_victim    =          0
Total_time_alloc_bcb_search_victim =          0
Max_time_alloc_bcb_search_victim =          0
Avg_time_alloc_bcb_search_victim =          0
The timer values for alloc_bcb_cond_wait_high_prio are:
Num_alloc_bcb_cond_wait_high_prio =          0
Total_time_alloc_bcb_cond_wait_high_prio =          0
Max_time_alloc_bcb_cond_wait_high_prio =          0
Avg_time_alloc_bcb_cond_wait_high_prio =          0
The timer values for alloc_bcb_cond_wait_low_prio are:
Num_alloc_bcb_cond_wait_low_prio =          0
Total_time_alloc_bcb_cond_wait_low_prio =          0
Max_time_alloc_bcb_cond_wait_low_prio =          0
Avg_time_alloc_bcb_cond_wait_low_prio =          0
Num_alloc_bcb_prioritize_vacuum =          0
Num_victim_use_invalid_bcb      =          0

```

```

The timer values for alloc_bcb_get_victim_search_own_private_list
are:
Num_alloc_bcb_get_victim_search_own_private_list = 0
Total_time_alloc_bcb_get_victim_search_own_private_list = 0
Max_time_alloc_bcb_get_victim_search_own_private_list = 0
Avg_time_alloc_bcb_get_victim_search_own_private_list = 0
The timer values for alloc_bcb_get_victim_search_others_private_list
are:
Num_alloc_bcb_get_victim_search_others_private_list = 0
Total_time_alloc_bcb_get_victim_search_others_private_list
= 0
Max_time_alloc_bcb_get_victim_search_others_private_list = 0
Avg_time_alloc_bcb_get_victim_search_others_private_list = 0
The timer values for alloc_bcb_get_victim_search_shared_list are:
Num_alloc_bcb_get_victim_search_shared_list = 0
Total_time_alloc_bcb_get_victim_search_shared_list = 0
Max_time_alloc_bcb_get_victim_search_shared_list = 0
Avg_time_alloc_bcb_get_victim_search_shared_list = 0
Num_victim_assign_direct_vacuum_void = 0
Num_victim_assign_direct_vacuum_lru = 0
Num_victim_assign_direct_flush = 0
Num_victim_assign_direct_panic = 0
Num_victim_assign_direct_adjust_lru = 0
Num_victim_assign_direct_adjust_lru_to_vacuum = 0
Num_victim_assign_direct_search_for_flush = 0
Num_victim_shared_lru_success = 0
Num_victim_own_private_lru_success = 0
Num_victim_other_private_lru_success = 0
Num_victim_shared_lru_fail = 0
Num_victim_own_private_lru_fail = 0
Num_victim_other_private_lru_fail = 0
Num_victim_all_lru_fail = 0
Num_victim_get_from_lru = 0
Num_victim_get_from_lru_was_empty = 0
Num_victim_get_from_lru_fail = 0
Num_victim_get_from_lru_bad_hint = 0
Num_lfcq_prv_get_total_calls = 0
Num_lfcq_prv_get_empty = 0
Num_lfcq_prv_get_big = 0
Num_lfcq_shr_get_total_calls = 0
Num_lfcq_shr_get_empty = 0
The timer values for DWB_flush_block are:
Num_DWB_flush_block = 0
Total_time_DWB_flush_block = 0
Max_time_DWB_flush_block = 0
Avg_time_DWB_flush_block = 0
The timer values for DWB_file_sync_helper are:
Num_DWB_file_sync_helper = 0
Total_time_DWB_file_sync_helper = 0
Max_time_DWB_file_sync_helper = 0
Avg_time_DWB_file_sync_helper = 0
The timer values for DWB_flush_block_cond_wait are:

```

```

Num_DWB_flush_block_cond_wait = 0
Total_time_DWB_flush_block_cond_wait = 0
Max_time_DWB_flush_block_cond_wait = 0
Avg_time_DWB_flush_block_cond_wait = 0
The timer values for DWB_flush_block_sort are:
Num_DWB_flush_block_sort = 0
Total_time_DWB_flush_block_sort = 0
Max_time_DWB_flush_block_sort = 0
Avg_time_DWB_flush_block_sort = 0
The timer values for DWB_decache_pages_after_write are:
Num_DWB_decache_pages_after_write = 0
Total_time_DWB_decache_pages_after_write = 0
Max_time_DWB_decache_pages_after_write = 0
Avg_time_DWB_decache_pages_after_write = 0
The timer values for DWB_wait_flush_block are:
Num_DWB_wait_flush_block = 0
Total_time_DWB_wait_flush_block = 0
Max_time_DWB_wait_flush_block = 0
Avg_time_DWB_wait_flush_block = 0
The timer values for DWB_wait_file_sync_helper are:
Num_DWB_wait_file_sync_helper = 0
Total_time_DWB_wait_file_sync_helper = 0
Max_time_DWB_wait_file_sync_helper = 0
Avg_time_DWB_wait_file_sync_helper = 0
The timer values for DWB_flush_force are:
Num_DWB_flush_force = 0
Total_time_DWB_flush_force = 0
Max_time_DWB_flush_force = 0
Avg_time_DWB_flush_force = 0
The timer values for Log_LZ4_compress are:
Num_Log_LZ4_compress = 0
Total_time_Log_LZ4_compress = 0
Max_time_Log_LZ4_compress = 0
Avg_time_Log_LZ4_compress = 0
The timer values for Log_LZ4_decompress are:
Num_Log_LZ4_decompress = 0
Total_time_Log_LZ4_decompress = 0
Max_time_Log_LZ4_decompress = 0
Avg_time_Log_LZ4_decompress = 0
Num_alloc_bcb_wait_threads_high_priority = 0
Num_alloc_bcb_wait_threads_low_priority = 0
Num_flushed_bcbs_wait_for_direct_victim = 0
Num_lfcq_big_private_lists = 0
Num_lfcq_private_lists = 1
Num_lfcq_shared_lists = 0
Num_data_page_avoid_dealloc = 0
Num_data_page_avoid_victim = 0
Num_data_page_fix_ext:
WORKER, PAGE_HEAP, OLD_PAGE_IN_PB, READ, UNCOND
= 8
WORKER, PAGE_VOLHEADER, OLD_PAGE_IN_PB, READ, UNCOND =
30

```

```

WORKER,PAGE_VOLBITMAP,OLD_PAGE_IN_PB      ,READ ,UNCOND      =
15
WORKER,PAGE_BTREE_R      ,OLD_PAGE_IN_PB      ,READ ,UNCOND
=          5
Num_data_page_promote_ext:
Num_data_page_promote_time_ext:
Num_data_page_unfix_ext:
WORKER,PAGE_HEAP      ,BUF_NON_DIRTY,HOLDER_NON_DIRTY,READ
=          8
WORKER,PAGE_VOLHEADER,BUF_NON_DIRTY,HOLDER_NON_DIRTY,READ =
30
WORKER,PAGE_VOLBITMAP,BUF_NON_DIRTY,HOLDER_NON_DIRTY,READ =
15
WORKER,PAGE_BTREE_R      ,BUF_NON_DIRTY,HOLDER_NON_DIRTY,READ
=          5
Time_data_page_lock_acquire_time:
Time_data_page_hold_acquire_time:
Time_data_page_fix_acquire_time:
WORKER,PAGE_HEAP      ,OLD_PAGE_IN_PB      ,READ ,UNCOND
=          27
WORKER,PAGE_VOLHEADER,OLD_PAGE_IN_PB      ,READ ,UNCOND
=          167
WORKER,PAGE_VOLBITMAP,OLD_PAGE_IN_PB      ,READ ,UNCOND
=          40
WORKER,PAGE_BTREE_R      ,OLD_PAGE_IN_PB      ,READ ,UNCOND
=          23
Num_mvcc_snapshot_ext:
DELETE      ,INS_VACUUMED      ,VISIBLE      =          5
SNAPSHOT,INS_VACUUMED      ,VISIBLE      =          5
Time_obj_lock_acquire_time:
Thread_stats_counters_timers:
Thread_pgbuf_daemon_stats_counters_timers:
Num_dwb_flushed_block_volumes:
Thread_loaddb_stats_counters_timers:

csql> ;.hist off

```

질의 수행 시간을 출력(;Time)

;Time 세션 명령어로 질의를 수행한 시간을 출력하도록 설정할 수 있다. **ON** 혹은 **OFF** 로 지정하며, 인자가 없으면 현재 설정값을 보여준다. 기본값은 **ON** 이다.

SELECT 질의에서는 페치(fetch)한 레코드를 출력하는 시간까지 포함한다. 따라서, **SELECT** 질의에서 모든 레코드의 출력이 한 번에 끝난 수행 시간을 보려면 CSQL 인터프리터 수행 시 **--no-pager** 옵션을 사용해야 한다.

```

$ csql -u dba --no-pager demodb
csql> ;time ON

```

```
csql> ;time
TIME IS ON
```

서버 저장 메시지 출력(;SERVER-output)

이 값을 ON으로 설정하면 서버의 DBMS_OUTPUT 버퍼에 저장된 메시지를 출력한다. 기본값은 OFF이다. DBMS_OUTPUT 버퍼는 PL/CSQL 저장 프로시저/함수에서 DBMS_OUTPUT.put_line()이나 DBMS_OUTPUT.put() 호출을 통해 쌓인 메시지들을 저장하고 있다. 실행한 SQL 문의 결과 출력 후에 '<DBMS_OUTPUT>' 표시 아래에 DBMS_OUTPUT 메시지들을 출력한다. 이 때, 저장 프로시저/함수 실행 도중에 오류가 발생하면 실행문 순서와 상관없이 예러 메시지를 포함한 CSQL의 기본 메시지들을 모두 출력한 다음 DBMS_OUTPUT 메시지를 출력하므로 주의가 필요하다. 예를 들어, 아래에서 DBMS_OUTPUT.put_line() 문은 ZERO_DIVIDE 예외를 일으키는 RETURN 문 이전에 실행되지만, 출력에서는 DBMS_OUTPUT 메시지가 예러 메시지보다 나중에 위치한다.

```
csql> ;server-output on
SERVER OUTPUT IS ON
csql>
csql> create or replace function late_message return integer as
csql> begin
csql>     DBMS_OUTPUT.put_line('Hello world');
csql>     return (1/0);    -- ZERO_DIVIDE exception
csql> end;
Execute OK. (0.024973 sec) Committed. (0.000000 sec)

1 command(s) successfully processed.
csql>
csql> select late_message() from dual;

In line 2, column 23,

ERROR: Stored procedure execute error:
      (line 4, column 13) division by zero

<DBMS_OUTPUT>
====
Hello world

0 command(s) successfully processed.
```

질의 결과를 칼럼 당 한 라인으로 출력(;LINE-output)

이 값을 ON으로 설정하면 질의 결과 레코드를 칼럼 당 한 라인으로 출력한다. 기본 설정은 OFF로서, 한 레코드는 한 라인으로 출력한다.

```
csql> ;line-output ON
csql> select * from athlete;
```

```
=== <Result of SELECT Command in Line 1> ===
```

```
<00001> code      : 10999
        name      : 'Fernandez Jesus'
        gender    : 'M'
        nation_code: 'ESP'
        event     : 'Handball'
<00002> code      : 10998
        name      : 'Fernandez Jaime'
        gender    : 'M'
        nation_code: 'AUS'
        event     : 'Rowing'
...

```

질의 수행 이력 확인(;HISTORYList)

앞서 수행된 명령어(입력 내용)를 수행 번호를 포함한 리스트로 보여준다.

```
csql> ;historylist
----< 1 >----
select * from nation;
----< 2 >----
select * from athlete;

```

지정된 수행 번호에 해당하는 입력 내용을 버퍼로 불러오기(;HISTORYRead)

;HISTORYRead 세션 명령어를 사용해 지정된 **;HISTORYList** 에서 확인한 수행 번호에 해당하는 내용을 명령어 버퍼로 불러올 수 있다. 해당 SQL 문을 직접 입력한 것과 같은 상태이므로 바로 **;run** 또는 **;xrun** 를 입력할 수 있다.

```
csql> ;historyread 1

```

기본 편집기를 호출(;EDIT)

지정된 편집기를 호출하는 세션 명령어이다. 기본 편집기는 Linux에서는 vi이고, Windows에서는 메모장이다. 다른 편집기로 지정하려면 **;EDITOR_Cmd** 명령어를 이용한다.

format / fmt

format 또는 **fmt** 옵션을 통해 SQL문을 포맷한 후 편집할 수 있다. 포맷터 등록은 **;Formatter_cmd** 세션 명령어를 이용한다.

```
csql> ;edit
SELECT * FROM t1

csql> ;edit format
SELECT

```

```
*
FROM
t1
```

포맷터 설정(;Formatter_cmd)

;EDIT 세션 명령어에서 옵션으로 사용될 포맷터를 지정한다. 예제와 같이 사용자가 설정하여 사용할 수 있고, OS 환경변수(CUBRID_CSQ_L_FORMATTER 또는 FORMATTER, 두 환경변수가 설정된 경우 CUBRID_CSQ_L_FORMATTER를 사용)를 미리 등록하여 사용할 수 있다.

```
csql> ;formatter_cmd /usr/local/bin/fsqf

$ export CUBRID_CSQ_L_FORMATTER=/home/cubrid/bin/fsqf
또는
$ export FORMATTER=/home/cubrid/bin/fsqf
```

참고

Free SQL Formatter 사용을 권고합니다.

다운로드 URL: <https://github.com/CUBRID/fsqf/releases>

편집기 설정(;EDITOR_Cmd)

;EDIT 세션 명령어에서 사용될 편집기를 지정한다. 예제와 같이 기본 편집기인 vi 대신에 해당 시스템에 설치된 다른 편집기(예: emacs)를 설정할 수 있다.

```
csql> ;editor_cmd emacs
csql> ;edit
```

OS 환경변수(CUBRID_CSQ_L_EDITOR 또는 EDITOR, 두 환경변수가 설정된 경우 CUBRID_CSQ_L_EDITOR를 사용)를 미리 등록하여 사용할 수도 있다.

```
export CUBRID_CSQ_L_EDITOR=emacs
또는
export EDITOR=emacs
```

싱글 라인 모드 설정(;Singleline)

싱글 라인 모드를 **ON** 또는 **OFF** 로 설정하는 명령어이다 (기본값 **ON**). 싱글 라인 모드 ON에서는 세미 콜론(:)과 ENTER를 입력하면 SQL문이 바로 실행된다. 싱글 라인 모드 OFF에서는 여러 개의 SQL 문을

저장해 두었다가 ;xr 혹은 ;ru 세션 명령어로 한꺼번에 수행한다. 만약, ON 또는 OFF 를 지정하지 않으면 현재 설정된 값을 보여준다.

```
csql> ;singleline off
SINGLE IS OFF
csql> ;singleline
SINGLE IS OFF
```

csql 세션 전환(;Connect)

CSQL을 종료하지 않고 다른 사용자로 접속을 전환하기 위한 세션 명령어이다. 명령어 형식은 다음과 같다.

- user: 접속할 사용자 이름
- database: 접속할 데이터베이스 이름 (생략 시, 현재 접속한 데이터베이스)
- host: 접속할 호스트 이름 (생략 시, localhost)

```
csql> ;connect public
csql> ;connect dba

csql> ;connect public testdb
csql> ;connect dba demodb

csql> ;connect Peter testdb@192.168.0.1
csql> ;connect public demodb@localhost
```

⚠ 경고

1. ‘;connect’ 세션 명령어를 실행하면 세션 전환 성공 여부와 관계 없이 현재의 CSQL 세션 연결은 해제된다.
2. CSQL이 시스템 관리자 모드인 경우 ‘;connect’ 세션 명령어는 사용할 수 없다 (csql -u dba --sysadm demodb 형태의 명령으로 CSQL 수행한 경우).

CUBRID SQL

이 장에서는 CUBRID에서 사용되는 데이터 타입, 함수와 연산자, 데이터 조회나 테이블 조작 등의 SQL 구문에 대해 설명한다. 인덱스나 트리거, 분할, 시리얼 및 사용자 정보 변경 등의 작업을 위한 SQL 구문도 찾아볼 수 있다.

주요 내용은 다음과 같다.

- 작성 규칙
 - 식별자: 테이블, 인덱스, 칼럼의 이름으로 허용되는 문자열인 식별자 작성 방법을 설명한다.
 - 예약어: 질의문을 구성하기 위한 문법적인 용도로 사용하는 예약어들을 나열한다. 예약어를 식별자로 사용하려면 식별자를 겹따옴표(""), 백틱(`) 혹은 대괄호([])로 반드시 감싸야 한다.
 - 주석
 - 리터럴: 상수 값 작성 방법을 설명한다.
- 데이터 타입: 데이터를 저장하는 형태인 데이터 타입에 대해 설명한다.
- 데이터 정의문: 테이블, 인덱스, 뷰, 시리얼을 생성, 변경, 삭제하는 방법을 설명한다.
- 연산자와 함수: 질의문에서 사용되는 연산자와 함수에 대해 설명한다.
- 데이터 조작성문: SELECT, INSERT, UPDATE, DELETE 구문에 대해 설명한다.
- 질의 최적화: 인덱스와 힌트, 인덱스 힌트 구문을 이용한 질의 최적화에 대해 설명한다.
- 병렬 질의 처리: 대량 데이터 처리를 위한 병렬 실행 기능에 대해 설명한다.
- 분할: 하나의 테이블을 여러 독립적인 논리적 단위로 분할하는 방법을 설명한다.
- 트리거(trigger): 특정 질의 수행 시 특정 기능이 같이 수행되도록 하는 트리거의 생성, 변경, 삭제 방법을 설명한다.
- 클래스 상속: 부모와 자식 테이블(클래스) 사이에 속성을 상속하는 방법을 설명한다.
- 데이터베이스 관리: 사용자 관리, SET, SHOW 문에 대해 설명한다.
- 사용자 스키마: 사용자와 함께 생성되는 데이터베이스의 논리적인 객체 모음을 설명한다.
- 시스템 카탈로그: CUBRID 데이터베이스의 내부 정보인 시스템 카탈로그에 대해 설명한다.
- Information Schema: 데이터베이스 메타데이터를 이식성 있게 노출하는 SQL 표준 INFORMATION_SCHEMA 뷰에 대해 설명한다.

작성 규칙

식별자

식별자 작성 원칙

테이블 이름, 인덱스 이름, 뷰 이름, 칼럼 이름, 사용자 이름 등이 식별자에 해당한다. 식별자는 다음의 원칙에 따라 작성해야 한다.

- 반드시 문자로 시작되어야 한다. 즉, 숫자나 기호로 시작할 수 없다.
- 대소문자를 구별하지 않는다.
- **예약어**는 허용되지 않는다.

```

identifier ::= identifier_letter [ { other_identifier } ; ]
identifier_letter ::= upper_case_letter | lower_case_letter
other_identifier ::= identifier_letter | digit | _ | #
digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
upper_case_letter ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
lower_case_letter ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
  
```

허용되는 식별자

문자로 시작되는 식별자

식별자의 첫 글자에는 반드시 문자를 사용해야 하며, 연산자로 사용되는 특수 문자를 제외한 나머지 특수 문자를 포함할 수 있다. 다음은 허용되는 식별자의 예제이다.

```

a
a_b
ssn#
this_is_an_example_#
  
```

큰따옴표, 대괄호, 백틱 부호로 둘러싸인 식별자

허용되지 않는 식별자 또는 예약어에 해당하더라도, 큰따옴표(" ")나 대괄호([]) 또는 백틱 부호(`)로 식별자를 둘러싸면 예외적으로 허용된다. 큰따옴표는 질의 관련 파라미터인 **ansi_quotes**가 yes일 때에만 식별자를 감싸는 부호로 사용할 수 있고, 이 값이 no이면 문자열을 감싸는 부호로 사용한다. **ansi_quotes**의 기본값은 yes이다. 다음은 허용되는 식별자의 예제이다.

```

"select"
"@lowcost"
"low cost"
"abc" "def"
[position]
  
```

⚠ 경고

큰따옴표, 대괄호, 백틱 부호로 둘러싸인 식별자라도 대괄호([])나 점(.)을 포함할 수 없다.

- 대괄호는 내부에서 쿼리를 파싱할 때 예약어 처리를 위해 기본으로 사용한다.
- 점은 Path Expression 또는 JSON에서 사용되며, 스키마 이름과 객체 이름을 구분하는데에도 사용된다.

다음은 큰따옴표(" ")로 둘러싸여 있지만 허용되지 않는 식별자이다.

```
"he[o"
"wor]d"
"hello[]world"
"hello.world"
```

i 참고

10.0 버전부터는 예약어로 된 칼럼 이름이 “테이블 이름 (또는 별칭)”과 같이 사용되는 경우, 칼럼 이름을 큰따옴표로 감싸지 않아도 된다.

```
CREATE TABLE tbl ("int" int, "double" double);

SELECT t.int FROM tbl t;
```

위의 SELECT 질의문에서 “int”가 “t.”과 함께 쓰인 칼럼 이름이다.

허용되지 않는 식별자

특수 문자나 숫자로 시작되는 식별자

다음과 같이 언더바(_), #을 제외한 특수 문자, 숫자로 시작되는 식별자는 허용되지 않는다.

```
_a
#ack
%nums
2fer
88abs
```

공백을 포함하는 식별자

다음과 같이 중간에 공백을 포함하는 식별자는 허용되지 않는다.

```
col1 t1
```

연산자로 사용하는 특수 문자를 포함하는 식별자

다음과 같이 연산자로 사용되는 특수 문자(+, -, *, /, %, ||, !, <, >, =, |, ^, &, ~)를 포함하는 식별자는 허용되지 않는다.

```
col+
col~
col&&
```

식별자 이름의 최대 길이

다음은 각 식별자 이름으로 허용되는 최대 바이트 길이를 정리한 표이다. 단위는 바이트 길이이며, 사용하는 문자셋에 따라 문자 수와 바이트 길이는 다를 수 있음에 주의한다. (예를 들어, UTF-8 문자 셋에서 한글 한 글자는 3바이트의 바이트 길이이다.)

식별자	최대 바이트 길이
Database	17
User	31
Table	222
Column	254
Index	254
Constraint	254
Stored Procedure	222
Trigger	222

식별자	최대 바이트 길이
View	222
Serial	222

참고

11.2 버전 전까지는 테이블, 트리거, 시리얼 이름의 최대 길이가 254바이트였다. 11.2 버전부터 테이블, 트리거, 시리얼 및 저장 프로시저 이름에는 스키마 이름이 포함된다. 따라서 식별자 이름은 소유자 이름의 길이인 31바이트와 구분자(.)의 길이인 1바이트를 제외하고 222바이트를 넘을 수 없다.

참고

기본 키(pk_<table_name>_<column_name>), 외래 키(fk_<table_name>_<column_name>)의 이름 등 자동으로 생성되는 제약조건(constraint) 이름도 식별자의 최대 길이인 254바이트를 넘을 수 없다.

예약어

아래는 CUBRID 키워드(keywords) 중 명령어, 함수명, 타입명 등으로 예약되어 있는 예약어(reserved words)를 정리한 표이다. 사용자는 테이블 이름, 칼럼 이름, 변수 이름과 같은 식별자(identifier)로 아래에 정리된 예약어를 사용할 수 없다. 단, 큰따옴표(" ")나 대괄호([]) 또는 백틱 부호(`)로 감싸는 방법으로 예약어를 식별자로 사용할 수 있다.

ABSOLUTE	ACTION	ACCESS
ADD	ADD_MONTHS	AFTER
ALL	ALLOCATE	ALTER

AND

ANY

ARE

AS

ASC

ASSERTION

AT

ATTACH

ATTRIBUTE

AVG

BEFORE

BETWEEN

BIGINT

BINARY

BIT

BIT_LENGTH

BLOB

BOOLEAN

BOTH

BREADTH

BY

CALL

CASCADE

CASCADED

CASE

CAST

CATALOG

CHANGE

CHAR

CHARACTER

CHECK

CLASS

CLASSES

CLOB

COALESCE

COLLATE

COLLATION

COLUMN

COMMIT

CONNECT

CONNECT_BY_ISCYCLE

CONNECT_BY_ISLEAF

CONNECT_BY_ROOT

CONNECTION

CONSTRAINT

CONSTRAINTS

CONTINUE

CONVERT

CORRESPONDING

COUNT

CREATE

CROSS

CURRENT

CURRENT_DATE

CURRENT_DATETIME

CURRENT_TIME

CURRENT_TIMESTAMP

CURRENT_USER

CURSOR

CYCLE

DATA

DATA_TYPE

DATABASE

DATE

DATETIME

DAY

DAY_HOUR

DAY_MILLISECOND

DAY_MINUTE

DAY_SECOND

DEALLOCATE

DEC

DECIMAL

DECLARE

DEFAULT

DEFERRABLE

DEFERRED

DELETE

DEPTH

DESC

DESCRIBE

DESCRIPTOR

DIAGNOSTICS

DIFFERENCE

DISCONNECT

DISTINCT

DISTINCTROW

DIV

DO

DOMAIN

DOUBLE

DUPLICATE

DROP

EACH

ELSE

ELSEIF

END

EQUALS

ESCAPE

EVALUATE

EXCEPT

EXCEPTION

EXEC

EXECUTE

EXISTS

EXTERNAL

EXTRACT

FALSE

FETCH

FILE

FIRST

FLOAT

FOR

FOREIGN

FOUND

FROM

FULL

FUNCTION

GENERAL

GET

GLOBAL

GO

GOTO

GRANT

GROUP

HAVING

HOUR

HOUR_MILLISECOND

HOUR_MINUTE

HOUR_SECOND

IDENTITY

IF

IGNORE

IMMEDIATE

IN

INDEX

INDICATOR

INHERIT

INITIALLY

INNER

INOUT

INPUT

INSERT

INT

INTEGER

INTERSECT

INTERSECTION

INTERVAL

INTO

IS

ISOLATION

JOIN

KEY

LANGUAGE

LAST

LEADING

LEAVE

LEFT

LESS

LEVEL

LIKE

LIMIT

LIST

LOCAL

LOCAL_TRANSACTION_ID

LOCALTIME

LOCALTIMESTAMP

LOOP

LOWER

MATCH

MAX

METHOD

MILLISECOND

MIN

MINUTE

MINUTE_MILLISECOND

MINUTE_SECOND

MOD

MODIFY

MODULE

MONTH

MULTISET

MULTISET_OF

NA

NAMES

NATIONAL

NATURAL

NCHAR

NEXT

NO

NONE

NOT

NULL

NULLIF

NUMERIC

OBJECT

OCTET_LENGTH

OF

OFF

ON

ONLY

OPTIMIZATION

OPTION

OR

ORDER

OUT

OUTER

OUTPUT

OVERLAPS

PARAMETERS

PARTIAL

POSITION

PRECISION

PREPARE

PRESERVE

PRIMARY

PRIOR

PRIVILEGES

PROCEDURE

QUERY

READ

REAL

RECURSIVE

REF

REFERENCES

REFERENCING

RELATIVE

RENAME

REPLACE

RESIGNAL

RESTRICT

RETURN

RETURNS

REVOKE

RIGHT

ROLE

ROLLBACK

ROLLUP

ROUTINE

ROW

ROWNUM

ROWS

SAVEPOINT

SCHEMA

SCOPE

SCROLL

SEARCH

SECOND

SECOND_MILLISECOND

SECTION

SELECT

SENSITIVE

SEQUENCE

SEQUENCE_OF

SERIALIZABLE

SESSION

SESSION_USER

SET

SET_OF

SETEQ

SHARED

SIBLINGS

SIGNAL

SIMILAR

SIZE

SMALLINT

SOME

SQL

SQLCODE

SQLERROR

SQLEXCEPTION

SQLSTATE

SQLWARNING

STATISTICS

STRING

SUBCLASS

SUBSET

SUBSETEQ

SUBSTRING

SUM

SUPERCLASS

SUPERSET

SUPERSETEQ

SYS_CONNECT_BY_PATH

SYS_DATE

SYS_DATETIME

SYS_TIME

SYS_TIMESTAMP

SYSDATE

SYSDATETIME

SYSTEM_USER

SYSTIME

TABLE

TEMPORARY

THEN

TIME

TIMESTAMP

TIMEZONE_HOUR

TIMEZONE_MINUTE

TO

TRAILING

TRANSACTION

TRANSLATE

TRANSLATION

TRIGGER

TRIM

TRUE

TRUNCATE

UNDER

UNION

UNIQUE

UNKNOWN

UPDATE

UPPER

USAGE

USE

USER

USING

UTIME

VALUE

VALUES

VARCHAR

VARIABLE

VARYING

VCLASS

VIEW

WHEN

WHENEVER

WHERE

WHILE

WITH

WITHOUT

WORK

WRITE

XOR

YEAR

YEAR_MONTH

ZONE

주석

주석은 '--'로 시작하여 해당 라인 전부가 적용되는 SQL 방식과 '//'로 시작하여 해당 라인 전부가 주석으로 간주되는 C++ 언어 방식, '/*'로 시작하여 '*/'로 끝나는 C 언어 방식과 같이 세 가지 형태로 사용할 수 있다.

다음은 주석의 예이다.

- -- 사용

```
-- 이것은 SQL 방식의 주석이다.
```

- // 사용

```
// 이것은 C++ 방식의 주석이다.
```

- /* */ 사용

```
/* 이것은 C 방식의 주석이다.*/

/* 이것은 C 방식을 이용하여
두개의 라인이 주석으로 적용되었다 */
```

리터럴

CUBRID에서 리터럴(literal) 값을 작성하는 방법을 기술한다.

숫자

숫자 값에는 정확한 수치(exact value)를 표기하는 방법과 근사치(approximate value)를 표기하는 방법이 있다.

- 정확한 수치는 일련의 숫자와 .으로 표현하며, 값의 범위에 따라 INT, BIGINT 또는 NUMERIC 타입으로 해석된다.

```
10, 123456789012, 1234234324234.23
```

- 근사치는 일련의 숫자와 . 그리고 E(공학용 표기. 10의 승수)로 표현하며, DOUBLE 타입으로 해석된다.

```
1.2345E15, 12345E5
```

- +, -를 숫자 앞에 표기할 수 있으며, 승수를 표현하는 E 뒤에 나오는 숫자 앞에도 +, -를 표기할 수 있다.

```
+10.2345, -1.2345E-15
```

날짜/시간

날짜와 시간을 나타내는 타입에는 DATE, TIME, DATETIME, TIMESTAMP 타입이 있으며, 문자열 앞에 date, time, datetime, timestamp 리터럴(대소문자 구분 없음)을 추가하여 이 값들을 표기한다.

날짜/시간 리터럴을 사용하면 문자열을 날짜/시간으로 변환하는 `TO_DATE()`, `TO_TIME()`, `TO_DATETIME()`, `TO_TIMESTAMP()` 와 같은 함수를 사용하지 않아도 된다. 단, 날짜와 시간을 표현하는 문자열의 순서는 반드시 지켜야 한다.

- 날짜 리터럴은 'YYYY-MM-DD' 또는 'MM/DD/YYYY'만 허용한다.

```
date'1974-12-31', date'12/31/1974'
```

- 시간 리터럴은 'HH:MI:SS', 'HH:MI:SS AM', 'HH:MI:SS PM'만 허용한다.

```
time'12:13:25', time'12:13:25 AM', time'12:13:25 PM'
```

- DATETIME 타입에서 사용하는 날짜/시간 리터럴은 'YYYY-MM-DD HH:MI:SS[.msec AM|PM]' 또는 'MM/DD/YYYY HH:MI:SS[.msec AM|PM]' 형식을 허용한다. msec은 밀리 초로, 3자리 숫자까지 표기한다.

```
datetime'1974-12-31 12:13:25.123 AM', datetime'12/31/1974  
12:13:25.123 AM'
```

- TIMESTAMP 타입에서 사용하는 날짜/시간 리터럴은 'YYYY-MM-DD HH:MI:SS[AM|PM]' 또는 'MM/DD/YYYY HH:MI:SS[AM|PM]' 형식을 허용한다.

```
timestamp'1974-12-31 12:13:25 AM', timestamp'12/31/1974 12:13:25 AM'
```

- 타임존을 포함하는 날짜/시간 타입의 리터럴은 위에서 설명한 것과 동일한 형식을 가지며, 뒤에 타임존 정보를 나타내는 오프셋 또는 지역 이름을 추가한다.

- 각 타입의 값을 나타내기 위해 문자열 앞에 datetimetz, datetimeltz, timestamptz 또는 timestamppltz 문자를 추가한다.

```
datetimetz'10/15/1986 5:45:15.135 am +02:30:20';  
datetimetz'10/15/1986 5:45:15.135 am +02:30';  
datetimetz'10/15/1986 5:45:15.135 am +02';  
datetimeltz'10/15/1986 5:45:15.135 am Europe/Bucharest'  
datetimetz'2001-10-11 02:03:04 AM Europe/Bucharest EEST';  
timestamppltz'10/15/1986 5:45:15 am Europe/Bucharest'  
timestamppltz'10/15/1986 5:45:15 am Europe/Bucharest'
```

- 문자열 앞에 오는 리터럴은 <날짜/시간 타입> WITH TIMEZONE 또는 <날짜/시간 타입> WITH LOCAL TIME ZONE으로 대체할 수 있다.

```
DATETIME WITH TIMEZONE = datetimetz
DATETIME WITH LOCAL TIMEZONE = datetimeltz
TIMESTAMP WITH TIMEZONE = timestamptz
TIMESTAMP WITH LOCAL TIMEZONE = timestamppltz
```

```
DATETIME WITH TIME ZONE '10/15/1986 5:45:15.135 am +02';
DATETIME WITH LOCAL TIME ZONE '10/15/1986 5:45:15.135 am +02';
```

참고

- <date/time type> WITH LOCAL TIME ZONE: 내부적으로 UTC 시간을 저장하며, 출력 시 로컬 (현재의 세션) 타임존으로 변환된다.
- <date/time type> WITH TIME ZONE: 내부적으로 UTC 시간과 생성 시 타임존 정보(사용자가 명시하거나 세션 타임존에 의해 결정됨)를 저장한다.

비트열

비트열은 2진수 형식과 16진수 형식의 2가지를 사용한다.

2진수 형식은 숫자 앞에 B 또는 0b를 붙여 표기한다. B 뒤에는 0과 1로 된 문자열을, 0b 뒤에는 0과 1로 된 숫자를 입력한다.

```
B'10100000'
0b10100000
```

2진수는 8자리씩 표기하며, 입력 숫자가 8자리로 나뉘지 않으면 뒤에 0이 8자리를 채울 때까지 덧붙은 값으로 저장된다. 즉, B'1'은 B'10000000'으로 저장된다.

16진수 형식은 숫자 앞에 X 또는 0x를 붙여 표기한다. X 뒤에는 16진수 문자열을, 0x 뒤에는 16진수 숫자를 입력한다.

```
X'a0'
0xA0
```

16진수는 2자리씩 표기하며, 입력 숫자가 2자리로 나뉘지 않으면 뒤에 0이 2자리를 채울 때까지 덧붙은 값으로 저장된다. 즉, X'a'는 X'a0'으로 저장된다.

문자열

문자열은 작은 따옴표로 감싸서 표현한다.

- 작은 따옴표를 문자열에 포함시키고 싶으면 연속해서 두 번 입력한다.

```
SELECT 'You''re welcome.';
```

- 백슬래시를 이용한 이스케이프는 **cubrid.conf**의 **no_backslash_escapes** 파라미터 값을 no로 설정했을 때만 사용할 수 있다. 기본값은 yes이다.

보다 자세한 설명은 [특수 문자 이스케이프](#)를 참고한다.

- 문자열 앞에 문자셋 소개자를 두고 문자열 뒤에는 COLLATE 수정자가 올 수 있다.

보다 자세한 설명은 [문자셋 소개자](#)를 참고한다.

컬렉션

컬렉션 타입에는 SET, MULTiset, LIST가 있으며, 쉼표로 구분되는 원소들을 중괄호({, })로 감싸서 표현한다.

```
{'c','c','c','b','b','a'}
```

보다 자세한 설명은 [컬렉션 데이터 타입](#)을 참고한다.

NULL

NULL 값은 데이터가 없다는 것을 의미한다. NULL은 대소문자를 구분하지 않아 null로도 쓰일 수 있다. NULL 값은 숫자 0 또는 빈 문자열("")이 아니라는 점에 주의한다.

데이터 타입

데이터 타입

수치형 데이터 타입

정수 또는 실수를 저장하기 위해 다음의 수치형 데이터 타입을 지원한다.

타입	Bytes	최소값	최대값	정확/근사치
SHORT, SMALLINT	2	-32,768	32,767	정확한 수치
INTEGER, INT	4	-2,147,483,648	+2,147,483,647	정확한 수치
BIGINT	8	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807	정확한 수치
NUMERIC, DECIMAL	16	정밀도 p : 1 스케일 s : 0	정밀도 p : 38 스케일 s : 38	정확한 수치
FLOAT, REAL	4	-3.402823466E+38 (ANSI/IEEE 754-1985 표준)	+3.402823466E+38 (ANSI/IEEE 754-1985 표준)	근사치 부동소수 점: 7자리
DOUBLE, DOUBLE PRECISION	8	-1.7976931348623157E+308 (ANSI/IEEE 754-1985 표준)	+1.7976931348623157E+308 (ANSI/IEEE 754-1985 표준)	근사치 부동소수 점: 15자리

수치형 데이터 타입은 정확한(exact) 타입과 근사치(approximate) 타입으로 구분된다. 정확한 수치형 데이터 타입(**SMALLINT, INT, BIGINT, NUMERIC**)은 정확하고 일관된 값을 가져야 하는 경우에 사용된다. 근사치 수치형 데이터 타입(**FLOAT, DOUBLE**)은 리터럴 값이 같아도 시스템에 따라 다르게 해석될 수 있으므로 주의한다.

CUBRID는 수치형 데이터 타입에 대해 UNSIGNED 타입을 지원하지 않는다.

위의 표에서 두 가지 이름으로 표기한 타입들은 동일하지만, 테이블 생성 후 **SHOW COLUMNS** 문으로 타입의 이름을 확인할 때에는 항상 위 단어로 표기된다. 예를 들어, 테이블을 생성할 때에는 **SHORT, SMALLINT** 둘 다 쓸 수 있으며, **SHOW COLUMNS** 문으로 타입의 이름을 확인할 때에는 항상 **SHORT** 로 표기된다.

정밀도와 스케일(Precision and Scaling)

숫자 데이터 타입의 정밀도(precision)는 그 값이 유지할 수 있는 유효한 자릿수로 정의된다. 이는 정확한 수치형이든 근사치 수치형이든 마찬가지이다.

스케일(scale)은 소수점 이하의 자릿수를 나타내는데, 정확한 수치형에서만 의미가 있다. 정확한 수치형으로 선언된 속성은 항상 고정된 정밀도와 스케일을 갖게 된다. **NUMERIC** (또는 **DECIMAL**) 데이터 타입은 항상 최소한 한 자리의 정밀도를 갖는다. 스케일의 범위는 0과 선언된 정밀도 사이여야 한다. 스케일이 정밀도보다 클 수는 없다. **INTEGER** 나 **SMALLINT**, **BIGINT** 데이터 타입에서는 스케일은 0이고(즉, 소수점 이하가 없음), 정밀도는 시스템에 의해 고정된다.

수치형 리터럴(Numeric Literals)

수치형 값을 입력하기 위해서는 특별한 기호가 사용될 수 있는데, 플러스(+)는 양수를, 마이너스(-)는 음수를 나타내는 데 사용한다. 과학용 표기법이 사용될 수도 있다. 화폐 값을 표현하기 위하여 시스템에 지정된 통화 기호를 사용할 수도 있다. 수치형 리터럴로 표현 가능한 최대 정밀도는 255이다.

수치형 변환(Numeric Coercions)

모든 수치형 데이터 타입은 상호 비교 가능하고, 이를 위해 서로 공통된 수치형 데이터 타입으로 자동 변환이 이루어진다. 명시적인 변환은 **CAST** 연산자를 이용해야 한다. 서로 다른 수치형 데이터가 서로 정렬되거나, 수식에서 계산될 때에는 시스템에 의하여 자동으로 변환된다. 예를 들어, **FLOAT** 타입의 속성 값과 **INTEGER** 타입의 속성 값을 더하게 되면, 시스템이 자동적으로 **INTEGER** 속성 값을 가장 근사한 **FLOAT** 값으로 변환한 후 덧셈을 수행한다.

다음은 **FLOAT** 타입 값과 **INTEGER** 타입 값을 더하는 경우 **FLOAT** 타입 값을 출력하는 예이다.

```
CREATE TABLE tbl (a INT, b FLOAT);
INSERT INTO tbl VALUES (10, 5.5);
SELECT a + b FROM tbl;
```

```
1.550000e+01
```

다음은 두 개의 정수 값을 더하는 경우 결과 값도 **INTEGER** 타입이 되기 때문에 오버플로우(overflow) 에러가 발생하는 예이다.

```
SELECT 100000000*1000000;
```

```
ERROR: Data overflow on data type integer.
```

위와 같은 경우 어느 한 쪽의 타입을 **BIGINT**로 명시하면, **BIGINT**로 결과 값을 결정하고, 정상적인 결과를 출력한다.

```
SELECT CAST(100000000 AS BIGINT)*1000000;
```

```
10000000000000000
```

⚠ 경고

CUBRID 2008 R2.0 미만 버전에서는 입력된 상수가 **INTEGER** 범위를 넘어서면 **NUMERIC**으로 처리되었으나, CUBRID 2008 R2.0 이상 버전에서는 **BIGINT**로 처리된다.

INT, INTEGER

INTEGER 데이터 타입은 정수 표현을 위해 사용하며, 표현할 수 있는 값의 범위는 -2,147,483,648에서 +2,147,483,647이다. 작은 정수를 표현하기 위해 **SMALLINT**를 사용하거나, 큰 정수를 표현하기 위해 **BIGINT**를 사용할 수 있다.

- **INTEGER**와 **INT**는 같은 의미로 사용된다.
- **INT** 타입에 실수가 입력되면, 소수점 아래 숫자가 반올림되어 정수값이 저장된다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

INTEGER에 8934를 지정하면 8934가 저장됨.
INTEGER에 7823467를 지정하면 7823467이 저장됨.
INTEGER에 89.8를 지정하면 90이 저장됨(소수점 뒤의 수치는 반올림됨).
INTEGER에 3458901122를 지정하면 오류가 발생함(표현 가능 범위를 초과하면 오류 발생).

SHORT, SMALLINT

SMALLINT 데이터 타입은 작은 정수 표현을 위해 사용되며, 표현할 수 있는 값의 범위는 -32,768에서 +32,767이다.

- **SMALLINT**와 **SHORT**는 같은 의미로 사용된다.
- **SMALLINT** 타입에 실수가 입력되면, 소수점 아래 숫자가 반올림되어 정수값이 저장된다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

SMALLINT에 8934를 지정하면 8934가 저장됨.
SMALLINT에 34.5를 지정하면 35가 저장됨(소수점 이하의 숫자는 반올림됨).

SMALLINT에 23467를 지정하면 23467이 저장됨.
 SMALLINT에 89354를 지정하면 오류가 발생함(표현 가능 범위를 초과하면 오류 발생).

BIGINT

BIGINT 데이터 타입은 큰 정수 표현을 위해 사용되며, 표현할 수 있는 값의 범위는 -9,223,372,036,854,775,808에서 9,223,372,036,854,775,807이다.

- **BIGINT** 타입에 실수가 입력되면, 소수점 아래 숫자가 반올림되어 정수값이 저장된다.
- 정밀도와 표현할 수 있는 범위를 기준으로는 다음과 같이 정렬할 수 있다.

SMALLINT $\subset$ **INTEGER** $\subset$ **BIGINT** $\subset$ **NUMERIC**

- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

BIGINT에 8934를 지정하면 8934가 저장됨.
 BIGINT에 89.1을 지정하면 89가 저장됨.
 BIGINT에 89.8을 지정하면 90이 저장됨(소수점 뒤의 수치는 반올림됨).
 BIGINT에 3458901122를 지정하면 3458901122가 저장됨.

NUMERIC, DECIMAL

NUMERIC 또는 **DECIMAL** 데이터 타입은 고정 소수점 숫자를 표현하기 위해 사용되며, 다음과 같이 전체 자리 수(정밀도)와 소수점 아래 자리 수(스케일)를 옵션으로 지정하여 정의할 수 있다. 정밀도 p 의 최소값은 1이고 최대값은 38이며, 정밀도 p 가 생략되면 기본값은 15이므로, 정수부가 15자리를 초과하는 데이터를 입력할 수 없다. 또한, 스케일 s 가 생략되면 스케일의 기본값은 0이므로 소수점 아래 첫째 자리에서 반올림한 정수를 반환한다.

NUMERIC [(p[, s])]

- 정밀도는 반드시 스케일 이상이어야 한다.
- 정밀도는 (데이터의 정수부 자리 수 + 스케일) 이상이 되도록 지정한다.
- **NUMERIC**과 **DECIMAL**, 그리고 **DEC**는 같은 의미로 사용된다.
- **NUMERIC** 타입끼리 연산한 결과 값의 정밀도와 스케일이 어떻게 달라지는지 확인하려면 **수치형 데이터 타입의 산술 연산과 타입 변환**을 참고한다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

NUMERIC에 12345.6789를 지정하면 12346이 저장됨(스케일 기본값은 0이므로 소수점 아래 첫째 자리에서 반올림함).

NUMERIC(4)에 12345.6789를 지정하면 오류가 발생함(정밀도는 데이터의 정수부 자릿수 이상이어야 함).
 NUMERIC(3,4)를 선언하면 오류가 발생함(정밀도는 스케일 이상이어야 함).
 NUMERIC(4,4)에 0.123456789를 지정하면 .1235가 저장됨(소수점 아래 다섯째 자리에서 반올림함).
 NUMERIC(4,4)에 -0.123456789를 지정하면 -.1235가 저장됨(소수점 아래 다섯째 자리에서 반올림한 후, - 부호를 붙임).

FLOAT, REAL

FLOAT (또는 **REAL**) 데이터 타입은 부동 소수점 숫자를 표현하기 위해 사용된다.

정규 값(normalized value)으로 표현할 수 있는 값의 범위는 -3.402823466E+38 에서 -1.175494351E-38, 0, 그리고 +1.175494351E-38 에서 +3.402823466E+38이며, 이 범위를 벗어나서 0에 가까운 값은 비정규 값(denormalized value)으로 표현한다. 이는 ANSI/IEEE 754-1985 표준을 준수한다.

정밀도 p 의 최소값은 1이고 최대값은 38이며, 정밀도 p 가 생략되거나 7 이하로 지정되면 단일 정밀도(single-precision, 7자리의 유효 숫자)로 표현된다. 만약 정밀도 p 가 7보다 크고 38 이하이면 이중 정밀도(double-precision, 15자리의 유효 숫자)로 표현되며, **DOUBLE** 데이터 타입으로 변환된다.

FLOAT 데이터 타입은 7자리의 유효 자릿수를 넘는 입력 값에 대해 근사치를 저장하는 타입이므로 유효 자릿수를 넘어서는 정확한 값을 저장하려면 사용하지 않도록 주의한다.

FLOAT[(p)]

- **FLOAT** 타입의 유효 자리 수는 7이다.
- **FLOAT** 타입은 근사치 데이터를 저장하므로 데이터 비교 시 주의해야 한다.
- **FLOAT**와 **REAL**은 같은 의미로 사용된다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

FLOAT에 16777217을 입력하면 16777216이 저장되고, 1.677722e+07이 출력된다(정밀도가 생략되면, 7개의 유효 숫자로 표현하므로 8번째 숫자를 반올림함).
 FLOAT(5)에 16777217을 입력하면 16777216이 저장되고, 1.677722e+07이 출력된다(정밀도가 7 이하이면, 7개의 유효 숫자로 표현하므로 8번째 숫자를 반올림함).
 FLOAT(5)에 16777.217을 입력하면 16777.216이 저장되고, 1.677722e+04가 출력된다(정밀도가 7 이하이면, 7개의 유효 숫자로 표현하므로 8번째 숫자를 반올림함).
 FLOAT(10)에 16777.217을 지정하면 16777.217이 저장되고, 1.6777217000000000e+04가 출력된다(정밀도가 7보다 크고 38 이하이면, DOUBLE 타입으로 변환되어 15개의 유효 숫자로 표현하므로 0을 채움).

DOUBLE, DOUBLE PRECISION

DOUBLE 데이터 타입은 부동 소수점 숫자를 표현하기 위해 사용된다.

정규 값(normalized value)으로 표현할 수 있는 값의 범위는 -1.7976931348623157E+308에서 -2.2250738585072014E-308, 0, 그리고 2.2250738585072014E-308에서 1.7976931348623157E+308이며, 이 범위를 벗어나서 0에 가까운 값은 비정규 값(denormalized value)으로 표현한다. 이는 ANSI/IEEE 754-1985 표준을 준수한다.

정밀도를 지정할 수 없으며, 이 타입이 지정된 데이터는 이중 정밀도(double-precision, 15자리의 유효 숫자)로 표현된다.

DOUBLE 데이터 타입은 15자리의 유효 자릿수를 넘는 입력 값에 대해 근사치를 저장하는 타입이므로 유효 자릿수를 넘어서는 정확한 값을 지정할 때에는 사용하지 않도록 주의한다.

- **DOUBLE**의 유효 자리 수는 15자리이다.
- **DOUBLE** 타입은 근사치 데이터를 저장하므로 데이터 비교 시 주의해야 한다.
- **DOUBLE**과 **DOUBLE PRECISION**은 같은 의미로 사용된다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

DOUBLE에 1234.56789를 입력하면 1234.56789가 저장되고, 1.234567890000000e+03이 출력된다.
DOUBLE에 9007199254740993을 입력하면 9007199254740992가 저장되고, 9.007199254740992e+15가 출력된다.

참고

MONETARY 타입은 제거될 예정이며(deprecated), 더 이상 사용을 권장하지 않는다.

날짜/시간 데이터 타입

날짜/시간 데이터 타입은 날짜, 시간 혹은 이 두 가지를 모두 표현할 때 사용하는 데이터 타입으로 다음과 같은 데이터 타입을 지원한다.

타입	bytes	최소값	최대값	비고
DATE	4	0001년 1월 1일	9999년 12월 31일	예외적으로 DATE

타입	bytes	최소값	최대값	비고
				'0000-00-00'을 입력할 수 있다.
TIME	4	00시 00분 00초	23시 59분 59초	
TIMESTAMP	4	1970년 1월 1일 0시 0분 1초(GMT) 1970년 1월 1일 9시 0분 1초(KST)	2038년 1월 19일 3시 14분 7초(GMT) 2038년 1월 19일 12시 14분 7초(KST)	예외적으로 TIMESTAMP '0000-00-00 00:00:00'을 입력할 수 있다.
DATETIME	8	0001년 1월 1일 0시 0분 0.000초	9999년 12월 31일 23시 59분 59.999초	예외적으로 DATETIME '0000-00-00 00:00:00'을 입력할 수 있다.
TIMESTAMPPLTZ	4	1970년 1월 1일 0시 0분 1초(GMT) 타임존에 따라 다름	2038년 1월 19일 3시 14분 7초(GMT) 타임존에 따라 다름	로컬 타임존의 TIMESTAMP. 예외적으로, TIMESTAMPPLTZ'0000-00-00 00:00:00' 형태가 허용된다.
TIMESTAMPPTZ	8	1970년 1월 1일 0시 0분 1초(GMT) 타임존에 따라 다름	2038년 1월 19일 3시 14분 7초(GMT) 타임존에 따라 다름	로컬 타임존의 TIMESTAMP. 예외적으로, TIMESTAMPPTZ '0000-00-00 00:00:00' 형태가 허용된다.
DATETIMEELTZ	8	0001년 1월 1일 0시 0분 0.000초	9999년 12월 31일 23시 59분 59.999초	로컬 타임존의 DATETIME. 예외적으로,

타입	bytes	최소값	최대값	비고
		UTC 타임존에 따라 다름	초 타임존에 따라 다름	DATETIMELTZ '0000-00-00 00:00:00' 형태가 허용된다.
DATETIMEZ	12	0001년 1월1일 0시0분 0.000초 UTC 타임존에 따라 다름	9999년12월31일 23시59분59.999초 초 타임존에 따라 다름	타임존의 DATETIME. 예외적으로, DATETIMEZ '0000-00-00 00:00:00' 형태가 허용된다.

범위와 해상도(Range and Resolution)

- 시간 값의 표현은 기본적으로 24시간 시스템에 의하여 그 범위가 결정된다. 날짜는 그레고리력 (Gregorian calendar)을 따른다. 이 두 제약 사항을 벗어나는 값이 날짜나 시간으로 입력되면 오류가 발생한다.
- **DATE** 중 연도의 범위는 0001~9999 AD이다.
- CUBRID 2008 R3.0 버전부터는 연도를 두 자리만 표기하면, 00~69는 2000~2069로 변환되고, 70~99는 1970~1999로 변환된다. R3.0 미만 버전에서는 01~99까지의 두 자리 연도를 표기하면, 각각 0001~0099로 변환된다.
- **TIMESTAMP**의 범위는 GMT로 1970년 1월 1일 0시 0분 1초부터 2038년 1월 19일 03시 14분 07초까지이다. KST (GMT+9)로는 1970년 1월 1일 9시 0분 1초부터 2038년 1월 19일 12시 14분 07초까지 저장할 수 있다. GMT로 timestamp'1970-01-01 00:00:00'은 timestamp'0000-00-00 00:00:00'와 같다.
- **TIMESTAMPPLTZ, TIMESTAMPTZ** 범위는 타임존에 따라 다르지만 UTC로 변환되는 값은 1970-01-01 00:00:01과 2038-01-19 03 03:14:07 사이여야 한다.
- **DATETIMELTZ, DATETIMEZ** 범위는 타임존에 따라 다르지만 UTC로 변환되는 값은 0001-01-01 00:00:0.000과 9999-12-31 23:59:59.999 사이여야 한다. 세션 타임존이 변경되면 데이터베이스에 저장된 값이 더 이상 유효하지 않다.
- 날짜, 시간, 타임스탬프와 관련된 연산은 시스템의 반올림 시스템에 따라 결과가 달라질 수 있다. 이러한 경우, 시간과 타임스탬프는 가장 근접한 초를 최소 해상도로, 날짜는 가장 근접한 날짜를 최소 해상도로 하여 결정된다.

변환(Coercion)

날짜/시간 데이터 타입의 값은 서로 똑같은 항목을 가지고 있는 경우에만 **CAST** 연산자를 이용한 명시적인 변환이 가능하며, 묵시적 변환은 **묵시적 타입 변환** 을 참고한다. 아래의 표는 명시적 변환이 가능한 타입을 설명한다. 날짜/시간 데이터 타입 간 산술 연산에 대한 내용은 **날짜/시간 데이터 타입의 산술 연산과 타입 변환** 을 참고한다.

명시적 변환

FROM \ TO	DATE	TIME	DATETIME	TIMESTAMP
DATE	-	X	O	O
TIME	X	-	X	X
DATETIME	O	O	-	O
TIMESTAMP	O	O	O	-

DATE, **DATETIME**, **TIMESTAMP** 타입의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜와 시간이 모두 0인 값은 허용한다. 해당 타입의 칼럼에 대한 질의 수행 시 인덱스가 있으면 이 값을 사용할 수 있다는 점에서 **NULL** 대신 사용하면 유용하다.

- **DATE**, **DATETIME**, **TIMESTAMP** 타입이 인자인 일부 함수는 인자의 날짜와 시간 값이 모두 0이면 시스템 파라미터 **return_null_on_function_errors** 의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors** 가 yes이면 **NULL** 을 반환하고 no이면 에러를 반환하며, 기본값은 no 이다.
- **DATE**, **DATETIME**, **TIMESTAMP** 타입을 반환하는 함수들은 날짜와 시간 값이 모두 0인 값을 반환할 수 있지만 JAVA 응용 프로그램에서는 이러한 값을 Date 객체에 저장할 수 없다. 따라서 연결 URL 문자열의 zeroDateTimeBehavior 속성(Property) 설정에 따라서 예외로 처리하거나 **NULL**을 반환하거나 또는 최소값을 반환한다(이에 관한 자세한 내용은 **연결 설정** 참고).
- 시스템 파라미터 **intl_date_lang**을 설정하면 **TO_DATE()**, **TO_TIME()**, **TO_DATETIME()**, **TO_TIMESTAMP()**, **DATE_FORMAT()**, **TIME_FORMAT()**, **TO_CHAR()**, **STR_TO_DATE()** 함수의 입력 문자열 형식이 해당 로컬의 날짜 형식을 따른다. 자세한 내용은 **구문/타입 관련 파라미터**과 각 함수의 설명을 참고한다.
- 타임존이 포함된 타입은 상위 타입과 동일한 변환 규칙을 따른다.

참고

날짜/시간 타입 및 타임존이 있는 날짜/시간 타입의 리터럴에 대해서는 **날짜/시간**을 참고한다.

DATE

DATE 데이터 타입은 연도(*yyyy*), 월(*mm*), 일(*dd*)을 표현하며, 지원 범위는 '01/01/0001'에서 '12/31/9999'까지이다. 연도는 생략 가능하며, 생략될 경우 현재 시스템의 연도 값이 자동으로 지정된다. 입력 형식은 다음과 같다.

```
date 'mm/dd[/yyyy]'
date '[yyyy-]mm-dd'
```

- 모든 항목은 정수 형태로 입력되어야 한다.
- **CSQL**은 'MM/DD/YYYY' 형식으로 날짜 값을 출력하고, **JDBC 응용 프로그램** 및 **CUBRID 매니저**는 'YYYY-MM-DD' 형식으로 날짜 값을 출력한다.
- 시스템의 현재 날짜 값은 **SYS_DATE** (또는 **SYSDATE**, **CURRENT_DATE**) 함수를 이용하여 **DATE** 데이터 타입에 저장할 수 있다.
- 문자열 타입의 데이터를 **DATE** 타입으로 변환하는 함수는 **TO_DATE()** 이다.
- 연, 월, 일에는 0을 입력할 수 없으나 예외적으로 연, 월, 일이 모두 0인 '0000-00-00'은 입력할 수 있다.
- 연도에 두자리인 경우, 70 이상이면 1900년대로 인식하고 70미만이면 2000년대로 인식한다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```
DATE '2008-10-31'은 '10/31/2008'로 출력된다.
DATE '10/31'은 '10/31/2011'으로 출력된다(연도가 생략되면 현재 연도가 자동으로 지정됨).
DATE '00-10-31'은 '10/31/2000'로 출력된다.
DATE '0000-10-31'은 에러가 출력된다(연도의 최소값은 1).
DATE '70-10-31'은 '10/31/1970'로 출력된다.
DATE '0070-10-31'은 '10/31/0070'로 출력된다.
```

TIME

TIME 데이터 타입은 시각(*hh*), 분(*mi*), 초(*ss*)를 표현하며, 지원 범위는 '00:00:00'에서 '23:59:59'까지이다. 초는 생략 가능하며, 생략될 경우 0초로 지정된다. 입력 형식은 12시간 표기법(AM/PM 표기법) 또는 24시간 표기법이 모두 허용되며, 다음과 같이 작성한다.

```
time'hh:mi[:ss] [am | pm]'
```

- 모든 항목은 정수로 입력되어야 한다.
- CSQL은 항상 AM/PM 표기법으로 시간 값을 출력하고, JDBC 응용 프로그램 및 CUBRID 매니저는 24 시간 표기법으로 시간 값을 출력한다.
- 24시간 표기법으로 시간 값을 입력할 때에도 AM/PM을 지정할 수 있으며, 이때 시간 값과 지정된 AM 또는 PM이 일치하지 않으면 오류가 발생한다.
- 모든 시간 값은 데이터베이스에는 24시간 표기법으로 저장된다.
- 시스템의 현재 시간값은 `SYS_TIME` (또는 `SYSTIME`, `CURRENT_TIME`) 함수를 이용하여 **TIME** 데이터 타입에 저장할 수 있다.
- 문자열 타입의 데이터를 **TIME** 타입으로 변환하는 함수는 `TO_TIME()` 이다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```
TIME'00:00:00'은 '12:00:00 AM'으로 출력된다.
TIME'1:15'는 '01:15:00 AM'으로 간주된다.
TIME'13:15:45'는 '01:15:45 PM'으로 간주된다.
TIME'13:15:45 pm'은 정상적으로 저장된다.
TIME'13:15:45 am'은 오류가 발생한다(주어진 시간 값과 AM/PM이 불일치).
```

TIMESTAMP

TIMESTAMP 데이터 타입은 날짜(연, 월, 일)와 시간(시, 분, 초)을 결합한 데이터 값을 표현하며, GMT로 '1970-01-01 00:00:01'부터 '2038-01-19 03:14:07'까지 표현할 수 있다. 이 범위를 초과하거나 밀리초 단위의 시간 데이터를 저장하는 경우라면, **DATETIME** 데이터 타입을 이용할 수 있다. **TIMESTAMP** 데이터 타입의 입력 형식은 다음과 같다.

```
timestamp'hh:mi[:ss] [am|pm] mm/dd[/yyyy]'
timestamp'hh:mi[:ss] [am|pm] [yyyy-]mm-dd'

timestamp'mm/dd[/yyyy] hh:mi[:ss] [am|pm]'
timestamp'[yyyy-]mm-dd hh:mi[:ss] [am|pm]'
```

- 모든 항목은 정수로 입력되어야 한다.
- 연도를 생략하면 기본값으로 현재 연도가 지정되고, 시간 값(시/분/초)을 생략하면 12:00:00 AM으로 지정된다.
- 시스템의 현재 타임스탬프 값은 `SYS_TIMESTAMP` (또는 `SYSTIMESTAMP`, `CURRENT_TIMESTAMP`) 함수를 이용하여 **TIMESTAMP** 데이터 타입에 저장할 수 있다.

- **TIMESTAMP()** 함수 또는 **TO_TIMESTAMP()** 함수를 사용하면, 문자열 데이터 타입의 데이터를 **TIMESTAMP** 데이터 타입으로 변환할 수 있다.
- 연, 월, 일에는 0을 입력할 수 없으나 예외적으로 연, 월, 일, 시, 분, 초가 모두 0인 '0000-00-00 00:00:00'은 입력할 수 있다. GMT timestamp'1970-01-01 12:00:00 AM' 또는 KST timestamp'1970-01-01 09:00:00 AM'은 timestamp'0000-00-00 00:00:00'으로 해석된다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```
TIMESTAMP'10/31'은 '12:00:00 AM 10/31/2011'으로 출력된다(연도/시간이 생략
될 경우, 기본값으로 출력).
TIMESTAMP'10/31/2008'은 '12:00:00 AM 10/31/2008'로 출력된다(시간이 생략
될 경우, 기본값으로 출력).
TIMESTAMP'13:15:45 10/31/2008'은 '01:15:45 PM 10/31/2008'로 출력된다.
TIMESTAMP'01:15:45 PM 2008-10-31'은 '01:15:45 PM 10/31/2008'로 출력된
다.
TIMESTAMP'13:15:45 2008-10-31'은 '01:15:45 PM 10/31/2008'로 출력된다.
TIMESTAMP'10/31/2008 01:15:45 PM'은 '01:15:45 PM 10/31/2008'로 출력된
다.
TIMESTAMP'10/31/2008 13:15:45'는 '01:15:45 PM 10/31/2008'로 출력된다.
TIMESTAMP'2008-10-31 01:15:45 PM'은 '01:15:45 PM 10/31/2008'로 출력된
다.
TIMESTAMP'2008-10-31 13:15:45'는 '01:15:45 PM 10/31/2008'로 출력된다.
TIMESTAMP'2099-10-31 01:15:45 PM'은 오류가 발생한다(TIMESTAMP 표현 가능 범
위 초과).
```

DATETIME

DATETIME 타입은 날짜(년, 월, 일)와 시간(시, 분, 초, 밀리초)을 결합한 데이터 값을 표현하며, GMT로 0001-01-01 00:00:00.000부터 9999-12-31 23:59:59.999까지 표현할 수 있다. **DATETIME** 타입 데이터의 입력 형식은 다음과 같다.

```
datetime'hh:mi[:ss[:msec]] [am|pm] mm/dd[/yyyy]'
datetime'hh:mi[:ss[:msec]] [am|pm] [yyyy-]mm-dd'
datetime'mm/dd[/yyyy] hh:mi[:ss[:ff]] [am|pm]'
datetime'[yyyy-]mm-dd hh:mi[:ss[:ff]] [am|pm]'
```

- 모든 항목은 정수로 입력되어야 한다.
- 연도를 생략하면 기본값으로 현재 연도가 지정되고, 시간 값(시/분/초/밀리초)을 생략하면 12:00:00.000 AM으로 지정된다.
- 시스템의 현재 타임스탬프 값은 **SYS_DATETIME** (또는 **SYSDATETIME**, **CURRENT_DATETIME**, **CURRENT_DATETIME()**, **NOW()**)를 이용하여 **DATETIME** 타입에 저장할 수 있다.
- 문자열 타입의 데이터를 **DATETIME** 타입으로 변환하는 함수는 **TO_DATETIME()** 이다.

- 연, 월, 일에는 0을 입력할 수 없으나 예외적으로 연, 월, 일, 시, 분, 초가 모두 0인 '0000-00-00 00:00:00'은 입력할 수 있다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```

DATETIME '10/31'은 '12:00:00.000 AM 10/31/2011'으로 출력된다(연도/시간이 생략될 경우, 기본값으로 출력).
DATETIME '10/31/2008'은 '12:00:00.000 AM 10/31/2008'로 출력된다.
DATETIME '13:15:45 10/31/2008'은 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '01:15:45 PM 2008-10-31'은 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '13:15:45 2008-10-31'은 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '10/31/2008 01:15:45 PM'은 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '10/31/2008 13:15:45'는 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '2008-10-31 01:15:45 PM'은 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '2008-10-31 13:15:45'는 '01:15:45.000 PM 10/31/2008'로 출력된다.
DATETIME '2099-10-31 01:15:45 PM'은 '01:15:45.000 PM 10/31/2099'로 출력된다.

```

문자열을 날짜/시간 타입으로 CAST

날짜/시간 타입 문자열 권장 형식

CAST() 연산자를 사용하여 문자열을 날짜/시간 타입으로 변환할 때에는 문자열을 다음과 같은 형식으로 작성하는 것을 권장한다. 참고로, **CAST()** 연산자에서 사용하는 날짜/시간 문자열 형식은 DB 생성 시 지정하는 로캘의 영향을 받지 않는다.

또한, **TO_DATE()**, **TO_TIME()**, **TO_DATETIME()**, **TO_TIMESTAMP()** 함수에서 문자열에 대한 날짜/시간 형식을 생략하는 경우에도 날짜/시간 문자열 형식을 아래와 같이 작성한다.

- **DATE** 타입

```

YYYY-MM-DD
MM/DD/YYYY

```

- **TIME** 타입

```
HH:MI:SS [AM|PM]
```

· DATETIME 타입

```
YYYY-MM-DD HH:MI:SS[.msec] [AM|PM]
HH:MI:SS[.msec] [AM|PM] YYYY-MM-DD

MM/DD/YYYY HH:MI:SS[.msec] [AM|PM]
HH:MI:SS[.msec] [AM|PM] MM/DD/YYYY
```

· TIMESTAMP 타입

```
YYYY-MM-DD HH:MI:SS [AM|PM]
HH:MI:SS [AM|PM] YYYY-MM-DD

MM/DD/YYYY HH:MI:SS [AM|PM]
HH:MI:SS [AM|PM] MM/DD/YYYY
```

날짜/시간 타입 문자열 허용 형식

`CAST()` 연산자는 날짜/시간 문자열에 대해 아래와 같은 형식을 허용한다.

DATE 문자열 허용 형식

```
[year sep] month sep day
```

· 2011-04-20 : 2011년 4월 20일

· 04-20 : 올해 4월 20일

구분자(*sep*)가 빗금(/)일 때에는 다음과 같은 순서로 인식한다.

```
month/day[/year]
```

· 04/20/2011 : 2011년 4월 20일

· 04/20 : 올해 4월 20일

구분자(*sep*)를 사용하지 않을 때에는 다음과 같은 형식으로 인식한다. 연도는 한 자리, 두 자리, 네 자리를 허용하고, 월은 한 자리, 두 자리를 허용한다. 일은 항상 두 자리를 입력해야 한다.

YYYYMMDD
 YYMMDD
 YMMDD
 MMDD
 MDD

- 20110420 : 2011년 4월 20일
- 110420 : 2011년 4월 20일
- 420 : 올해 4월 20일

TIME 문자열 허용 형식

[hour]:min[:[sec]][.[msec]] [am|pm]

- 09:10:15.359 am : 오전 9시 10분 15초(0.359초는 버림)
- 09:10:15 : 오전 9시 10분 15초
- 09:10 : 오전 9시 10분
- :10 : 오전 12시 10분

[[[[[[Y]Y]Y]Y]M]MDD]HHMISS[.[msec]] [am|pm]

- 20110420091015.359 am : 오전 9시 10분 15초
- 0420091015 : 오전 9시 10분 15초

[H]HMMSS[.[msec]] [am|pm]

- 091015.359 am : 오전 9시 10분 15초
- 91015 : 오전 9시 10분 15초

[M]MSS[.[msec]] [am|pm]

- 1015.359 am : 오전 12시 10분 15초
- 1015 : 오전 12시 10분 15초


```
[S]S[.msec]] [am|pm]
```

- 15.359 am : 오전 12시 0분 15초
- 15 : 오전 12시 0분 15초

참고

CUBRID 2008 R3.1 이하 버전에서는 [H]H 형식을 허용했다. 즉 R3.1 이하 버전에서 문자열 '10'은 **TIME**'10:00:00'으로 변환되었으나, R4.0부터는 **TIME**'00:00:10'으로 변환된다.

DATETIME 문자열 허용 형식

```
[year sep] month sep day [sep] [sep] hour [sep min[sep sec[.msec]]]
```

- 04-20 09 : 올해 4월 20일 오전 9시

```
month/day[/year] [sep] hour [sep min [sep sec[.msec]]]
```

- 04/20 09 : 올해 4월 20일 오전 9시

```
year sep month sep day sep hour [sep min[sep sec[.msec]]]
```

- 2011-04-20 09 : 2011년 4월 20일 오전 9시

```
month/day/year sep hour [sep min[sep sec [.msec]]]
```

- 04/20/2011 09 : 2011년 4월 20일 오전 9시

```
YYMMDDH (시간이 한 자리 수일 때에만 허용)
```

- 1104209 : 2011년 4월 20일 오전 9시

```
YYMMDDHHMI[SS[.msec]]
```

- 1104200910.359 : 2011년 4월 20일 오전 9시 10분(0.359초는 버림)

- 110420091000.359 : 2011년 4월 20일 오전 9시 10분 0.359초

```
YYYYMMDDHHMISS[.msec]
```

- 201104200910.359 : 2020년 11월 4일 오후 8시 9분 10.359초
- 20110420091000.359 : 2011년 4월 20일 오전 9시 10분 0.359초

시간-날짜 순서의 문자열 허용 형식

```
[hour]:min[:sec[.msec]] [am|pm] [year-]month-day
```

- 09:10:15.359 am 2011-04-20 : 2011년 4월 20일 오전 9시 10분 15.359초
- :10 04-20 : 올해 4월 20일 오전 12시 10분

```
[hour]:min[:sec[.msec]] [am|pm] month/day[/[year]]
```

- 09:10:15.359 am 04/20/2011 : 2011년 4월 20일 오전 9시 10분 15.359초
- :10 04/20 : 올해 4월 20일 오전 12시 10분

```
hour[:min[:sec[.msec]]] [am|pm] [year-]month-day
```

- 09:10:15.359 am 04-20 : 올해 4월 20일 오전 9시 10분 15.359초
- 09 04-20 : 올해 4월 20일 오전 9시

```
hour[:min[:sec[.msec]]] [am|pm] month/day[/[year]]
```

- 09:10:15.359 am 04/20 : 올해 4월 20일 오전 9시 10분 15.359초
- 09 04/20 : 올해 4월 20일 오전 9시

규칙

*msec*은 밀리초를 나타내는 일련의 숫자이다. 앞에서 네 번째 자리부터 이후의 숫자는 무시된다. 값 사이를 구분하는 구분자의 규칙은 다음과 같다.

- **TIME** 문자열은 시간 구분자로 항상 하나의 콜론(:)을 사용해야 한다.
- **DATE** 와 **DATETIME** 문자열은 구분자 없이 연속된 숫자로 나타낼 수 있고, **DATETIME** 문자열은 시간과 날짜를 공백으로 구분할 수 있다.

- 입력 문자열 안에서 구분자들은 동일해야 한다.
- 시간-날짜 순서의 문자열은 시간 구분자로 콜론(:)만 사용할 수 있으며, 날짜 구분자로 하이픈(-)이나 빗금(/)만 사용할 수 있다. 날짜 입력 시 하이픈을 사용하는 경우 yyyy-mm-dd 순으로 입력하며, 빗금(/)을 사용하는 경우 mm/dd/yyyy 순으로 입력한다.

날짜 부분의 문자열에는 다음 규칙이 적용된다.

- 연도는 구문이 허용하는 한 생략할 수 있다.
- 연도를 두 자리로 입력하면 1970년~2069년 범위의 연도를 나타낸다. 즉, YY<70 이면 2000+YY으로 처리하고, YY>=70이면 1900+YY으로 처리한다. 한 자리나 세 자리, 네 자리 숫자로 연도를 입력하면 해당 숫자 그대로를 나타낸다.
- 문자열 앞뒤의 공백과 뒤의 문자열은 무시된다. **DATETIME**, **TIME** 문자열을 위한 am/pm 지정자는 시간 값의 일부로 인식하지만, 공백이 아닌 문자가 뒤에 붙으면 am/pm 지정자로 인식되지 않는다.

CUBRID의 **TIMESTAMP** 타입은 **DATE** 타입과 **TIME** 타입으로 구성되고, **DATETIME** 타입은 **DATE** 타입과 **TIME** 타입에 밀리초(milliseconds)가 더해져서 구성된다. 입력 문자열은 날짜(**DATE** 문자열), 시간(**TIME** 문자열), 혹은 둘 다(**DATETIME** 문자열) 포함할 수 있다. 특정 타입의 데이터가 보유한 문자열은 다른 타입으로도 변환될 수 있으며 다음과 같은 규칙이 적용된다.

- **DATE** 문자열을 **DATETIME** 타입으로 변환하면 시간 값은 '00:00:00'이 된다.
- **TIME** 문자열을 **DATETIME** 타입으로 변환하면 콜론(:)이 날짜 구분자로 인식되어 **TIME** 문자열이 날짜를 나타내는 문자열로 인식되고, 시간 값은 '00:00:00'이 된다.
- **DATETIME** 문자열을 **DATE** 타입으로 변환하면 결과값에서 시간 부분은 무시되지만, 시간 입력값의 형식은 유효해야 한다.
- **DATETIME** 문자열을 **TIME** 타입으로 변환할 수 있지만, 다음과 같은 규칙이 적용된다.
 - 문자열에 있는 날짜와 시간은 최소한 하나의 공백에 의해 구분되어야 한다.
 - 결과값에서 날짜 부분은 무시되지만, 날짜 입력값의 형식이 유효해야 한다.
 - 날짜 부분의 연도가 4자리 이상이거나(0으로 시작할 수 있음), 시간 부분이 최소한 시와 분([H]H:[M]M)을 포함해야 한다. 그렇지 않으면 날짜 부분이 [MM]SS 포맷의 **TIME** 타입으로 인식되고, 뒤이어 나오는 문자열은 무시된다.
- **DATETIME** 문자열의 각 단위(년, 월, 일, 시, 분, 초) 중 하나가 999999보다 크면, 숫자가 아닌 것으로 인식하여 해당 단위가 포함된 문자열이 무시된다. 예를 들어 '2009-10-21 20:9943:10'은 분 단위의 값이 범위를 벗어나므로 에러가 발생한다. 그러나 '2009-10-21 20:1000123:10'이 입력되면 '2009'를 MMSS 포맷의 **TIME** 타입으로 인식하여 **TIME**'00:20:09'를 반환한다.

- 시간-날짜 순서의 문자열을 **TIME** 타입으로 변환하면 문자열의 날짜 부분은 무시되지만, 날짜 부분의 형식은 유효해야 한다.
- 시간 부분이 있는 모든 입력 문자열은 변환 시 [*msec*] 을 허용하지만, **DATETIME** 타입만 그 값을 유지한다. **DATE**, **TIMESTAMP**, **TIME** 와 같은 타입으로 변환하면 *msec* 값을 버린다.
- **DATETIME**, **TIME** 문자열에서의 모든 변환은 시간 값 뒤에 나오는 영문 로캘(locale) 또는 서버의 현재 로캘로 쓰여진 am/pm 지정자를 허용한다.

```
SELECT CAST('420' AS DATE);
```

```
cast('420' as date)
=====
04/20/2012
```

```
SELECT CAST('91015' AS TIME);
```

```
cast('91015' as time)
=====
09:10:15 AM
```

```
SELECT CAST('110420091035.359' AS DATETIME);
```

```
cast('110420091035.359' as datetime)
=====
09:10:35.359 AM 04/20/2011
```

```
SELECT CAST('110420091035.359' AS TIMESTAMP);
```

```
cast('110420091035.359' as timestamp)
=====
09:10:35 AM 04/20/2011
```

타임존이 있는 날짜/시간 데이터 타입

타임존이 있는 날짜/시간 데이터 타입은 타임존을 명시하여 입력하거나 출력할 수 있는 날짜/시간 타입이다. 타임존을 설정하는 방법은 지역 이름을 명시하는 방법과 시간의 오프셋을 명시하는 방법이 있다.

기존의 날짜/시간 타입 이름 뒤에 TZ 또는 LTZ가 붙어 있는 경우 타임존 정보를 고려하게 되는데, TZ는 타임존을 의미하며, LTZ는 로컬 타임존을 의미한다.

- TZ 타입은 <date/time type> WITH TIME ZONE으로도 표현이 가능하다. 내부적으로 UTC 시간과 생성 시 타임존 정보(사용자가 명시하거나 세션 타임존에 의해 결정됨)를 저장한다. TZ 타입은 타임존을 저장하기 위해 4바이트가 추가로 필요하다.
- LTZ 타입은 <date/time type> WITH LOCAL TIME ZONE으로도 표현이 가능하다. 내부적으로 UTC 시간을 저장하며, 출력 시 로컬(현재의 세션) 타임존으로 변환된다.

타임존이 없는 타입과 비교하기 위해, 다음 표에는 타임존이 없는 타입과 타임존이 있는 타입을 함께 설명하였다.

표의 설명에 있는 UTC는 협정 세계시(Coordinated Universal Time)를 나타낸다.

구분	타입	입력	저장	출력	설명
DATE	DATE	타임존 비포함	입력 값	절대 값(입력 값과 동일)	날짜
DATETIME	DATETIME	타임존 비포함	입력 값	절대 값(입력 값과 동일)	밀리초를 포함한 날짜/시간
	DATETIMETZ	타임존 포함	UTC + 타임존 (지역 또는 오프셋)	절대 값(입력한 타임존 유지)	날짜/시간 + 타임존 정보
	DATETIMELTZ	타임존 포함	UTC	상대 값(세션 타임존에 따라 변환됨)	세션 타임존에서의 날짜/시간
TIME	TIME	타임존 비포함	입력 값		시간

구분	타입	입력	저장	출력	설명
				절대 값(입력 값과 동일)	
TIMESTAMP	TIMESTAMP	타임존 비포함	UTC	상대 값(세션 타임존에 따라 변환됨)	입력 값을 세션 타임존의 값으로 해석함
	TIMESTAMPTZ	타임존 포함	UTC + 타임존 (지역 또는 오프셋)	절대 값(입력한 타임존 유지)	UTC + 타임존이 있는 타임스탬프
	TIMESTAMPTZ	타임존 포함	UTC	상대 값(세션 타임존에 따라 변환됨)	세션 타임존. TIMESTAMP의 값과 같음. 출력할 때 타임존 지정자를 포함함

타임존이 있는 날짜/시간 타입의 최대값, 최소값, 범위와 해상도 등 나머지 특징들은 일반적인 날짜/시간 타입의 특징과 동일하다.

참고

- CUBRID에서, TIMESTAMP가 1970년 1월 1일 UTC 이후 경과된 '초'로 보관된다(UNIX 시간).
- 타 DBMS의 TIMESTAMP는 CUBRID의 DATETIME과 비슷한 방식이며 'milliseconds'를 보관한다.

타임존 타입을 사용하는 함수의 예를 보려면 다음을 참고한다. [날짜/시간 함수와 연산자](#)

다음은 세션 타임존의 변경에 따라 DATETIME, DATETIMETZ와 DATETIMELTZ의 출력 값이 다르게 나타나는 예이다.

```
-- csql> ;set timezone="+09"
```

```
CREATE TABLE tbl (a DATETIME, b DATETIMETZ, c DATETIMELTZ);
INSERT INTO tbl VALUES (datetime'2015-02-24 12:30',
datetimeetz'2015-02-24 12:30', datetimeltz'2015-02-24 12:30');

SELECT * FROM tbl
```

```
12:30:00.000 PM 02/24/2015      12:30:00.000 PM 02/24/2015 +09:
00                          12:30:00.000 PM 02/24/2015 +09:00
```

```
-- csql> ;set timezone="+07"
```

```
SELECT * FROM tbl;
```

```
12:30:00.000 PM 02/24/2015      12:30:00.000 PM 02/24/2015 +09:
00                          10:30:00.000 AM 02/24/2015 +07:00
```

다음은 세션 타임존의 변경에 따라 `TIMESTAMP`, `TIMESTAMPTZ`와 `TIMESTAMPLTZ`의 출력 값이 다르게 나타나는 예이다.

```
-- ;set timezone="+09"
```

```
CREATE TABLE tbl (a TIMESTAMP, b TIMESTAMPTZ, c TIMESTAMPLTZ);
INSERT INTO tbl VALUES (timestamp'2015-02-24 12:30',
timestamptz'2015-02-24 12:30', timestamppltz'2015-02-24 12:30');

SELECT * FROM tbl;
```

```
12:30:00 PM 02/24/2015      12:30:00 PM 02/24/2015 +09:
00                          12:30:00 PM 02/24/2015 +09:00
```

```
-- csql> ;set timezone="+07"
```

```
SELECT * FROM tbl;
```

```
10:30:00 AM 02/24/2015      12:30:00 PM 02/24/2015 +09:
00                          10:30:00 AM 02/24/2015 +07:00
```

문자열을 `TIMESTAMP` 타입으로 변환

문자열을 timestamp/timestampltz/timestamptz로 변환하는 작업은 문자열로부터 TIMESTAMP 객체를 생성하는 과정에서 수행된다.

From/to	Timestamp	Timestampltz	Timestamptz
String (타임존 생략)	세션 타임존으로 날짜/시간을 UTC로 변환하고 Unix 시간으로 인코딩하여 저장	세션 타임존으로 날짜/시간을 UTC로 변환하고 Unix 시간으로 인코딩하여 저장	세션 타임존으로 날짜/시간을 UTC로 변환하고 Unix 시간과 세션의 TZ_ID로 인코딩하여 저장
String (타임존 포함)	오류 (timestamp 에서는 타임존을 허용하지 않음)	문자열의 타임존으로 날짜/시간을 UTC로 변환하고 Unix 시간으로 인코딩하여 저장	문자열의 타임존으로 날짜/시간을 UTC로 변환하고 Unix 시간과 세션의 TZ_ID로 인코딩하여 저장

문자열을 DATETIME 타입으로 변환

문자열을 datetime/datetimeltz/datetimetz로 변환하는 작업은 문자열로부터 DATETIME 객체를 생성하는 과정에서 수행된다.

From/to	Datetime	Datetimeltz	Datetimetz
String (타임존 생략)	문자열에서 분석된 값을 저장	세션 타임존으로 날짜/시간을 UTC로 변환하고 새로운 값들로 저장	세션 타임존으로 날짜/시간을 UTC로 변환하고 새로운 값과 세션의 TZ_ID로 저장
String (타임존 포함)	오류 (datetime 에서는 타임존을 허용하지 않음)	문자열의 타임존으로 날짜/시간을 UTC로 변환하고 새로운 값들로 저장	문자열의 타임존으로 날짜/시간을 UTC로 변환하고 새로운 값과 세션의 TZ_ID로 저장

DATETIME 및 TIMESTAMP 타입을 문자열로 변환

From/to	String (타임존 출력 불허)	String (타임존 강제 출력)	String (타임존 비요청 - 자동 선택)
TIMESTAMP	세션 타임존으로 Unix 시간을 디코딩한 후 출력	세션 타임존으로 Unix 시간을 디코딩한 후 세션 타임존 문자열과 함께 출력	세션 타임존으로 Unix 시간을 디코딩한 후 출력. 세션 타임존 문자열은 출력하지 않음
TIMESTAMPSTZ	세션 타임존으로 Unix 시간을 디코딩한 후 출력	세션 타임존으로 Unix 시간을 디코딩한 후 세션 타임존 문자열과 함께 출력	세션 타임존으로 Unix 시간을 디코딩한 후 출력. 세션 타임존 문자열 출력
TIMESTAMPZ	값의 타임존으로 Unix 시간을 디코딩한 후 출력	값의 타임존으로 Unix 시간을 디코딩한 후 값의 타임존 문자열과 함께 출력	값의 타임존으로 Unix 시간을 디코딩한 후 출력. 값의 타임존 문자열 출력
DATETIME	저장된 값을 출력	세션 타임존 문자열과 함께 저장된 값을 출력	저장된 값을 출력. 타임존 문자열은 출력하지 않음
DATETIMESTZ	세션 타임존으로 UTC 값을 변환후 출력	세션 타임존으로 UTC 값을 변환 후 세션 타임존 문자열과 함께 출력	세션 타임존으로 UTC 값을 변환후 출력. 세션 타임존 문자열 출력
DATETIMEZ	값의 타임존으로 UTC 값을 변환후 출력	값의 타임존으로 UTC 값을 변환 후 값의 타임존 문자열과 함께 출력	값의 타임존으로 UTC 값을 변환후 출력. 값의 타임존 문자열 출력

타임존 설정

다음은 cubrid.conf 파일에서 설정하는 타임존 관련 파라미터들이다. 파라미터의 설정과 관련해서는 [cubrid.conf 설정 파일과 기본 제공 파라미터](#)를 참고한다.

- **timezone**

세션에 대한 타임존을 설정하며, 기본값은 **server_timezone**의 값이다.

- **server_timezone**

서버에 대한 타임존을 설정하며, 기본값은 OS의 타임존이다.

- **tz_leap_second_support**

윤초(leap second)에 대한 지원 여부를 yes 또는 no로 설정하며, 기본값은 no이다.

타임존 함수

다음은 타임존과 관련된 함수들이다. 설명을 보려면 각 함수의 이름을 클릭한다.

- [DBTIMEZONE\(\)](#)

- [SESSIONTIMEZONE\(\)](#)

- [FROM_TZ\(\)](#)

- [NEW_TIME\(\)](#)

- [TZ_OFFSET\(\)](#)

타임존 타입을 사용하는 함수

DATETIME, TIMESTAMP, TIME 타입의 값을 입력 값으로 사용하는 함수들은 모두 타임존 타입을 사용할 수 있다.

다음은 타임존 타입 값을 사용하는 예인데, 타임존이 없는 경우와 동일하게 동작한다. 다만, LTZ로 끝나는 타입의 경우 출력 값은 로컬 타임존의 설정(timezone 파라미터)을 따른다.

다음 예에서 숫자의 기본 단위는 DATETIME 타입의 최소 단위인 밀리초이다.

```
SELECT datetimeltz '09/01/2009 03:30:30 pm' + 1;
```

```
03:30:30.001 PM 09/01/2009 Asia/Seoul
```

```
SELECT datetimeltz '09/01/2009 03:30:30 pm' - 1;
```

```
03:30:29.999 PM 09/01/2009 Asia/Seoul
```

다음 예에서 숫자의 기본 단위는 TIMESTAMP 타입의 최소 단위인 초이다.

```
SELECT timestampz '09/01/2009 03:30:30 pm' + 1;
```

```
03:30:31 PM 09/01/2009 Asia/Seoul
```

```
SELECT timestampz '09/01/2009 03:30:30 pm' - 1;
```

```
03:30:29 PM 09/01/2009 Asia/Seoul
```

```
SELECT EXTRACT (hour from datetimetz'10/15/1986 5:45:15.135 am
Europe/Bucharest');
```

```
5
```

이름이 LTZ로 끝나는 타입은 출력 시 로컬 타임존의 설정을 따른다. 따라서 아래 예와 같이 timezone 파라미터의 값이 'Asia/Seoul'로 설정되어 있다면 EXTRACT 함수는 해당 타임존의 시(hour)를 출력한다.

```
-- csq> ;set timezone='Asia/Seoul'
```

```
SELECT EXTRACT (hour from datetimetz'10/15/1986 5:45:15.135 am
Europe/Bucharest');
```

```
12
```

타임존 타입에 대한 변환 함수

다음은 문자열에서 날짜/시간 타입 값으로 변환하거나 반대로 날짜/시간 타입 값에서 문자열로 변환하는 함수들인데, 이들의 입력 값에는 오프셋, 존, 일광 절약과 같은 타임존 정보가 추가될 수 있다.

- `DATE_FORMAT()`
- `STR_TO_DATE()`
- `TO_CHAR()`

- `TO_DATETIME_TZ()`
- `TO_TIMESTAMP_TZ()`

각 함수들의 사용 방법은 함수 이름을 클릭하여 해당 함수의 설명을 참고한다.

```
SELECT DATE_FORMAT (datetimetz'2012-02-02 10:10:10 Europe/Zurich
CET', '%TZR %TZD %TZh %TZM');
SELECT STR_TO_DATE ('2001-10-11 02:03:04 AM Europe/Bucharest EEST',
'%Y-%m-%d %h:%i:%s %p %TZR %TZD');
SELECT TO_CHAR (datetimetz'2001-10-11 02:03:04 AM Europe/Bucharest
EEST');
SELECT TO_DATETIME_TZ ('2001-10-11 02:03:04 AM Europe/Bucharest
EEST');
SELECT TO_TIMESTAMP_TZ ('2001-10-11 02:03:04 AM Europe/Bucharest');
```

참고

`TO_TIMESTAMP_TZ()` , `TO_DATETIME_TZ()` 함수는 날짜/시간 인자에 TZR, TZD, TZh 및 TZM 정보를 포함할 수 있다는 것을 제외하고는 `TO_TIMESTAMP()` 및 `TO_DATETIME()` 함수와 동일하다.

타임존의 지역 이름은 IANA(Internet Assigned Numbers Authority) 타임존 데이터베이스에 있는 지역을 사용하는데, IANA 타임존에 대해서는 <http://www.iana.org/time-zones>을 참고한다.

IANA 타임존

IANA(Internet Assigned Numbers Authority) 타임존 데이터베이스에는 수많은 세계 대표 장소에 대한 지역 시간의 역사를 표현하는 코드와 데이터가 들어 있다.

이 데이터베이스는 타임 존 경계, UTC 오프셋, 그리고 일광 절약 규칙에 대해 정치체에 의해 변경된 사항을 반영하기 위해 정기적으로 업데이트되고 있으며, 관리 절차는 *BCP 175: Procedures for Maintaining the Time Zone Database*. <<https://datatracker.ietf.org/doc/html/rfc6557>>에 설명되어 있다. 자세한 사항은 <http://www.iana.org/time-zones>를 참고한다.

CUBRID는 IANA 타임존을 지원하며, CUBRID 설치 패키지에 포함되어 있는 IANA 타임존 라이브러리를 그대로 사용할 수 있다. 최신 타임존으로 업데이트하고 싶은 경우 타임존 데이터를 업데이트하고, 타임존 라이브러리를 컴파일한 후 데이터베이스를 재구동해야 한다.

이와 관련하여 [타임존 라이브러리 컴파일](#)을 참고한다.

비트 문자열 데이터 타입

비트 문자열은 0과 1로 이루어진 8비트 단위의 이진 값의 순열(sequence)이다. CUBRID는 두 가지 비트 문자열을 지원한다.

- 고정길이 비트 문자열(**BIT**)
- 가변길이 비트 문자열(**BIT VARYING**)

메서드의 인자나 속성의 타입으로 비트열을 사용할 수 있으며, 비트열 리터럴은 2진수 형식이나 16진수 형식을 사용한다. 2진수 형식으로 사용할 때에는 다음과 같이 문자 **B** 뒤에 0과 1로 이루어진 문자열을 붙이거나, **0b** 뒤에 값을 붙여 표현한다.

```
B'1010'
0b1010
```

16진수 형식은 대문자 **X** 뒤에 0-9 그리고 A-F 문자로 이루어진 문자열을 붙이거나 **0x** 뒤에 값을 붙여 표현한다. 예를 들어, 아래는 앞에서 2진수로 표현한 것과 같은 값을 16진수로 나타낸 것이다.

```
X'a'
0xA
```

16진수에서 사용되는 문자는 대소문자를 구분하지 않는다. 즉, X'4f'와 X'4F'는 같은 값으로 간주한다.

길이(Length)

비트열이 테이블 속성이나 메서드 선언에 사용될 때에는 최대 길이를 표시해야 한다. 비트열이 가질 수 있는 최대 길이는 1,073,741,823비트이다.

비트열의 변환(Bit String Coercion)

고정길이와 가변길이 비트열 간에는 서로 비교를 위하여 자동 변환이 이루어진다. 명시적인 변환은 **CAST** 연산자를 이용해야 한다.

BIT(n)

고정길이 2진수 혹은 16진수 비트열은 **BIT** (*n*)로 나타내는데, 여기서 *n* 은 최대 비트의 개수를 나타낸다. 만약, *n* 이 생략되면 길이는 1로 지정된다. 비트열은 8비트 단위로 왼쪽부터 값이 채워진다. 예를 들어, B'1'의 값은 B'10000000'과 같은 값으로 출력된다. 따라서, 8비트 단위로 길이를 선언하고 8비트 단위로 값을 입력할 것을 권장한다.

참고

BIT(4)로 선언된 칼럼에 B'1'을 INSERT하면 CSQL에서는 X'8'로 출력되고, CUBRID Manager에서는 X'80'으로 출력된다.

- n 은 0보다 큰 숫자여야 한다.
- 비트열의 크기가 n 을 넘어설 경우에는 절삭되고, 0으로 채워진다.
- n 보다 작은 비트열이 저장될 때에는 나머지 오른쪽 부분이 0으로 채워진다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```
CREATE TABLE bit_tbl(a1 BIT, a2 BIT(1), a3 BIT(8), a4 BIT VARYING);
INSERT INTO bit_tbl VALUES (B'1', B'1', B'1', B'1');
INSERT INTO bit_tbl VALUES (0b1, 0b1, 0b1, 0b1);
INSERT INTO bit_tbl(a3,a4) VALUES (B'1010', B'1010');
INSERT INTO bit_tbl(a3,a4) VALUES (0xaa, 0xaa);
SELECT * FROM bit_tbl;
```

a1	a2	a3
a4		
=====		
====		
X'8'	X'8'	X'80'
X'8'		
X'8'	X'8'	X'80'
X'8'		
NULL	NULL	X'a0'
X'a'		
NULL	NULL	X'aa'
X'aa'		

BIT VARYING(n)

가변길이 비트열은 **BIT VARYING** (n)으로 나타낸다. 여기서 n 은 최대 비트의 개수를 나타낸다. 만약, n 이 생략되면 최대 길이인 1,073,741,823으로 지정된다. 비트열은 8비트 단위로 왼쪽부터 값이 채워진다. 예를 들어, B'1'의 값을 입력하면 B'10000000'과 같은 값으로 출력된다. 따라서, 8비트 단위로 크기를 선언하고 8비트 단위로 값을 입력할 것을 권장한다.

참고

BIT VARYING(4)로 선언된 칼럼에 B'1'을 INSERT하면 CSQL에서는 X'8'로 출력되고, CUBRID Manager 또는 응용 프로그램에서는 X'80'으로 출력된다.

- 비트열의 크기가 n 을 넘어설 경우에는 절삭되고 0으로 채워진다.
- n 보다 작은 비트열이 저장될 때에도 나머지 부분이 0으로 채워지지 않는다.
- n 은 0보다 큰 숫자여야 한다.
- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

```
CREATE TABLE bitvar_tbl(a1 BIT VARYING, a2 BIT VARYING(8));
INSERT INTO bitvar_tbl VALUES (B'1', B'1');
INSERT INTO bitvar_tbl VALUES (0b1010, 0b1010);
INSERT INTO bitvar_tbl VALUES (0xaa, 0xaa);
INSERT INTO bitvar_tbl(a1) VALUES (0xaaa);
INSERT INTO bitvar_tbl(a2) VALUES (0xaaa);
SELECT * FROM bitvar_tbl;
```

a1	a2
X'8'	X'8'
X'a'	X'a'
X'aa'	X'aa'
X'aaa'	NULL
NULL	X'aa'

문자열 데이터 타입

CUBRID는 두 종류의 문자열(character string) 타입을 지원한다.

- 고정길이 문자열 : **CHAR** (n)
- 가변길이 문자열 : **VARCHAR** (n)

참고

NCHAR, NCHAR VARYING 은 9.0 버전부터 더 이상 지원하지 않으며, 대신 **CHAR, VARCHAR** 타입을 사용하도록 한다.

다음은 문자열 타입을 사용할 때 적용되는 규칙이다.

- 문자열은 작은 따옴표로 감싸서 표현한다. SQL 구문 관련 파라미터인 **ansi_quotes**의 값에 따라 문자열을 감싸는 부호로 큰 따옴표도 사용할 수 있다. **ansi_quotes** 값을 **no**로 설정하면 큰 따옴표로 감싼 문자열을 식별자로 처리하지 않고 문자열로 처리한다. 기본값은 **yes**이다. 자세한 설명은 [구문/타입 관련 파라미터](#)를 참고한다.
- ANSI 표준에 따라 두 개의 문자열 사이에 공간으로 취급할 수 있는 문자(예: 공백, 탭, 줄바꿈 등)가 있다면, 두 개의 문자열은 연속된 하나의 문자열로 취급된다. 예를 들면, 다음과 같이 두 개의 문자열 사이에 줄바꿈이 있는 경우가 있다.

```
'abc'
'def'
```

위 문자열은 아래에 있는 하나의 문자열과 동일하다.

```
'abcdef'
```

- 작은 따옴표 자체를 문자열에 포함시키려면, 두 개의 작은 따옴표를 연속으로 입력하면 된다. 예를 들어, 아래의 왼쪽 문자열은 실제로 오른쪽과 같이 저장된다.

```
'''abcde''fghij'      'abcde'fghij
```

- 모든 문자열에 대한 토큰의 최대 크기는 16KB이다.
- 특정 국가의 언어를 입력하고자 하는 경우 DB 생성 시 언어의 로캘 이름과 문자셋을 지정하며, 이후 **CHARSET** 소개자(혹은 **COLLATE** 수정자)에 의해 로캘을 변경하여 사용할 수도 있다. 이에 대한 자세한 설명은 [다국어 개요](#)를 참고한다.

길이(Length)

문자의 개수를 지정한다.

입력된 문자열이 지정된 길이를 초과하는 경우, 지정된 길이에 맞도록 데이터를 자르므로(truncate) 주의한다.

또한, 고정 길이 문자열 타입인 **CHAR**에서는 선언한 길이에 고정되므로, 문자를 저장할 때 오른쪽에 공백 문자(trailing space)를 채운다. 한편, 가변 길이 문자열 타입인 **VARCHAR**에서는 공백 문자를 채우지 않고 실제 입력된 문자열만큼 저장한다.

VARCHAR 타입에서 지정할 수 있는 최대 길이는 1,073,741,823이며, **CHAR** 타입에서 지정할 수 있는 최대 길이는 268,435,455이다.

또한, **CSQL** 문장으로 한 번에 입력 또는 출력할 수 있는 최대 크기는 8192KB이다.

참고

9.0 미만 버전에서 **CHAR** 나 **VARCHAR** 타입의 길이는 문자의 개수가 아닌 문자의 바이트 크기를 나타내었다.

문자셋(Character Set, charset)

문자셋(문자 집합)은 특정 문자(symbol)를 컴퓨터에 저장할 때, 어떠한 코드로 인코딩할 것인지에 대한 규칙이 정의된 집합을 의미한다. CUBRID가 사용할 문자셋은 DB 생성 시, **CHARSET** 소개자 또는 **COLLATE** 수정자 사용 시 지정될 수 있다. 문자셋에 대한 자세한 설명은 [다국어 개요](#) 을 참고한다.

문자셋의 정렬(Collating Character Set)

콜레이션(collation)은 어느 문자셋이 설정된 상태에서 데이터베이스에 저장된 값들을 검색하거나 정렬하는 작업을 위해 문자들을 서로 비교할 때 사용하는 규칙들의 집합이다. 문자셋에 대한 자세한 설명은 [다국어 개요](#) 을 참고한다.

문자열 변환(Character String Coercion)

고정길이와 가변길이 문자열 사이에는 두 문자의 길이가 비교 가능할 수 있도록 자동 변환된다. 자동 변환은 동일한 문자셋에 속하는 문자열에만 적용된다.

예를 들어, 데이터 타입이 **CHAR** (5)인 칼럼을 추출하여 데이터 타입이 **CHAR** (10)인 칼럼에 삽입하는 경우 자동으로 데이터 타입이 **CHAR** (10)으로 변환되어 삽입된다. 문자열을 명시적으로 변환할 수도 있는데, 이 때에는 **CAST** 연산자를 사용한다([CAST\(\)](#) 참조).

문자열 압축(String compression)

가변 문자열 타입 값(VARCHAR(n))은 데이터베이스(힙 파일, 인덱스 파일 또는 목록 파일)에 저장하기 전에 압축할 수 있다(LZO1X 알고리즘 사용). 최소 255바이트 이상이면 압축이 시도된다(이 값은 미리 정의되어 있으며 변경할 수 없음). 압축이 효율적이지 않으면(압축 값의 크기 및 오버헤드가 압축 전의 원래 값과 동일하거나 큰 경우) 압축되지 않은 채로 값이 저장된다. 압축은 기본적으로 활성화되어 있으며 시스템 파라미터 `enable_string_compression` 를 설정하여 비활성화할 수 있다. 압축 오버헤드는 8바이트(압축 버퍼 크기 4바이트, 압축 해제 문자열 예상 크기 4바이트)이다. 압축된 문자열은 데이터베이스에서 읽을 때 압축이 풀린다. 데이터 값의 압축 여부를 파악하려면 `DISK_SIZE` 함수 결과를 인자가 동일한 `OCTET_LENGTH` 함수 결과와 비교한다. `DISK_SIZE` 값이 더 작으면(값 오버헤드 무시) 압축이 사용되었음을 나타낸다.

CHAR(n)

고정길이 문자열은 **CHAR** (n)로 표현하며, 여기서 n 은 문자의 개수를 나타낸다. n 이 생략되면 길이는 기본값인 1로 지정된다.

문자열의 길이가 n 을 초과하면 **allow_truncated_string** 설정 값이 **yes** 인 경우 초과 부분을 절삭하지만, 설정 값이 **no**인 경우 에러가 발생한다. n 보다 작은 문자열이 저장되면 나머지 부분은 공백 문자로 채워진다.

CHAR (n)와 **CHARACTER** (n)는 같은 의미로 사용된다.

참고

CUBRID 9.0 미만 버전에서는 n 이 문자의 개수가 아니라 바이트 길이를 나타낸다.

- n 은 1 이상 2048 이하의 바이트 수를 지정하며, 고정 길이 문자형으로 해당 크기만큼 공간을 확보한다.
- 공백 값은 빈 따옴표("")로 처리하며, 이 경우 **LENGTH** 함수의 리턴 값은 0이 아니라 **CHAR** (n)에서 정의한 고정길이이다. 즉, **CHAR** (10)인 칼럼에 공백 값을 넣더라도 리턴 값은 10이며, n 이 생략되면 기본값이 1 이므로 **CHAR** (1)로 간주된다.
- 채우는(padding) 문자로 사용되는 공백은 특수 문자를 비롯한 어느 문자보다도 작은 것으로 간주된다.

CHAR(12)에 'pacesetter'를 저장하면 'pacesetter '가 된다(10자리 문자열과 공백 문자 2개로 구성됨).
CHAR(10)에 'pacesetter'를 저장하면 'pacesetter'가 된다(10을 넘어서는 부분이 공백 문자이므로 이를 절삭하고 10자리 문자열로 구성됨. 단, ****allow_truncated_string**** 설정 값이 ****no****\인 경우 에러가 발생).
CHAR(4)에 'pacesetter'를 저장하면 'pace'가 된다(문자열의 크기가 4보다 크므로 절삭함. 단, ****allow_truncated_string**** 설정 값이 ****no****\인 경우 에러가 발생).
CHAR에 'p'를 저장하면 'p'가 된다(n 이 생략되면 길이는 기본값인 1로 지정됨).

- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

VARCHAR(n) 또는 **CHAR VARYING**(n)

가변길이 문자열은 **VARCHAR** (n)로 표현하며, 여기서 n 은 문자의 개수를 나타낸다. n 이 생략되면 길이는 최대 길이인 1,073,741,823로 지정된다.

문자열의 길이가 n 을 초과하면 **allow_truncated_string** 설정 값이 **yes**인 경우 초과 부분을 절삭하지만, 설정 값이 **no**인 경우 에러가 발생한다. n 보다 작은 문자열이 저장되면 **CHAR** (n)는 나머지 부분을 공백 문자로 채우지만 **VARCHAR** (n)에는 해당 문자열 길이만큼만 저장한다.

VARCHAR (n)와 **CHARACTER VARYING** (n), **CHAR VARYING** (n)은 같은 의미로 사용된다.

참고

CUBRID 9.0 미만 버전에서는 n 이 문자의 개수가 아니라 바이트 길이를 나타낸다.

- **STRING** 은 **VARCHAR** (최대 길이)와 같다.
- n 은 1부터 1,073,741,823(1G) 사이의 정수이다.
- 공백 값은 빈 따옴표("")로 처리하며, 이 경우 **LENGTH** 함수의 리턴 값은 0이다.

VARCHAR(4)에 'pacesetter'를 저장하면 'pace'가 된다(문자열의 크기가 4보다 크므로 절삭함. 단, ****allow_truncated_string**** 설정 값이 ****no****\인 경우 에러가 발생).

VARCHAR(12)에 'pacesetter'를 저장하면 'pacesetter'가 된다(10자리 문자열로 구성됨).

VARCHAR(12)에 'pacesetter '를 저장하면 'pacesetter '가 된다(10자리 문자열과 공백 문자 2개로 구성됨).

VARCHAR(10)에 'pacesetter'를 저장하면 'pacesetter'가 된다(10을 넘어서는 부분이 공백 문자이므로 이를 절삭하고 10자리 문자열로 구성됨. 단, ****allow_truncated_string**** 설정 값이 ****no****\인 경우 에러가 발생).

VARCHAR에 'p'를 저장하면 'p'가 된다(n 이 생략되면 최대 길이는 기본값인 1, 073, 741, 823로 지정되고, 저장 시 나머지 부분은 공백 문자로 채워지지 않음).

- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

STRING

STRING 은 가변길이 문자열 데이터 타입이다. **STRING** 은 **VARCHAR** 를 최대 길이로 지정한 것과 같다. 즉 **STRING** 은 **VARCHAR** (1,073,741,823)과 동일하다.

특수 문자 이스케이프

CUBRID는 특수 문자를 이스케이프(escape)하는 방법을 두 가지 지원한다. 하나는 따옴표를 이용한 방법이고, 다른 하나는 백슬래시(\)를 이용한 방법이다.

- 따옴표 이스케이프

cubrid.conf 의 시스템 파라미터 **ansi_quotes**가 no로 설정되어 있으면 문자열을 감쌀 때 큰따옴표(")와 작은따옴표(') 둘 다 사용할 수 있다. **ansi_quotes** 파라미터의 기본값은 **yes** 로, 문자열을 감쌀 때 작은따옴표만 사용할 수 있다.

- 작은따옴표로 감싼 문자열에 포함된 작은따옴표는 두 개의 작은따옴표("")를 쓴다.

- 큰따옴표로 감싼 문자열에 포함된 큰따옴표는 두 개의 큰따옴표(“”)를 쓴다. (**ansi_quotes** 값이 no인 경우)
- 큰따옴표로 감싼 문자열에 포함된 작은따옴표는 이스케이프하지 않아도 된다. (**ansi_quotes** 값이 no인 경우)
- 작은따옴표로 감싼 문자열에 포함된 큰따옴표는 이스케이프하지 않아도 된다.

· 백슬래시 이스케이프

백슬래시(\)를 이용한 이스케이프는 **cubrid.conf**의 시스템 파라미터 **no_backslash_escapes**를 no로 설정했을 때에만 사용할 수 있다. **no_backslash_escapes** 파라미터의 기본값은 **yes** 이다. **no_backslash_escapes**의 값이 no인 경우, 다음과 같은 특수 문자를 의미한다.

- \': 작은따옴표(')
- \": 큰따옴표(“)
- \n : 뉴라인(newline, linefeed) 문자
- \r : 캐리지 리턴(carriage return) 문자
- \t : 탭(tab) 문자
- \\ : 백슬래시(backslash)
- \% : 퍼센트 기호(%). 자세한 내용은 아래 설명을 참고한다.
- _ : 언더바(_). 자세한 내용은 아래 설명을 참고한다.

다른 모든 이스케이프에 대해서는 백슬래시가 무시된다. 예를 들어 “x”는 그냥 “x”라고 입력한 것과 같다.

\% 와 **_** 는 **LIKE** 와 같은 패턴 매칭 구문에서 퍼센트 기호와 언더바를 찾을 때 쓰이며, 백슬래시가 없으면 와일드카드 문자(wildcard character)로 쓰인다. 패턴 매칭 구문 밖에서는 와일드카드 문자가 아닌 일반 문자열 “\%”와 “_”로 그대로 쓰인다. 자세한 내용은 **LIKE** 을 참고한다.

다음은 **cubrid.conf**의 시스템 파라미터 **ansi_quotes**가 yes(기본값)이고 **no_backslash_escapes**가 no일 때 백슬래시에 대해 이스케이프를 수행한 결과이다.

```
-- ansi_quotes=yes, no_backslash_escapes=no
SELECT STRCMP('single quotes test(')', 'single quotes test(\')');
```

위의 구문을 실행하면, 백슬래시가 이스케이프 문자로 인식되므로 두 문자열은 같은 것으로 인식된다.

```
strcmp('single quotes test(')', 'single quotes test(')')
=====
0
```

```
SELECT
STRCMP('a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z',
'a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z');
```

위의 구문을 실행하면, 백슬래시가 이스케이프 문자로 인식되므로 두 문자열은 같은 것으로 인식된다.

```
strcmp('abcdefghijklm
s      uvwxyz', 'abcdefghijklm
s      uvwxyz')
=====
0
```

```
SELECT LENGTH('\');
```

위의 구문을 실행하면, 백슬래시가 이스케이프 문자로 인식되므로 문자열의 길이는 1이 된다.

```
char_length('\')
=====
1
```

다음은 **cubrid.conf**의 시스템 파라미터 **ansi_quotes**가 **yes**(기본값)이고 **no_backslash_escapes**가 **yes**(기본값)일 때 수행한 결과이다. 백슬래시는 일반 문자로 인식된다.

```
-- ansi_quotes=yes, no_backslash_escapes=yes

SELECT STRCMP('single quotes test(')', 'single quotes test(\')');
```

위의 구문을 실행하면, 아직 따옴표가 열린 것으로 간주하여 아래와 같은 예러가 발생한다. CSQL 인터프리터의 SQL 입력 화면에서 위의 구문을 입력하면 다음 따옴표가 입력되기를 대기한다.

```
ERROR: syntax error, unexpected UNTERMINATED_STRING, expecting
SELECT or VALUE or VALUES or '('
```

SELECT

```
STRCMP('a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z',
'a\b\c\d\efghijklm\nopq\r\s\tuvwxyz');
```

위의 구문을 실행하면, 백슬래시가 일반 문자로 인식되므로 두 문자열의 비교 결과는 서로 다른 것으로 인식된다.

```
      strcmp('a\b\c\d\e\f\g\h\i\j\k\l\m\n\o\p\q\r\s\t\u\v\w\x\y\z',
'a\b\c\d\efghijklm\nopq\r\s\tuvwxyz')
=====
=====
-1
```

```
SELECT LENGTH('\\');
```

위의 구문을 실행하면, 백슬래시가 일반 문자로 인식되므로 문자열의 길이는 2가 된다.

```
      char_length('\\')
=====
2
```

다음은 **cubrid.conf**의 시스템 파라미터 **ansi_quotes**가 yes이고 **no_backslash_escapes**가 no일 때 LIKE 절에 대해 이스케이프를 수행한 결과이다.

```
-- ansi_quotes=yes, no_backslash_escapes=no

CREATE TABLE t1 (a VARCHAR(200));
INSERT INTO t1 VALUES ('aaabbb'), ('aaa%');

SELECT a FROM t1 WHERE a LIKE 'aaa\%' ESCAPE '\\';
```

```
      a
=====
      'aaa%'
```

‘%’가 패턴 매칭 문자가 아닌 일반 문자로 인식되기 때문에, 질의를 수행하면 1개의 행만 리턴한다.

LIKE 절의 문자열에서는 백슬래시가 항상 일반 문자로 간주되기 때문에, ‘%’를 패턴 매칭 문자가 아닌 일반 문자로 인식시키려면 ESCAPE 절을 지정하여 백슬래시가 이스케이프 문자임을 명시해야 한다. ESCAPE 절에서는 백슬래시가 이스케이프 문자로 간주되기 때문에, 두 개의 백슬래시 문자를 사용했다.

위의 SELECT 질의에서 이스케이프 문자를 백슬래시가 아닌 다른 문자로 쓰려면 아래와 같이 사용할 수 있다.

```
SELECT a FROM t1 WHERE a LIKE 'aaa#%' ESCAPE '#';
```

비교 규칙

두 문자열 값을 비교할 때 후행 공백에 대한 비교 규칙은 다음과 같다.

- 후행 공백 무시
- 후행 공백 포함

후행 공백 무시

두 문자열 값이 모두 고정 길이 타입 (CHAR) 인 경우의 비교는 아래 예와 같이 후행 공백을 무시한다. 'abc'와 'abc ' 비교 결과는 "일치" 이다.

후행 공백 포함

두 문자열 값이 모두 가변 길이 타입 (VARCHAR) 인 경우의 비교는 아래 예와 같이 후행 공백을 무시하지 않는다. 'abc'를 'abc ' 비교 결과는 'abc '가 'abc'보다 "크다" 이다.

예외

두 문자열 값을 비교할 때 하나는 고정형이고 다른 하나는 변수형이면 CUBRID는 후행 공백 포함 규칙을 따른다.

ENUM 데이터 타입

ENUM 타입은 열거형 문자열 상수들의 중복 없는 순서 집합으로 구성되어 있는 타입이다. 열거형 칼럼을 생성하는 구문은 다음과 같다.

```
<enum_type> ::=
    ENUM '(' <char_string_literal_list> ')'

<char_string_literal_list> ::=
    <char_string_literal_list> ',' CHAR_STRING |
    CHAR_STRING
```

다음은 **ENUM** 칼럼을 정의하는 예이다.

```
CREATE TABLE tbl (
  color ENUM ('red', 'yellow', 'blue', 'green')
);
```

- 이 타입의 칼럼에 **DEFAULT** 속성이 지정될 수 있다.

색인은 원소들이 열거형 타입에 정의된 순서에 따라 각 원소에 연관되어 있다. 예를 들어, *color* 칼럼 (NULL 값을 허용한다고 가정)은 다음 값 중 하나를 가질 수 있다.

값	색인 번호
NULL	NULL
'red'	1
'yellow'	2
'blue'	3
'green'	4

ENUM 타입 값의 집합은 512 개보다 많은 원소를 가질 수 없으며, 집합의 각 원소는 고유해야 한다. 각 **ENUM** 타입 값에 대해 해당 색인만 저장하기 때문에, **ENUM** 타입은 2 바이트의 저장 공간을 사용한다. 문자열 값이 아니라 해당 색인만 저장함으로써 저장 공간을 줄일 수 있으며 이로 인한 성능 이점도 얻을 수 있다.

ENUM 타입을 사용할 때 열거형 값과 값의 색인 둘 다 활용할 수 있다. 예를 들어, **ENUM** 타입 칼럼에 값을 삽입할 때 사용자는 **ENUM** 타입의 값 또는 색인을 사용할 수 있다.

```
-- insert enum element 'yellow' with index 2
INSERT INTO tbl (color) VALUES ('yellow');
-- insert enum element 'red' with index 1
INSERT INTO tbl (color) VALUES (1);
```

표현식에서 사용될 때 **ENUM** 타입은 문맥에 따라 **CHAR** 타입 또는 숫자로 동작한다.


```
-- the first result column has ENUM type, the second has INTEGER
type and the third has VARCHAR type
SELECT color, color + 0, CONCAT(color, '') FROM tbl;
```

```
color                color+0  concat(color, '')
=====
'yellow'              2  'yellow'
'red'                 1  'red'
```

문맥 상 CHAR나 숫자 타입이 아닌 다른 타입으로 사용될 때, 열거형은 색인 또는 열거형 값을 사용하는 해당 타입으로 변환된다. 아래 표는 타입 변환 시 **ENUM** 타입의 어떤 부분이 사용되는지를 보여준다.

타입	값(색인/값)
SHORT	색인
INTEGER	색인
BIGINT	색인
FLOAT	색인
DOUBLE	색인
NUMERIC	색인
TIME	값
DATE	값
DATETIME	값
TIMESTAMP	값

타입	값(색인/값)
CHAR	값
VARCHAR	값
BIT	값
VARBIT	값

ENUM 타입 비교

= 또는 **IN** 연산자가 (<enum_칼럼> <연산자> <상수>) 형식으로 주어진다면, 시스템은 상수를 **ENUM** 타입으로 변환하려고 한다. 변환에 실패하면 오류를 반환하지 않고 비교 결과를 FALSE로 반환하는데, 이와 같은 동작은 이러한 두 개의 연산자에 대한 인덱스 스캔 질의 계획을 허용하기 위해서이다.

다른 모든 **비교 연산자**에 대해 **ENUM** 타입은 다른 피연산자의 타입으로 변환된다. 두 개의 **ENUM** 타입 간에 비교 연산이 수행되면 두 피연산자 모두 **CHAR** 타입으로 변환되며 **CHAR** 타입의 규칙을 따른다. =과 **IN** 조건을 제외한 **ENUM** 칼럼을 포함한 나머지 조건은 인덱스 스캔의 고려 대상이 아니다.

이러한 규칙을 이해하기 위해 다음의 예를 살펴보자.

```
CREATE TABLE tbl (
  color ENUM ('red', 'yellow', 'blue', 'green')
);

INSERT INTO tbl (color) VALUES (1), (2), (3), (4);
```

다음 질의는 상수 'red'를 색인 번호 1을 가진 열거형 값 'red'로 변환한다.

```
SELECT color FROM tbl WHERE color = 'red';
```

```
color
=====
'red'
```

```
SELECT color FROM tbl WHERE color = 1;
```

```
color
=====
'red'
```

다음 질의는 변환 오류를 반환하지는 않지만 어떠한 결과도 반환하지 않는다.

```
SELECT color FROM tbl WHERE color = date'2010-01-01';
SELECT color FROM tbl WHERE color = 15;
SELECT color FROM tbl WHERE color = 'asdf';
```

다음 질의에서 **ENUM** 타입은 다른 피연산자의 타입으로 변환된다.

```
-- CHAR comparison using the enum value
SELECT color FROM tbl WHERE color < 'pink';
```

```
color
=====
'blue'
'green'
```

```
-- INTEGER comparison using the enum index
SELECT color FROM tbl WHERE color > 3;
```

```
color
=====
'green'
```

```
-- Conversion error
SELECT color FROM tbl WHERE color > date'2012-01-01';
```

```
ERROR: Cannot coerce value of domain "enum" to domain "date".
```

ENUM 타입 정렬

ENUM 타입 값은 열거형 값이 아니라 값의 색인에 의해 정렬된다. **ENUM** 타입 칼럼을 정의할 때는 열거되는 값의 순서도 함께 고려해야 한다.

```
SELECT color FROM tbl ORDER BY color ASC;
```

```

color
=====
'red'
'yellow'
'blue'
'green'

```

ENUM 타입 칼럼으로 저장된 값을 **CHAR** 값으로 정렬하려면 열거형 값을 **CHAR** 타입으로 변환(CAST)하면 된다.

```
SELECT color FROM tbl ORDER BY CAST (color AS VARCHAR) ASC;
```

```

color
=====
'blue'
'green'
'red'
'yellow'

```

참고 사항

ENUM 타입은 재사용 가능한 타입이 아니다. 여러 개의 칼럼이 같은 **ENUM** 집합의 값을 요구한다 하더라도, 각각의 칼럼에 대해 **ENUM** 타입이 정의되어야 한다. **ENUM** 타입의 두 칼럼을 비교하는 경우 비록 두 **ENUM** 타입이 같은 집합 값을 정의했다고 하더라도, 두 칼럼이 **CHAR** 타입으로 변환된 것처럼 동작한다.

ALTER ... CHANGE 문을 사용하여 **ENUM** 타입 값의 집합을 수정하려면 **alter_table_change_type_strict** 파라미터의 값이 반드시 yes여야 한다. 이 경우 시스템은 새로운 타입으로 변환한 열거형 값(문자열 리터럴)을 사용한다. 기존 값이 새로운 **ENUM** 타입 값의 집합을 벗어나면 자동으로 공백 문자열("")로 매핑된다.

```

CREATE TABLE tbl(color ENUM ('red', 'green', 'blue'));
INSERT INTO tbl VALUES('red'), ('green'), ('blue');

```

다음 문장은 **ENUM** 타입이 'yellow' 값을 가지도록 변경한다.

```

ALTER TABLE tbl CHANGE color color ENUM ('red', 'green', 'blue',
'yellow');
INSERT into tbl VALUES(4);
SELECT color FROM tbl;

```

```

color
=====
'red'
'green'
'blue'
'yellow'

```

다음의 예에서 'green' 값이 새로운 **ENUM** 타입 정의에 매핑되지 않기 때문에, 모든 튜플의 'green'이 "로 변경된다.

```

ALTER TABLE tbl CHANGE color color ENUM ('red', 'yellow', 'blue');
SELECT color FROM tbl;

```

```

color
=====
'red'
''
'blue'
'yellow'

```

ENUM 타입은 CUBRID 드라이버에서 문자열로 매핑된다. 다음은 JDBC 응용에서 **ENUM** 타입을 사용하는 예이다.

```

Statement stmt = connection.createStatement("SELECT color FROM tbl");
ResultSet rs = stmt.executeQuery();

while(rs.next()){
    System.out.println(rs.getString());
}

```

다음은 CCI 응용에서 **ENUM** 타입을 사용하는 예이다.

```

req_id = cci_prepare (conn, "SELECT color FROM tbl", 0, &err);
error = cci_execute (req_id, 0, 0, &err);
if (error < CCI_ER_NO_ERROR)
{
    /* handle error */
}

error = cci_cursor (req_id, 1, CCI_CURSOR_CURRENT, &err);
if (error < CCI_ER_NO_ERROR)
{
    /* handle error */
}

```

```
error = cci_fetch (req_id, &err);
if (error < CCI_ER_NO_ERROR)
{
    /* handle error */
}

cci_get_data (req, idx, CCI_A_TYPE_STR, &data, 1);
```

BLOB/CLOB 데이터 타입

External LOB(Large Object) 타입은 텍스트 또는 이미지 등 크기가 큰 객체를 처리하기 위한 데이터 타입이다. **LOB** 타입 데이터가 생성 및 삽입되면 이는 외부 저장소에 파일로 저장되고 CUBRID 데이터베이스에는 해당 파일의 위치 정보(**LOB** locator)가 저장된다. 데이터베이스에서 해당 데이터(**LOB** locator)가 삭제되면, 외부 저장소에 저장된 해당 파일이 함께 삭제된다. CUBRID는 두 가지 **LOB** 타입을 지원한다.

- Binary Large Object(**BLOB**)
- Character Large Object(**CLOB**)

참고

관련 용어

- **LOB** (Large Object): 이진 바이너리 또는 텍스트 등 크기가 큰 객체이다.
- **FBO** (File Based Object): DB 데이터를 DB 외부의 파일로 저장하는 객체이다.
- **External LOB**: LOB 데이터를 DB 외부에 파일로 저장하는 객체로서 FBO라고도 하며, CUBRID는 이를 지원한다. Internal LOB은 **LOB** 데이터를 DB 내부에 저장하는 객체이다.
- **External Storage**: LOB을 저장하는 외부 저장소이다(예: POSIX 파일 시스템).
- **LOB Locator**: 외부 저장소에 저장된 파일의 경로명이다.
- **LOB Data**: LOB Locator에 명시된 위치에 있는 파일의 내용이다.

LOB 데이터는 외부 저장소에 다음과 같은 파일명으로 저장된다.

```
{table_name}_{unique_name}
```

- *table_name* : prefix로 삽입되어 하나의 외부 저장소에 여러 테이블의 **LOB** 데이터를 저장할 수 있다.

· *unique_name* : 데이터베이스 서버가 임의로 생성하는 이름이다.

LOB 데이터의 저장소는 데이터베이스 서버 상의 로컬 파일 시스템이다. **LOB** 데이터는 **cubrid createdb** 유틸리티의 **-lob-base-path** 옵션 값으로 지정된 경로에 저장되며, 옵션이 생략될 경우 데이터베이스 볼륨이 생성되는 [db-vol path]/lob 경로에 저장된다. 보다 자세한 내용은 **createdb** 및 **저장소 생성 및 관리**를 참고한다.

CUBRID가 제공하는 API나 도구를 사용하지 않고 사용자가 임의로 **LOB** 파일 내용을 수정하면, 해당 내용의 일치성이 보장되지 않으므로 주의한다.

데이터베이스 위치 정보 파일(**databases.txt**)에 **LOB** 데이터 파일 경로가 등록되어 있음에도 불구하고 해당 경로가 삭제된 경우, 데이터베이스 서버(**cub_server**) 및 독립 모드(standalone)로 동작하는 유틸리티가 정상적으로 실행되지 않으므로 주의한다.

BLOB

바이너리 데이터를 DB 외부에 저장하기 위한 타입으로, **BLOB** 데이터의 최대 길이는 외부 저장소에서 생성 가능한 파일 크기이다. **BLOB** 타입은 SQL 문에서 비트열 타입으로 입출력 값을 표현한다. 즉, **BIT** (n), **BIT VARYING** (n) 타입과 호환되며, 명시적 타입 변환만 허용된다. 데이터 길이가 서로 다른 경우에는 최대 길이가 작은 타입에 맞추어 절삭(truncate)된다. **BLOB** 타입 값을 바이너리 값으로 변환하는 경우, 변환된 데이터는 최대 1GB를 넘을 수 없다. 반대로 바이너리를 **BLOB** 타입으로 변환하는 경우, 변환된 데이터는 **BLOB** 저장소에서 제공하는 최대 파일 크기를 넘을 수 없다.

CLOB

문자열 데이터를 DB 외부에 저장하기 위한 타입으로, **CLOB** 데이터의 최대 길이는 외부 저장소에서 생성 가능한 파일 크기이다. **CLOB** 타입은 SQL 문에서 문자열 타입으로 입출력 값을 표현한다. 즉, **CHAR** (n), **VARCHAR** (n) 타입과 호환된다. 단, 명시적 타입 변환만 허용되며, 데이터 길이가 서로 다른 경우에는 최대 길이가 작은 타입에 맞추어 절삭(truncate)된다. **CLOB** 타입 값을 문자열 값으로 변환하는 경우, 변환된 데이터는 최대 1GB를 넘을 수 없다. 반대로 문자열을 **CLOB** 타입으로 변환하는 경우, 변환된 데이터는 **CLOB** 저장소에서 제공하는 최대 파일 크기를 넘을 수 없다.

정의 및 변경

CREATE TABLE 문 또는 **ALTER TABLE** 문을 사용하여 **BLOB** / **CLOB** 타입 칼럼을 생성/추가/삭제할 수 있다.

- **LOB** 타입 칼럼에 대해서는 인덱스를 생성할 수 없다.
- **LOB** 타입 칼럼에 대해서는 **PRIMARY KEY**, **FOREIGN KEY**, **UNIQUE**, **NOT NULL** 제약 조건을 정의할 수 없다. 또한, **SHARED** 속성을 정의할 수 없으며, **DEFAULT** 속성은 **NULL** 값에 대해서만 정의할 수 있다.
- **LOB** 타입 칼럼/데이터는 컬렉션 타입의 원소가 될 수 없다.

- **LOB** 타입 칼럼이 있는 레코드를 삭제하는 경우, **LOB** 칼럼 값(Locator) 및 외부 저장소 내 파일을 모두 삭제한다. 또한, 기본 키 테이블에서 **LOB** 타입 칼럼이 있는 레코드가 삭제됨에 따라 이를 참조하는 외래 키 테이블의 레코드가 함께 삭제되는 경우, **LOB** 칼럼 값(Locator) 및 외부 저장소 내 **LOB** 파일을 모두 삭제한다. 단, **ALTER TABLE ... DROP** 문을 사용하여 **LOB** 칼럼을 삭제하거나 **DROP TABLE** 문을 사용하여 해당 테이블을 삭제하는 경우, **LOB** 칼럼 값(Lob locator)만 삭제하고 **LOB** 칼럼이 참조하는 외부 저장소 내 **LOB** 파일은 삭제하지 않는다.

```
-- creating a table and CLOB column
CREATE TABLE doc_t (doc_id VARCHAR(64) PRIMARY KEY, content CLOB);

-- an error occurs when UNIQUE constraint is defined on CLOB column
ALTER TABLE doc_t ADD CONSTRAINT content_unique UNIQUE(content);

-- an error occurs when creating an index on CLOB column
CREATE INDEX i_doc_t_content ON doc_t (content);

-- creating a table and BLOB column
CREATE TABLE image_t (image_id VARCHAR(36) PRIMARY KEY, doc_id
VARCHAR(64) NOT NULL, image BLOB);

-- an error occurs when adding a BLOB column with NOT NULL constraint
ALTER TABLE image_t ADD COLUMN thumbnail BLOB NOT NULL;

-- an error occurs when adding a BLOB column with DEFAULT attribute
ALTER TABLE image_t ADD COLUMN thumbnail2 BLOB DEFAULT
BIT_TO_BLOB(X'010101');
```

저장 및 변경

BLOB / **CLOB** 타입 칼럼에는 각각 **BLOB** / **CLOB** 타입 값이 저장되며, 바이너리 또는 문자열 데이터를 입력하는 경우에는 각각 **BIT_TO_BLOB()**, **CHAR_TO_CLOB()** 함수를 사용하여 명시적으로 타입을 변환을 수행하여야 한다.

INSERT 문을 사용하여 **LOB** 칼럼에 값을 입력하면, 내부적으로는 외부 저장소에 파일을 생성하여 해당 데이터를 저장하고, 실제 칼럼 값으로 해당 파일의 경로(Locator) 정보를 저장한다.

DELETE 문을 사용하여 **LOB** 칼럼이 존재하는 레코드를 삭제하면, 해당 **LOB** 칼럼 값이 참조하는 파일을 함께 삭제한다.

UPDATE 문을 사용하여 **LOB** 칼럼 값을 변경하는 경우, 새로운 값이 **NULL** 인지 여부에 따라 다음과 같이 동작하면서 칼럼 값을 변경한다.

- **LOB** 타입 칼럼 값을 **NULL** 이 아닌 값으로 변경하는 경우: **LOB** 칼럼에 이미 외부 파일을 참조하는 Locator 가 저장되어 있다면, 해당 파일을 삭제한다. 그리고 새로운 파일을 생성하여 **NULL**이 아닌 값을 저장한 후, **LOB** 칼럼 값에 새로운 파일에 대한 locator를 저장한다.

- **LOB** 타입 칼럼 값을 **NULL** 로 변경하는 경우: LOB 칼럼에 이미 외부 파일을 참조하는 Locator 가 저장되어 있다면, 해당 파일을 삭제한다. 그리고 **LOB** 칼럼 값에 **NULL**을 바로 저장한다.

```
-- inserting data after explicit type conversion into CLOB type
column
INSERT INTO doc_t (doc_id, content) VALUES ('doc-1',
CHAR_TO_CLOB('This is a Dog'));
INSERT INTO doc_t (doc_id, content) VALUES ('doc-2',
CHAR_TO_CLOB('This is a Cat'));

-- inserting data after explicit type conversion into BLOB type
column
INSERT INTO image_t VALUES ('image-0', 'doc-0',
BIT_TO_BLOB(X'000001'));
INSERT INTO image_t VALUES ('image-1', 'doc-1',
BIT_TO_BLOB(X'000010'));
INSERT INTO image_t VALUES ('image-2', 'doc-2',
BIT_TO_BLOB(X'000100'));

-- inserting data from a sub-query result
INSERT INTO image_t SELECT 'image-1010', 'doc-1010', image FROM
image_t WHERE image_id = 'image-0';

-- updating CLOB column value to NULL
UPDATE doc_t SET content = NULL WHERE doc_id = 'doc-1';

-- updating CLOB column value
UPDATE doc_t SET content = CHAR_TO_CLOB('This is a Dog') WHERE
doc_id = 'doc-1';

-- updating BLOB column value
UPDATE image_t SET image = (SELECT image FROM image_t WHERE image_id
= 'image-0') WHERE image_id = 'image-1';

-- deleting BLOB column value and its referencing files
DELETE FROM image_t WHERE image_id = 'image-1010';
```

조회

LOB 타입 칼럼을 조회하면 칼럼이 참조하는 파일에 저장된 데이터를 출력한다. `CAST()` 연산자, `CLOB_TO_CHAR()` 함수, `BLOB_TO_BIT()` 함수를 사용하여 명시적 타입 변환을 수행할 수 있다.

- CSQL에서 질의를 실행할 경우, 파일에 저장된 데이터가 아닌 칼럼 값(Locator)을 출력한다. **BLOB** / **CLOB** 칼럼이 참조하는 데이터를 출력하기 위해서는 `CLOB_TO_CHAR()` 함수를 사용하여 문자열로 변환해야 한다.
- 문자열 처리 함수를 사용하기 위해서는 `CLOB_TO_CHAR()` 함수를 사용하여 문자열로 변환해야 한다.
- **GROUP BY** 절, **ORDER BY** 절에 **LOB** 칼럼을 명시할 수 없다.

- 비교 연산자, 관계 연산자, **IN**, **NOT IN** 연산자를 사용하여 **LOB** 칼럼을 비교할 수 없다. 단, **IS NULL** 조건식을 사용하여 **LOB** 칼럼 값(Locator)이 **NULL** 인지 비교할 수 있다. 즉, 칼럼 값이 **NULL**이면 **TRUE**를 반환하는데, 칼럼 값이 **NULL**인 경우는 **LOB** 데이터를 저장하는 파일이 존재하지 않는다는 의미이다.
- **LOB** 칼럼을 생성하고 데이터를 입력한 이후 **LOB** 데이터 파일을 삭제하면, **LOB** 칼럼 값(Locator)이 유효하지 않은 파일을 참조하는 상태가 된다. 이처럼 **LOB** locator와 **LOB** 데이터 파일이 매칭되지 않는 칼럼에 대해 `CLOB_TO_CHAR()`, `BLOB_TO_BIT()`, `CLOB_LENGTH()`, `BLOB_LENGTH()` 함수를 사용하면 **NULL**을 출력한다.

```
-- displaying locator value when selecting CLOB and BLOB column in
CSQL interpreter
SELECT doc_t.doc_id, content, image FROM doc_t, image_t WHERE
doc_t.doc_id = image_t.doc_id;
```

```
doc_id          content          image
=====
'doc-1'         file:ces_658/doc_t.
00001282208855807171_7329  file:ces_318/image_t.
00001282208855809474_7474
'doc-2'         file:ces_180/doc_t.
00001282208854194135_5598  file:ces_519/image_t.
00001282208854205773_1215

2 rows selected.
```

```
-- using string functions after coercing its type by CLOB_TO_CHAR( )
SELECT CLOB_TO_CHAR(content), SUBSTRING(CLOB_TO_CHAR(content), 10)
FROM doc_t;
```

```
clob_to_char(content)  substring( clob_to_char(content) from 10)
=====
'This is a Dog'       ' Dog'
'This is a Cat'       ' Cat'

2 rows selected.
```

```
SELECT CLOB_TO_CHAR(content) FROM doc_t WHERE CLOB_TO_CHAR(content)
LIKE '%Dog%';
```

```
clob_to_char(content)
=====
'This is a Dog'
```

```
SELECT CLOB_TO_CHAR(content) FROM doc_t ORDER BY
CLOB_TO_CHAR(content);
```

```
clob_to_char(content)
=====
'This is a Cat'
'This is a Dog'
```

```
SELECT * FROM doc_t WHERE content LIKE 'This%';
```

doc_id	content
'doc-1'	file:ces_004/doc_t.00001366272829040346_0773
'doc-2'	file:ces_256/doc_t.00001366272815153996_1229

```
-- an error occurs when LOB column specified in ORDER BY/GROUP BY
clauses
```

```
SELECT * FROM doc_t ORDER BY content;
```

```
ERROR: doc_t.content can not be an ORDER BY column
```

참고

- CUBRID 11.4에서는 LOB 파일 경로 정보를 나타내는 **locator** 가 이전 버전의 절대 경로에서 databases.txt에 정의된 lob-base-path 디렉터리를 기준으로 한 **상대 경로**로 변경되었다. 이를 통해 경로 관리가 더 유연해지고, 데이터베이스의 이동 및 백업/복원 과정에서도 LOB 파일의 경로 문제가 줄어들게 되었다.
- **lob-base-path** 설정을 통해 LOB 파일이 저장될 디렉터리를 정의하며, 모든 LOB locator는 이 디렉터리를 기준으로 계산된 **상대 경로**로 관리된다.
- databases.txt에 정의된 '**lob-base-path**' 경로가 /home1/data1 인 경우 CUBRID 버전별 locator의 예는 아래와 같다.
 - **CUBRID 11.4 (상대 경로):** file:ces_004/doc_t.00001366272829040346_0773

◦ CUBRID 11.3 이전 (절대 경로): `file:/home1/data1/ces_004/doc_t.00001366272829040346_0773`

- locator가 상대 경로로 저장되기 때문에 databases.txt의 '**lob-base-path**' 를 변경하는 경우
- 이전에 생성된 LOB 데이터는 조회가 불가능하다.
- 이전에 생성된 LOB 데이터를 조회하기 위해서는 해당 LOB 파일들을 변경된 '**lob-base-path**' 디렉터리로 이동하여야 한다.

```
# 예 :
% cp -r /home1/data1/* /home2/data2
```

- CUBRID 11.3 이전에서 절대 경로 locator로 저장된 LOB 데이터는 CUBRID 11.4에서 별도 변경 없이 조회 가능하다.
- 데이터베이스 이동 및 복원 시의 이점 : 데이터베이스를 다른 위치로 이동하더라도 **lob-base-path** 를 기반으로 경로를 관리하므로 LOB 파일 경로를 수동으로 수정할 필요가 없어졌다.
- 이 변경 사항은 CUBRID 11.4의 중요한 개선 중 하나이며, 데이터베이스 관리 효율성을 크게 향상시켰다.

연산자와 함수

CAST() 연산자를 사용하여 **BLOB** / **CLOB** 타입과 바이너리 타입/문자열 타입 간 명시적 타입 변환을 수행할 수 있다. 자세한 내용은 **CAST()** 연산자를 참고한다.

```
CAST (<bit_type_column_or_value> AS { BLOB | CLOB })
CAST (<char_type_column_or_value> AS { BLOB | CLOB })
```

다음은 **BLOB** / **CLOB** 타입 처리 및 변환을 위해 제공하는 함수이다. 자세한 설명은 **LOB 함수** 절을 참고한다.

- **CLOB_TO_CHAR()**
- **BLOB_TO_BIT()**
- **CHAR_TO_CLOB()**
- **BIT_TO_BLOB()**
- **CHAR_TO_BLOB()**
- **CLOB_FROM_FILE()**

- `BLOB_FROM_FILE()`
- `CLOB_LENGTH()`
- `BLOB_LENGTH()`

참고

"<blob_or_clob_column> IS NULL ": IS NULL 조건식을 사용하여 **LOB** 칼럼 값(Locator)이 **NULL** 인지 비교하고, **NULL** 이면 **TRUE** 를 반환한다.

저장소 생성 및 관리

LOB 데이터 파일은 기본적으로 데이터베이스 볼륨이 생성되는 <db-volume-path>/lob 디렉터리에 저장된다. 데이터베이스 생성 시 `createdb -B` 옵션을 통해서 **LOB** 데이터 파일을 저장할 디렉터리를 지정할 수 있다. 지정된 디렉터리가 존재하지 않으면 디렉터리 생성을 시도하며, 생성 실패 시에는 에러를 출력한다. 자세한 내용은 `createdb -B` 옵션을 참고한다.

```
# 현재 작업 디렉터리에 image_db 볼륨이 생성되고 LOB 데이터 파일이 저장된다.
% cubrid createdb image_db en_US

# 로컬 파일 시스템 내 "/home1/data1" 경로에 LOB 데이터 파일이 저장된다.
% cubrid createdb --lob-base-path="file:/home1/data1" image_db en_US
```

spacedb 유틸리티를 실행하여 LOB 파일이 저장되는 디렉터리를 확인할 수 있다.

```
% cubrid spacedb image_db

Space description for database 'image_db' with pagesize 16.0K. (log
pagesize: 16.0K)

Valid Purpose total_size free_size Vol Name
      0  GENERIC      512.0M    510.1M  /home1/data1/image_db

Space description for temporary volumes for database 'image_db' with
pagesize 16.0K.

Valid Purpose total_size free_size Vol Name
LOB space description file:/home1/data1
```

파일 저장소를 추가로 생성하려면 디스크 공간을 확보한 후 **databases.txt**의 **lob-base-path**를 증설한 디스크 위치로 변경한다. **databases.txt**의 변경 내용을 반영하기 위하여 DB 서버를 재구동한다. 단, **databases.txt**의 **lob-base-path**를 변경하더라도 예전 저장소에 저장된 **LOB** 데이터는 접근 가능하다.

```
# You can change to a new directory from the lob-base-path of
databases.txt file.
% cat $CUBRID_DATABASES/databases.txt

#db-name      vol-path          db-host      log-path
lob-base-path
image_db      /home1/data1    localhost    /home1/data1
file:/home1/data2
```

LOB 타입 칼럼의 데이터 파일에 대한 백업 및 복구는 지원하지 않으며, **LOB** 타입 칼럼의 메타 데이터 값(Locator)에 대해서만 백업 및 복구를 지원한다.

copydb 유틸리티를 사용하여 데이터베이스를 복사하는 경우, 관련 옵션이 지정되지 않으면 **LOB** 파일 디렉터리 경로가 복사되지 않으므로 추가로 **databases.txt** 파일을 설정해야 한다. 자세한 내용은 **copydb** 유틸리티의 `copydb -B` 및 `copydb --copy-lob-path` 옵션을 참고한다.

트랜잭션 지원 및 복구

LOB 데이터 변경에 대한 커밋/롤백을 지원한다. 즉, 트랜잭션 내에서 **LOB** Locator와 실제 **LOB** 데이터의 매핑(mapping)에 대한 유효성을 보장하고, 데이터베이스 장애 시 회복(recovery)을 지원한다. 트랜잭션 수행 도중 데이터베이스가 종료되어 해당 트랜잭션이 롤백 처리됨에 따라 **LOB** Locator와 **LOB** 데이터 간 매핑이 일치하지 않으면 에러를 출력한다. 아래의 예를 참고한다.

```
-- csql> ;AUTOCOMMIT OFF
```

```
CREATE TABLE doc_t (doc_id VARCHAR(64) PRIMARY KEY, content CLOB);
INSERT INTO doc_t VALUES ('doc-10', CHAR_TO_CLOB('This is content'));
COMMIT;
UPDATE doc_t SET content = CHAR_TO_CLOB('This is content 2') WHERE
doc_id = 'doc-10';
ROLLBACK;
SELECT doc_id, CLOB_TO_CHAR(content) FROM doc_t WHERE doc_id =
'doc-10';
```

```
doc_id    content
=====
'doc-10'  'This is content'
```

```
-- csql> ;AUTOCOMMIT OFF

INSERT INTO doc_t VALUES ('doc-11', CHAR_TO_CLOB ('This is
content'));
COMMIT;
UPDATE doc_t SET content = CHAR_TO_CLOB('This is content 3') WHERE
doc_id = 'doc-11';

-- system crash occurred and then restart server
SELECT doc_id, CLOB_TO_CHAR(content) FROM doc_t WHERE doc_id =
'doc-11';
```

```
-- Error : LOB Locator references to the previous LOB data because
only LOB Locator is rollbacked.
```

참고

- JDBC와 같은 드라이버를 통해 애플리케이션에서 **LOB** 데이터를 조회하는 경우, JDBC는 데이터베이스 서버로부터 ResultSet을 가져온 후 ResultSet에 대해 커서(cursor) 위치를 변경하면서 레코드를 인출할 수 있다. 즉, ResultSet을 가져온 시점에는 **LOB** 칼럼의 메타 데이터인 Locator만 저장되어 있고, 레코드를 인출하는 시점에 Locator가 참조하는 파일로부터 **LOB** 데이터가 실제로 인출된다. 따라서, 두 시점 사이에 **LOB** 데이터가 업데이트되는 경우, **LOB** Locator와 실제 **LOB** 데이터의 매핑(mapping)이 유효하지 않아 에러가 발생할 수 있다.
- **LOB** 타입 칼럼의 메타 데이터(Locator)에 대해서만 백업/복구를 지원한다. 따라서, 장애가 발생해서 특정 시점으로 복구할 때 **LOB** Locator와 **LOB** 데이터의 매핑이 유효하지 않아 에러가 발생할 수 있다.
- **LOB** 데이터를 다른 장비에 **INSERT** 하려면 반드시 **LOB** 칼럼 메타 데이터(Locator)가 참조하는 **LOB** 데이터를 읽어서 **INSERT** 해야 한다.
- CUBRID HA에서 **LOB** 칼럼 메타 데이터(Locator)는 복제되고, **LOB** 데이터는 복제되지 않는다. 따라서 **LOB** 타입 저장소가 로컬에 위치할 경우, 슬레이브 노드 또는 failover 이후 마스터 노드에서 해당 칼럼에 대한 작업을 허용하지 않는다.

경고

CUBRID 2008 R3.0 이하 버전에서는 **glo** (Generalized Large Object) 클래스를 사용하여 Large Object를 처리했으나, CUBRID 2008 R3.1 버전부터는 **glo** 클래스를 제거하고 **BLOB** / **CLOB** 데이

터 타입을 지원한다. 따라서, 2008 R3.0 미만 버전의 **glo** 클래스를 사용하는 환경에서는 CUBRID 버전 업그레이드를 수행할 때 DB 스키마 및 애플리케이션을 수정해야 한다.

컬렉션 데이터 타입

여러 개의 데이터 값을 하나의 속성에 저장할 수 있도록 하는 컬렉션 데이터 타입은 관계형 데이터 베이스의 확장된 기능이다. 컬렉션의 각 원소는 하나의 데이터 타입이어야 하며, 테이블(단, 뷰는 제외) 타입이 될 수도 있다. BLOB, CLOB 타입을 제외한 나머지 타입들은 컬렉션 타입의 원소가 될 수 있다.

타입	설명	타입 정의	입력 데이터	저장 데이터
SET	중복을 허용하지 않는 합집합	col_name SET VARCHAR(20) 또는 col_name SET (VARCHAR(20))	{'c','c','c','b','b','a'}	{'a','b','c'}
MULTISET	중복을 허용하는 합집합	col_name MULTISET VARCHAR(20) 또는 col_name MULTISET (VARCHAR(20))	{'c','c','c','b','b','a'}	{'a','b','b','c','c','c'}
LIST SEQUENCE	또는 중복을 허용하고, 데이터 입력 순서대로 저장하는 합집합	col_name LIST VARCHAR(20) 또는 col_name LIST (VARCHAR(20))	{'c','c','c','b','b','a'}	{'c','c','c','b','b','a'}

위의 표와 같이, 컬렉션 타입으로 지정되는 값은 중괄호('{, }') 안에 각 값들을 쉼표(,)로 구분하여 나열할 수 있다.

컬렉션 타입은 지정된 타입이 같다면 **CAST** 연산자를 이용하여 명시적으로 타입 변환이 가능하다. 아래 표는 명시적 변환이 가능한 컬렉션 타입에 관한 것이다.

FROM \ TO	SET	MULTISET	LIST
SET	-	Yes	Yes
MULTISET	Yes	-	No
LIST	Yes	Yes	-

컬렉션 타입은 콜레이션을 지원하지 않는다. 따라서 다음과 같은 질의는 오류를 반환한다.

```
CREATE TABLE tbl (str SET (string) COLLATE utf8_en_ci);
```

```
Syntax error: unexpected 'COLLATE', expecting ',' or ')'
```

SET

SET 는 각 원소가 서로 다른 값을 갖는 집합이다. **SET** 의 원소는 하나의 데이터 타입을 가져야 하고, 다른 테이블의 레코드를 가질 수도 있다.

```
CREATE TABLE set_tbl (col_1 SET (CHAR(1)));
INSERT INTO set_tbl VALUES ({'c','c','c','b','b','a'});
INSERT INTO set_tbl VALUES ({NULL});
INSERT INTO set_tbl VALUES ({''});
SELECT * FROM set_tbl;
```

```
col_1
=====
{'a', 'b', 'c'}
{NULL}
{' '}
```

```
SELECT CAST (col_1 AS MULTISET), CAST (col_1 AS LIST) FROM set_tbl;
```

```
cast(col_1 as multiset)  cast(col_1 as sequence)
=====
{'a', 'b', 'c'}  {'a', 'b', 'c'}
```

```
{NULL} {NULL}
{' '} {' '}
```

```
INSERT INTO set_tbl VALUES ('');
```

```
ERROR: Casting '' to type set is not supported.
```

MULTISET

MULTISET 는 중복이 허용되는 집합이다. **MULTISET** 의 원소는 하나의 데이터 타입을 가져야 하고, 다른 테이블의 레코드를 가질 수도 있다.

```
CREATE TABLE multiset_tbl (col_1 MULTISET (CHAR(1)));
INSERT INTO multiset_tbl VALUES ({'c','c','c','b','b','a'});
SELECT * FROM multiset_tbl;
```

```
col_1
=====
{'a', 'b', 'b', 'c', 'c', 'c'}
```

```
SELECT CAST(col_1 AS SET), CAST(col_1 AS LIST) FROM multiset_tbl;
```

```
cast(col_1 as set)  cast(col_1 as sequence)
=====
{'a', 'b', 'c'}  {'c', 'c', 'c', 'b', 'b', 'a'}
```

LIST 또는 SEQUENCE

LIST (SEQUENCE) 는 원소가 입력된 순서가 유지되며, 중복이 허용되는 집합이다. **LIST** 의 원소는 하나의 데이터 타입을 가져야 하고, 다른 테이블의 레코드를 가질 수도 있다.

```
CREATE TABLE list_tbl (col_1 LIST (CHAR(1)));
INSERT INTO list_tbl VALUES ({'c','c','c','b','b','a'});
SELECT * FROM list_tbl;
```

```
col_1
=====
{'c', 'c', 'c', 'b', 'b', 'a'}
```

```
SELECT CAST(col_1 AS SET), CAST(col_1 AS MULTiset) FROM list_tbl;
```

```
cast(col_1 as set)  cast(col_1 as multiset)
=====
{'a', 'b', 'c'}  {'a', 'b', 'b', 'c', 'c', 'c'}
```

JSON 데이터 타입

CUBRID 10.2부터 RFC 7159 에서 정의된 native **JSON** 데이터 타입을 지원한다. **JSON** 데이터 타입은 JSON 데이터에 대해 자동 검증을 제공하며, 빠른 액세스와 동작이 가능하다.

참고

10.2 서버에 연결하는 그 이전 버전의 드라이버는 JSON 타입 컬럼을 Varchar 타입으로 인식한다.

JSON 데이터 생성

JSON 포맷의 문자열 형식이 JSON 데이터 타입 컬럼에 입력되면 JSON 값으로 자동으로 변환된다.

```
-- 문자열을 JSON 타입 컬럼에 입력
CREATE TABLE t (id int, j JSON);
INSERT INTO t VALUES (1, '{"a":1}');
SELECT j, TYPEOF(j) FROM t;
```

```
j                                typeof(j)
=====
{"a":1}                          'json'
```

CAST 를 사용하거나 문자열 앞에 json 키워드를 사용하여 강제로 JSON으로 변환 할 수도 있다.

```
-- 문자열을 json으로 cast
SELECT CAST('{"a":1}' as JSON);
```

```
cast('{"a":1}' as json)
=====
{"a":1}
```

```
-- json 키워드 사용
SELECT json '{"a":1}', TYPEOF (json '{"a":1}');
```

```
json '{"a":1}'          typeof(json '{"a":1}')
=====
{"a":1}                'json'
```

JSON 데이터 타입은 `JSON_OBJECT` 나 `JSON_ARRAY` 를 사용하여 생성할 수도 있다.

JSON 유효성 검사

JSON 데이터로의 변환은 내장된 유효성 검사를 수행하고 유효한 JSON 문자열이 아닌 경우 오류를 반환한다.

```
-- 따옴표가 없는 문자열은 유효한 json이 아니다
SELECT json 'abc';
```

In line 1, column 8,

에러: before ' ; '
유효하지 않은 JSON: 'abc'.

더 엄격한 유효성 검사 규칙을 가진 JSON 타입 컬럼은 `JSON 스키마 표준 초안` (draft JSON Schema standard) 를 사용하여 정의할 수 있다. 만약 JSON 스키마를 다루어본 적이 없다면 `JSON 스키마의 이해` (Understanding JSON Schema) 를 참고하면 된다.

다음은 스키마 사용 방법에 대한 간단한 예제이다.:

```
-- j 컬럼이 JSON의 문자열 타입만을 입력할 수 있도록 설정
CREATE TABLE t (id int, j JSON ('{"type": "string"}'));
```

```
-- JSON 스키마에 유효한 문자열 타입 JSON 입력
INSERT into t values (1, '"abc"');
```

1 개의 명령어가 실행되었습니다.

```
-- JSON 스키마에 유효하지 않은 문자열 타입 JSON 입력
INSERT into t values (2, '{"a":1}');
```

```
에러: before ' ); '
주어진 JSON 은 JSON schema에 대해 유효하지 않습니다. (스키마 경로: #, 키워드: type, JSON 경로: #)
```

JSON 데이터의 타입

JSON 데이터의 값은 RFC 7159 <<https://datatracker.ietf.org/doc/html/rfc7159#section-3>> 에서 정의된 것과 같이 객체 (Object), 배열 (Array) 또는 스칼라 (Scalar) 여야 한다. 스칼라 값은 문자열, 숫자형, 불리언 (boolean) 또는 널 (null) 이다.

JSON 데이터 타입 표:

타입	CUBRID JSON 타입		설명
Object	JSON_OBJECT		키-값 쌍의 집합
Array	JSON_ARRAY		JSON 데이터 값의 배열
Scalar	String	STRING	따옴표로 묶인 문자열
	Number	INTEGER	32비트의 부호 있는 (signed) 숫자
		BIGINT	64비트의 부호 있는 (signed) 숫자
		DOUBLE	정수가 아닌 숫자 또는 $2^{63} - 1$ 보다 큰 정수
	true	BOOLEAN	True 불리언 (boolean) 값
	false	BOOLEAN	

타입	CUBRID JSON 타입	설명
		False 불리언 (boolean) 값
null	JSON_NULL	Null 값

JSON 값의 CUBRID JSON 타입은 `JSON_TYPE` 함수로 가져올 수 있다.

JSON 데이터 변환

JSON 데이터 타입은 명시적 또는 암시적으로 다른 타입으로부터 변환하여 얻을 수 있다.

JSON 데이터값을 다른 JSON 타입으로 변환하는 것은 변환하려는 타입이 스키마를 가지고 있고 변환된 값이 스키마 유효성 검사를 통과하지 못하는 경우 실패할 수 있다.

다른 타입에서 JSON으로의 변환은 다음 표에서 설명한다:

기존 타입	CUBRID JSON 타입
Any string	문자열은 모든 타입의 JSON 데이터로 변환할 수 있다.
	참고 만약 문자열의 문자셋이 UTF8이 아니라면 문자열을 먼저 UTF8로 변환한 후 JSON 데이터로 변환한다.
Short, Integer	INTEGER
Bigint	BIGINT
Float, Double	DOUBLE

기존 타입	CUBRID JSON 타입

Numeric	DOUBLE

JSON 데이터 타입에서 다른 타입으로의 변환은 다음 표에서 설명한다.:

CUBRID JSON 타입	허용되는 다른 타입

JSON_OBJECT	JSON 데이터 값을 출력한 문자열

JSON_ARRAY	JSON 데이터 값을 출력한 문자열

STRING	문자열로부터 변환할 수 있는 모든 타입

INTEGER	숫자값으로부터 변환할 수 있는 모든 타입

BIGINT	Bigint 값으로부터 변환할 수 있는 모든 타입

DOUBLE	Double 값으로부터 변환할 수 있는 모든 타입

BOOLEAN	문자열로 변환된다면 "true" 또는 "false"

	숫자형으로 변환된다면 0 또는 1

JSON_NULL	JSON 'null'을 출력한 문자열

JSON 경로

JSON 경로는 JSON 내에서 json 요소를 참조할 수 있는 방법이다. 많은 JSON 함수는 JSON 내부에서 동작 위치를 정의하기 위해 JSON 경로 또는 JSON 포인터 인수를 필요로 한다. JSON 경로는 항상 '\$'로 시작하며 배열 인덱스, 객체 키 토큰 및 와일드카드가 뒤따를 수 있다. 만약 '\$' 뒤에 다른 토큰이 없으면 경로는 JSON 데이터 루트를 가리킨다.

```

<json_path>::=
  <start_token> [<path_token>] ...

<start_token>::=
  $

<path_token>::=
  <array_access_token> | <object_key_access_token> |
  <wildcard_token>

<array_access_token>::=
  [idx]

<object_key_access_token>::=
  .[key_identifier | "key_str"]

<wildcard_token>::=
  .*|[*]|**path_token

```

예를 들어, ‘{“a”:[0,1,2,{{“b”:5}}]}’에 대해 ‘\$.a[3].b’의 의미는 “루트의 ‘a’를 가진 멤버의 인덱스 3에 해당하는 요소의 키 ‘b’를 가진 멤버”이며 JSON 값 ‘5’를 가리킨다;

object_key_access_tokens를 키 문자열로 사용하여 동일한 key_identifiers를 표현할 수 있으며 이스케이프 문자를 사용할 수도 있다. 예: ‘\$’은 큰따옴표를 가진 멤버를 키로 참조하는 데 사용할 수 있다.

JSON 와일드카드는 다음 세 가지 유형 중 하나가 될 수 있다.

- *, 와일드카드와 매칭되는 객체 멤버
- [*, 와일드카드와 매칭되는 배열 인덱스
- **, 객체 키의 시퀀스 배열 인덱스를 매칭. ** 와일드카드는 토큰(path_token)을 뒤에 붙여야 한다.

JSON 포인터 및 JSON 텍스트와 같은 경로 식은 ASCII 또는 UTF-8 문자셋으로 인코딩되어야 한다. 만약 다른 문자셋이 사용된다면, UTF-8으로 변환(coercion)될 것이다.

JSON 포인터

<https://datatracker.ietf.org/doc/html/rfc6901> 에서 정의한 JSON 포인터는 JSON 경로와 다른 방법을 제공한다. JSON 포인터는 JSON 경로 및 JSON 텍스트와 동일하게 ASCII 또는 UTF-8 문자셋으로 인코딩되어야 한다. 만약 다른 문자셋이 사용된다면, UTF-8으로 변환(coercion)될 것이다.

```

<json_pointer>::=
  [/path_token] ... [/-]

```



```
'$.a[10].bb' 는 '/a/10/bb' 와 동일하다
'$' 는 '' 와 동일하다
```

특수 문자 '-'는 마지막 path_token으로만 사용할 수 있으며 json_array의 끝부분을 가리키는 데 사용할 수 있다.

JSON 포인터는 비와일드카드 JSON 경로가 가리키는 것과 동일한 경로를 가리키는데 사용할 수 있다.

묵시적 타입 변환

표현식 내에서 타입을 변환해야 할 때 자동으로 해당 타입으로 변환하는 것을 묵시적 타입 변환(implicit type conversion)이라고 한다.

SET, MULTISSET, LIST, SEQUENCE 는 명시적으로 변환되어야 한다.

DATETIME 및 **TIMESTAMP** 타입(타임존이 있는 타입 포함)을 **DATE** 타입 또는 **TIME** 타입으로 변환하면 데이터 손실이 발생한다. **DATE** 타입을 **DATETIME** 타입 또는 **TIMESTAMP** 타입(타임존이 있는 타입 포함)으로 변환하면 시간이 '12:00:00 AM'으로 설정된다.

타임존 타입 값에서 타임존 부분은 참조용일 뿐이며, 정확한 값은 UTC 참조에 저장된다. 타임존이 있는 타입에서 타임존이 없는 타입으로 값을 변환하면 세션 타임존을 사용한 것처럼 변환된다. 타임존이 없는 타입에서 타임존이 있는 타입으로 값을 변환하면 세션 타임존을 고려하여 변환한다. 타임존 타입과 관련한 값 변환에 대한 자세한 내용은 [날짜/시간 데이터 타입](#) 을 참고한다.

문자열 타입이나 정확한 수치형 타입을 부동소수점 수치형 타입으로 변환하면 값이 정확하지 않을 수 있다. 문자열 타입과 정확한 수치형 타입은 값을 표현하기 위해 십진 정밀도(decimal precision)를 사용하지만 부동소수점 수치형 타입은 이진 정밀도(binary precision)를 사용하기 때문이다.

CUBRID가 수행하는 묵시적 타입 변환은 다음과 같다.

묵시적 타입 변환 표 1

From \ To	DATE TIME	DATE TIMEL TZ	DATE TIMET Z	DATE	TIME	TIMES TAMP	TIMES TAMP LTZ	TIMES TAMP TZ
DATETIME	-	0	0	0	0	0	0	0

From \ To	DATE TIME	DATE TIMEL TZ	DATE TIMET Z	DATE	TIME	TIMES TAMP	TIMES TAMP LTZ	TIMES TAMP TZ
DATETI MELTZ	0	-	0	0	0	0	0	0
DATETI METZ	0	0	-	0	0	0	0	0
DATE	0	0	0	-		0	0	0
TIME					-			
TIMEST AMP	0	0	0	0	0	-	0	0
TIMEST AMPLTZ	0	0	0	0	0	0	-	0
TIMEST AMPTZ	0	0	0	0	0	0	0	-
DOUBL E					0	0	0	0
FLOAT					0	0	0	0
NUMER IC						0	0	0
BIGINT					0	0	0	0

From To	DATE TIME	DATE TIMEL TZ	DATE TIMET Z	DATE	TIME	TIMES TAMP	TIMES TAMP LTZ	TIMES TAMP TZ
INT					0	0	0	0
SHORT					0	0	0	0
BIT								
VARBIT								
CHAR	0	0	0	0	0	0	0	0
VARCHAR	0	0	0	0	0	0	0	0

숫자 값이 **TIME** 또는 **TIMESTAMP(TIMESTAMPLTZ, TIMESTAMPTZ)**로 변경될 때의 제약 사항

- **NUMERIC** 타입을 제외한 모든 숫자 타입은 **TIME** 타입으로 변환될 수 있으며, 이때 **TIME**은 입력 숫자를 86,400초(1일)로 나눈 나머지 값을 초로 계산한 값이다.
- **NUMERIC**을 포함한 모든 숫자 타입은 **TIMESTAMP, TIMESTAMPLTZ, TIMESTAMPTZ** 타입으로 변환될 수 있으며, 이때 입력 숫자는 최대 2,147,483,647을 초과할 수 없다.

목시적 타입 변환 표 2

From To	INT	SHORT	BIT	VAR BIT	CHAR	VAR CHAR	DOUBLE	FLOAT	NUMERIC	BIGINT
DATE TIME					0	0				
					0	0				

From \ To	INT	SHORT	BIT	VARIABLE	CHAR	VARIABLE	DOUBLE	FLOAT	NUMERIC	BIGINT
DATE										
TIMESTAMP										
DATE					0	0				
TIMESTAMP										
DATE					0	0				
TIME					0	0				
TIMESTAMP					0	0				
TIMESTAMP					0	0				
TIMESTAMP					0	0				
DOUBLE	0	0			0	0	-	0	0	0
FLOAT	0	0			0	0	0	-	0	0
NUMERIC	0	0			0	0	0	0	-	0

From \ To	INT	SHORT	BIGINT	VARCHAR	CHAR	VARCHAR	DOUBLE	FLOAT	NUMERIC	BIGINT
BIGINT	0	0			0	0	0	0	0	-
INT	-	0			0	0	0	0	0	0
SHORT	0	-			0	0	0	0	0	0
BIGINT			-	0	0	0				
VARCHAR			0	-	0	0				
CHAR	0	0	0	0	-	0	0	0	0	0
VARCHAR	0	0	0	0	0	-	0	0	0	0

변환 규칙

INSERT와 UPDATE

영향을 받는 칼럼의 타입으로 값의 타입이 변환된다.

```
CREATE TABLE t(i INT);
INSERT INTO t VALUES('123');

SELECT * FROM t;
```

```
i
=====
123
```

함수

함수에 입력한 인자 값을 함수에서 지정한 타입으로 변환할 수 있으면 인자의 타입이 변환된다. 아래 함수에서 기대하는 입력 인자는 숫자이므로, 문자열을 숫자로 변환한다.

```
SELECT MOD('123', '2');
```

```
mod('123', '2')
=====
1.0000000000000000e+00
```

함수는 인자로 여러 타입의 값을 입력받을 수 있는데, 함수에서 지정하지 않은 타입의 값이 전달되면 다음의 우선순위에 따라 타입이 변환된다.

- 날짜/시간 타입 (**DATETIME** > **TIMESTAMP** > **DATE** > **TIME**)
- 근사치 수치형 타입 (**DOUBLE** > **FLOAT**)
- 정확한 수치형 타입 (**NUMERIC** > **BIGINT** > **INT** > **SHORT**)
- 문자열 타입 (**CHAR** > **VARCHAR**)

비교 연산

다음은 비교 연산자의 피연산자 타입에 따른 변환 규칙이다.

피연산자1 타입	피연산자2 타입	변환	비교 타입
수치형 타입	수치형 타입	없음	NUMERIC
	문자열 타입	피연산자2를 DOUBLE 로 변환	NUMERIC
	날짜/시간 타입	피연산자1을 날짜/시간으로 변환	TIME/TIMESTAMP

피연산자1 타입	피연산자2 타입	변환	비교 타입
문자열 타입	수치형 타입	피연산자1을 DOUBLE 로 변환	NUMERIC
	문자열 타입	없음	문자열
	날짜/시간 타입	피연산자1을 날짜/시간 타입으로 변환	날짜/시간
날짜/시간 타입	수치형 타입	피연산자2를 날짜/시간으로 변환	TIME/TIMESTAMP
	문자열 타입	피연산자2를 날짜/시간 타입으로 변환	날짜/시간
	날짜/시간 타입	우선순위가 높은 타입으로 변환	날짜/시간

날짜/시간 타입과 수치형 타입이 비교되는 경우 위 표의 **숫자가 TIME 또는 TIMESTAMP로 변환될 때 제약 조건**을 참고한다.

피연산자1이 문자열 타입의 칼럼이고 피연산자2가 값인 경우, 다음과 같은 예외가 있다.

피연산자1 타입	피연산자2 타입	변환	비교
문자열 타입의 칼럼	수치형 타입 값	피연산자2를 문자열 타입으로 변환	문자열
	날짜/시간 타입 값	피연산자2를 문자열 타입으로 변환	문자열

피연산자2가 집합인 연산자(**IS IN**, **IS NOT IN**, **= ALL**, **= ANY**, **< ALL**, **< ANY**, **<= ALL**, **<= ANY**, **>= ALL**, **>= ANY**)에 대해서는 위의 예외가 적용되지 않는다.

다음은 비교 연산에서 묵시적 타입 변환의 예이다.

· 수치형 타입과 문자열 타입 피연산자

문자열 타입 피연산자가 **DOUBLE** 로 변환된다.

```
CREATE TABLE t1(i INT, s STRING);
INSERT INTO t1 VALUES(1,'1'),(2,'2'),(3,'3'),(4,'4'), (12,'12');

SELECT i FROM t1 WHERE i < '11.3';
```

```
      i
=====
      1
      2
      3
      4
```

```
SELECT ('2' <= 11);
```

```
 ('2'<=11)
=====
          1
```

· 문자열 타입과 날짜/시간 타입 피연산자

문자열 타입 피연산자가 날짜/시간 타입으로 변환된다.

```
SELECT ('2010-01-01' < date'2010-02-02');
```

```
 ('2010-01-01'<date '2010-02-02')
=====
                                1
```

```
SELECT (date'2010-02-02' >= '2010-01-01');
```

```
 (date '2010-02-02'>='2010-01-01')
=====
                                1
```


· 문자열 타입과 수치형 타입 호스트 변수 피연산자

수치형 타입 호스트 변수가 문자열 타입으로 변환된다.

```
PREPARE s FROM 'SELECT s FROM t1 WHERE s < ?';
EXECUTE s USING 11;
```

```
      s
=====
      '1'
```

· 문자열 타입 칼럼과 수치형 타입 값 피연산자

수치형 타입 값이 문자열 타입으로 변환된다.

```
SELECT s FROM t1 WHERE s > 11;
```

```
      s
=====
      '2'
      '3'
      '4'
      '12'
```

```
SELECT s FROM t1 WHERE s BETWEEN 11 AND 33;
```

```
      s
=====
      '2'
      '3'
      '12'
```

· 문자열 타입 칼럼과 날짜/시간 타입 값 피연산자

날짜/시간 타입 값이 문자열 타입으로 변환된다.

```
CREATE TABLE t2 (s STRING);
INSERT INTO t2 VALUES ('01/01/1998'), ('01/01/1999'),
('01/01/2000');
SELECT s FROM t2;
```

```

      s
=====
'01/01/1998'
'01/01/1999'
'01/01/2000'

```

```
SELECT s FROM t2 WHERE s <= date'02/02/1998';
```

위의 질의에서 date'02/02/1998'이 문자열 '02/02/1998'으로 변환되어 문자열 간의 비교 연산이 수행된다.

```

      s
=====
'01/01/1998'
'01/01/1999'
'01/01/2000'

```

범위 연산

· 수치형 타입과 문자열 타입 피연산자

문자열 타입 피연산자가 **DOUBLE** 로 변환된다.

```
CREATE TABLE t3 (i INT);
INSERT INTO t3 VALUES (1), (2), (3), (4);
SELECT i FROM t3 WHERE i <= ALL {'11', '12'};
```

```

      i
=====
1
2
3
4

```

· 문자열 타입과 날짜/시간 타입 피연산자

문자열 타입 피연산자가 날짜/시간 타입으로 변환된다.

```
SELECT s FROM t2;
```

```

s
=====
'01/01/1998'
'01/01/1999'
'01/01/2000'

```

```
SELECT s FROM t2 WHERE s <= ALL {date'02/02/1998',date'01/01/2000'};
```

```

s
=====
'01/01/1998'

```

해당 타입으로 변환할 수 없으면 오류를 반환한다.

산술 연산

· 날짜/시간 타입 피연산자

날짜/시간 타입의 피연산자가 '-' 연산자에 주어지고 타입이 서로 다르면, 두 타입을 비교하여 우선순위가 높은 쪽의 타입으로 변환된다. 다음 예는 왼쪽 피연산자의 데이터 타입이 **DATE** 에서 **DATETIME** 으로 바뀌어 결과는 **DATETIME** 의 '-' 연산 결과인 밀리초를 출력한다.

```
SELECT date'2002-01-01' - datetime'2001-02-02 12:00:00 am';
```

```

date '2002-01-01' - datetime '2001-02-02 12:00:00 am'
=====
28771200000

```

· 수치형 타입 피연산자

수치형 타입의 피연산자가 주어지고 타입이 서로 다르면, 두 타입을 비교하여 우선순위가 높은 쪽의 타입으로 변환된다.

· 날짜/시간 타입과 수치형 타입 피연산자

날짜/시간 타입과 수치형 타입의 피연산자가 '+' 또는 '-' 연산자에 주어지면, 수치형 타입 피연산자는 **BIGINT**, **INT**, **SHORT** 중 하나로 변환된다.

· 날짜/시간 타입과 문자열 타입 피연산자

날짜/시간 타입과 문자열 타입이 피연산자이면, '+'와 '-' 연산자만 허용한다. '+' 연산자가 사용되면 다음 규칙이 적용된다.

- 문자열 타입은 인터벌(interval) 값을 지닌 **BIGINT** 로 변환된다. 인터벌은 날짜/시간 타입 피연산자의 가장 작은 단위를 의미하며, 각 타입의 인터벌 값은 다음과 같다.
 - **DATE** : 일수(days)
 - **TIME, TIMESTAMP** : 초수(seconds)
 - **DATETIME** : 밀리초수(milliseconds)
- 부동소수점수는 반올림된다.
- 결과 타입은 날짜/시간 타입 피연산자의 타입이다.

```
SELECT date'2002-01-01' + '10';
```

```
date '2002-01-01'+'10'
=====
01/11/2002
```

날짜/시간 타입과 문자열 타입이 피연산자이고, '-' 연산자가 사용되면 다음 규칙이 적용된다.

- 날짜/시간 타입 피연산자가 **DATE, DATETIME, TIMESTAMP** 이면 문자열은 **DATETIME** 으로 변환되고, 날짜/시간 타입 피연산자가 **TIME** 이면 문자열은 **TIME** 으로 변환된다.
- 결과 타입은 항상 **BIGINT** 이다.

```
SELECT date'2002-01-01'-'2001-01-01';
```

```
date '2002-01-01'-'2001-01-01'
=====
31536000000
-- this causes an error
```

```
SELECT date'2002-01-01'-'10';
```

```
ERROR: Cannot coerce '10' to type datetime.
```

· 수치형 타입과 문자열 타입 피연산자

수치형 타입과 문자열 타입이 피연산자이면 다음 규칙이 적용된다.

- 문자열은 가능하면 **DOUBLE**로 변환된다.
- 결과 타입은 **DOUBLE**이다.

```
SELECT 4 + '5.2';
```

```
          4+'5.2'
=====
9.199999999999999e+00
```

CUBRID 2008 R3.1 이하 버전과 달리, 날짜/시간 형태의 문자열, 즉 '2010-09-15'와 같은 문자열은 날짜/시간 타입으로 변환되지 않는다. 날짜/시간 타입을 갖는 리터럴 DATE'2010-09-15'은 덧셈, 뺄셈 연산에 사용할 수 있다.

```
SELECT '2002-01-01'+1;
```

```
ERROR: Cannot coerce '2002-01-01' to type double.
```

```
SELECT date'2002-01-01'+1;
```

```
date '2002-01-01'+1
=====
01/02/2002
```

· 문자열 타입 피연산자

두 문자열을 곱하거나 나누거나 빼면 숫자로 변환되며, 결과로 **DOUBLE** 타입의 값을 반환한다.

```
SELECT '3'*'2';
```

```
          '3'*'2'
=====
6.000000000000000e+00
```

'+' 연산자의 동작은 **cubrid.conf**의 시스템 파라미터인 **plus_as_concat**을 어떻게 설정하느냐에 따라 결정된다. 자세한 내용은 **구문/타입 관련 파라미터**를 참고한다.

- **plus_as_concat** 값이 yes(기본값)이면 두 개의 문자열을 연결한 값을 반환한다.

```
SELECT '1'+'1';
```

```
      '1'+'1'
=====
      '11'
```

- **plus_as_concat** 값이 no이고 두 개의 문자열이 숫자로 변환 가능하면, 두 숫자를 더하여 **DOUBLE** 타입의 값을 반환한다.

```
SELECT '1'+'1';
```

```
      '1'+'1'
=====
2.0000000000000000e+00
```

해당 타입으로 변환할 수 없으면 오류를 반환한다.

데이터 정의문

테이블 정의문

CREATE TABLE

테이블 정의

CREATE TABLE 문을 사용하여 새로운 테이블을 생성한다. 기존 뷰 또는 동의어와 이름이 같은 테이블은 생성할 수 없다. 테이블 이름 작성 원칙은 **식별자**를 참고한다.

```
CREATE {TABLE | CLASS} [IF NOT EXISTS] [schema_name.]table_name
[<subclass_definition>]
[CLASS ATTRIBUTE (<column_definition>, ...)]
[[({<table_constraint>}... ,] <column_definition> [{,
{<column_definition> | <table_constraint>}}...])]
[INHERIT <resolution>, ...]
[<table_options>]

<subclass_definition> ::= {UNDER | AS SUBCLASS OF}
```

```

[schema_name.]superclass_name, ...

<column_definition> ::=
    column_name <data_type> [{<default_or_shared_or_ai> |
<on_update> | <column_constraint>}] [COMMENT 'column_comment_string']

    <data_type> ::= <column_type> [<charset_modifier_clause>]
[<collation_modifier_clause>]

    <charset_modifier_clause> ::= {CHARACTER_SET|CHARSET}
{<char_string_literal>|<identifier>}

    <collation_modifier_clause> ::= COLLATE
{<char_string_literal>|<identifier>}

    <default_or_shared_or_ai> ::=
        SHARED <value_specification> |
        DEFAULT <value_specification> |
        AUTO_INCREMENT [(seed, increment)]

    <on_update> ::= [ON UPDATE <value_specification>]

    <column_constraint> ::= [CONSTRAINT constraint_name] { NOT
NULL | UNIQUE | PRIMARY KEY | [FOREIGN KEY] [WITH
<index_with_option>] <referential_definition> }

<table_constraint> ::=
{
    {KEY|INDEX} index_name <index_col_desc> |
    [CONSTRAINT [constraint_name]]
    {
        UNIQUE [KEY|INDEX](column_name, ...) |
        PRIMARY KEY (column_name, ...) |
        <referential_constraint>
    }
} COMMENT 'index_comment_string'

<index_col_desc> ::=
{ ( {column_name | function_name (argument_list)} [ASC |
DESC] [{, {column_name | function_name (argument_list)} [ASC |
DESC]} ...] ) }
    [WHERE <filter_predicate>]
    [WITH <index_with_option>]
    [INVISIBLE]
    [COMMENT 'index_comment_string']

    <referential_constraint> ::= FOREIGN KEY [<foreign_key_name>]
(column_name, ...) [WITH <index_with_option>]
<referential_definition>

    <referential_definition> ::=
        REFERENCES [schema_name.]referenced_table_name

```

```

(column_name, ...) [<referential_triggered_action> ...]

    <referential_triggered_action> ::=
        ON UPDATE <referential_action> |
        ON DELETE <referential_action>

    <referential_action> ::= CASCADE | RESTRICT | NO
ACTION | SET NULL

    <resolution> ::= [CLASS] {column_name} OF
[schema_name.]superclass_name [AS alias]
    <table_options> ::= <table_option> [[,] <table_option> ...]
    <table_option> ::= REUSE_OID | DONT_REUSE_OID |
        COMMENT [=] 'table_comment_string' |
        [CHARSET charset_name] [COLLATE
collation_name] |
        ENCRYPT [=] [AES | ARIA] |
        AUTO_INCREMENT = initial_value

    <index_with_option> ::= {DEDUPLICATE '=' deduplicate_level }

```

- **IF NOT EXISTS:** 생성하려는 테이블이 존재하는 경우 에러 없이 테이블을 생성하지 않는다.
- *schema_name*: 스키마 이름을 지정한다(최대 31바이트). 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 생성할 테이블의 이름을 지정한다(최대 222바이트).
- *column_name*: 생성할 칼럼의 이름을 지정한다(최대 254바이트).
- *column_type*: 칼럼의 데이터 타입을 지정한다.
- [**SHARED** *value* | **DEFAULT** *value*]: 칼럼의 초기값을 지정한다.
- **ON UPDATE**: 레코드의 필드가 수정되었을 때 갱신될 필드에 대한 수식을 지정한다. 자세한 내용은 **ON UPDATE** 을 참고한다.
- *<column_constraint>*: 칼럼의 제약 조건을 지정하며 제약 조건의 종류에는 **NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY** 가 있다.
- *<default_or_shared_or_ai>*: **DEFAULT, SHARED, AUTO_INCREMENT** 중 하나만 사용될 수 있다. **AUTO_INCREMENT**이 지정될 때 “(seed, increment)”와 “**AUTO_INCREMENT** = *initial_value*”는 동시에 정의될 수 없다.
- *table_comment_string*: 테이블의 커멘트를 지정한다.
- *column_comment_string*: 칼럼의 커멘트를 지정한다.
- *index_comment_string*: 인덱스의 커멘트를 지정한다.

- *deduplicate_level*: deduplicate 레벨을 지정한다(0 ~ 14). 자세한 내용은 **DEDUPLICATE**를 참고한다.

참고

- **DBA**와 **DBA** 멤버는 다른 스키마에 테이블을 생성할 수 있다. 사용자가 **DBA**도 아니고 **DBA** 멤버도 아니면 해당 사용자의 스키마에서만 테이블을 생성할 수 있다.

참고

*deduplicate_level*은 0부터 14까지의 정수이다. 0은 **DEDUPLICATE** 옵션이 없었던 CUBRID 11.2 또는 이하 버전과 동일한 구성의 인덱스를 의미한다.

```
CREATE TABLE olympic2 (
    host_year      INT      NOT NULL PRIMARY KEY,
    host_nation    VARCHAR(40) NOT NULL,
    host_city      VARCHAR(20) NOT NULL,
    opening_date   DATE      NOT NULL,
    closing_date   DATE      NOT NULL,
    mascot         VARCHAR(20),
    slogan         VARCHAR(40),
    introduction    VARCHAR(1500)
);
```

다음은 ALTER 문을 사용하여 테이블 커멘트를 추가하는 예제이다.

```
ALTER TABLE olympic2 COMMENT = 'this is new comment for olympic2';
```

다음은 테이블 생성 시 인덱스 커멘트를 포함하는 예제이다.

```
CREATE TABLE tbl (a INT, index i_t_a (a) COMMENT 'index comment');
```

참고

테이블 스키마의 CHECK 제약 조건

테이블 스키마에 정의된 CHECK 제약 조건은 파싱되지만, 실제 동작은 무시된다. 파싱되는 이유는 타 DBMS로부터 마이그레이션을 진행하는 경우 호환성을 제공하기 위해서이다.

```
CREATE TABLE tbl (
    id INT PRIMARY KEY,
    CHECK (id > 0)
)
```

칼럼 정의

칼럼은 테이블에서 각 열에 해당하는 항목이며, 칼럼은 칼럼 이름과 데이터 타입을 명시하여 정의한다.

```
<column_definition> ::=
    column_name <data_type> [[<default_or_shared_or_ai>] |
    [<on_update>] | [<column_constraint>]] ... [COMMENT 'comment_string']

    <data_type> ::= <column_type> [<charset_modifier_clause>]
    [<collation_modifier_clause>]

    <charset_modifier_clause> ::= {CHARACTER_SET|CHARSET}
    {<char_string_literal>|<identifier>}

    <collation_modifier_clause> ::= COLLATE
    {<char_string_literal>|<identifier>}

    <default_or_shared_or_ai> ::=
        SHARED <value_specification> |
        DEFAULT <value_specification> |
        AUTO_INCREMENT [(seed, increment)]

    <on_update> ::= [ON UPDATE <value_specification>]

    <column_constraint> ::= [CONSTRAINT constraint_name] {NOT NULL |
    UNIQUE | PRIMARY KEY | [FOREIGN KEY] [WITH <index_with_option>]
    <referential_definition>}
```

칼럼 이름

칼럼 이름 작성 원칙은 **식별자** 절을 참고한다. 생성한 칼럼의 이름은 **ALTER TABLE** 문의 **RENAME COLUMN** 절을 사용하여 변경할 수 있다.

다음은 *full_name* 과 *age*, 2개의 칼럼을 가지는 *manager2* 테이블을 생성하는 예제이다.

```
CREATE TABLE manager2 (full_name VARCHAR(40), age INT );
```

참고

- 칼럼 이름의 첫 글자는 반드시 알파벳이어야 한다.
- 칼럼 이름은 테이블 내에서 고유해야 한다.

칼럼의 초기 값 설정(SHARED, DEFAULT)

테이블의 칼럼의 초기값을 **SHARED** 또는 **DEFAULT** 값을 통해 정의할 수 있다. **SHARED**, **DEFAULT** 값은 **ALTER TABLE** 문에서 변경할 수 있다.

- **SHARED** : 칼럼 값은 모든 행에서 동일하다. 따라서 **SHARED** 속성은 **UNIQUE** 제약 조건과 동시에 정의할 수 없다. 초기에 설정한 값과 다른 새로운 값을 **INSERT** 하면, 해당 칼럼 값은 모든 행에서 새로운 값으로 갱신된다.
- **DEFAULT** : 새로운 행을 삽입할 때 칼럼 값을 지정하지 않으면 **DEFAULT** 속성으로 설정한 값이 저장된다.

DEFAULT 의 값으로 허용되는 의사 칼럼(pseudocolumn)과 함수는 다음과 같다.

DEFAULT 값	데이터 타입
SYS_TIMESTAMP	TIMESTAMP
UNIX_TIMESTAMP()	INTEGER
CURRENT_TIMESTAMP	TIMESTAMP
SYS_DATETIME	DATETIME
CURRENT_DATETIME	DATETIME
SYS_DATE	DATE
CURRENT_DATE	DATE

DEFAULT 값	데이터 타입
SYS_TIME	TIME
CURRENT_TIME	TIME
USER, USER()	STRING
TO_CHAR(date_time[, format])	STRING
TO_CHAR(number[, format])	STRING

참고

위의 표에 명시되지 않은 함수를 **DEFAULT** 값으로 사용하는 경우, 테이블 생성 시점에 평가된 결과를 **DEFAULT** 값으로 저장한다. 따라서 **INSERT** 시 해당 칼럼의 기본 값은 **INSERT** 수행시 함수 수행 값이 아닌, 테이블 생성 시점에 수행된 결과 값을 사용한다. 저장 함수인 경우 **DEFAULT** 값으로 사용할 수 없으므로 오류를 반환한다.

```
CREATE TABLE default_val_func_test (id INT DEFAULT RAND ());
INSERT INTO default_val_func_test (id) VALUES (1);
INSERT INTO default_val_func_test (id) VALUES (DEFAULT);
INSERT INTO default_val_func_test (id) VALUES (DEFAULT);

SELECT * FROM default_val_func_test;
```

```
      id
=====
      1
1572882525
1572882525
```

```
CREATE FUNCTION fn_int (a int) return int as begin return a; end;
CREATE TABLE default_val_sp_test (id INT DEFAULT fn_int (1));
```

```
ERROR: before ' ); '
'fn_int(1)' function can not be used in DEFAULT clause.
```

❗ 참고

CUBRID 9.0 미만 버전에서는 테이블 생성 시 **DATE, DATETIME, TIME, TIMESTAMP** 칼럼의 **DEFAULT** 값을 **SYS_DATE, SYS_DATETIME, SYS_TIME, SYS_TIMESTAMP** 로 지정하면, **CREATE TABLE** 시점의 값이 저장되었다. 따라서 CUBRID 9.0 미만 버전에서 데이터가 **INSERT** 되는 시점의 값을 입력하려면 **INSERT** 구문의 **VALUES** 절에 해당 함수를 입력해야 한다.

```
CREATE TABLE colval_tbl
(id INT, name VARCHAR SHARED 'AAA', phone VARCHAR DEFAULT
'000-0000');
INSERT INTO colval_tbl (id) VALUES (1), (2);
SELECT * FROM colval_tbl;
```

id	name	phone
1	'AAA'	'000-0000'
2	'AAA'	'000-0000'

```
--updating column values on every row
INSERT INTO colval_tbl(id, name) VALUES (3,'BBB');
INSERT INTO colval_tbl(id) VALUES (4),(5);
SELECT * FROM colval_tbl;
```

id	name	phone
1	'BBB'	'000-0000'
2	'BBB'	'000-0000'
3	'BBB'	'000-0000'
4	'BBB'	'000-0000'
5	'BBB'	'000-0000'

```
--changing DEFAULT value in the ALTER TABLE statement
ALTER TABLE colval_tbl MODIFY phone VARCHAR DEFAULT '111-1111';
INSERT INTO colval_tbl (id) VALUES (6);
SELECT * FROM colval_tbl;
```

id	name	phone
1	'BBB'	'000-0000'
2	'BBB'	'000-0000'
3	'BBB'	'000-0000'
4	'BBB'	'000-0000'
5	'BBB'	'000-0000'
6	'BBB'	'111-1111'

```
--use DEFAULT TO_CHAR in CREATE TABLE statement
CREATE TABLE t1(id1 INT, id2 VARCHAR(20) DEFAULT
TO_CHAR(12345,'S999999'));
INSERT INTO t1 (id1) VALUES (1);
SELECT * FROM t1;
```

id1	id2
1	' +12345'

하나 이상의 칼럼에 의사 칼럼의 **DEFAULT** 값 지정이 가능하다.

```
CREATE TABLE tbl (date1 DATE DEFAULT SYSDATE, date2 DATE DEFAULT
SYSDATE);
CREATE TABLE tbl (date1 DATE DEFAULT SYSDATE,
ts1 TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
CREATE TABLE t1(id1 INT, id2 VARCHAR(20) DEFAULT
TO_CHAR(12345,'S999999'), id3 VARCHAR(20) DEFAULT TO_CHAR(SYS_TIME,
'HH24:MI:SS'));
ALTER TABLE t1 add column id4 varchar (20) default
TO_CHAR(SYS_DATETIME, 'yyyy/mm/dd hh:mi:ss'), id5 DATE DEFAULT
SYSDATE;
```

자동 증가

칼럼 값에 자동으로 일련 번호를 부여하기 위해 칼럼에 **AUTO_INCREMENT** 속성을 정의할 수 있다. **SMALLINT**, **INTEGER**, **BIGINT**, **NUMERIC** ($p, 0$) 타입에 한정하여 정의할 수 있다.

동일한 칼럼에 **AUTO_INCREMENT** 속성과 **SHARED** 또는 **DEFAULT** 속성을 동시에 정의할 수 없으며, 사용자가 직접 입력한 값과 자동 증가 특성에 의해 입력된 값이 서로 충돌되지 않도록 주의해야 한다.

AUTO_INCREMENT 의 초기값은 **ALTER TABLE** 문을 이용하여 바꿀 수 있다. 자세한 내용은 **ALTER TABLE** 의 **AUTO_INCREMENT 절** 을 참고한다.

```
CREATE TABLE [schema_name.]table_name (id INT AUTO_INCREMENT[(seed,
increment)]);
```

```
CREATE TABLE [schema_name.]table_name (id INT AUTO_INCREMENT)
AUTO_INCREMENT = seed ;
```

- *seed*: 번호가 시작하는 초기값이다. 모든 정수가 허용되며 기본값은 **1** 이다.
- *increment*: 행마다 증가되는 증가값이다. 양의 정수만 허용되며 기본값은 **1** 이다.

CREATE TABLE [schema_name.]table_name (id int **AUTO_INCREMENT**) **AUTO_INCREMENT** = seed; 구문을 사용할 때에는 다음과 같은 제약 사항이 있다.

- **AUTO_INCREMENT** 속성을 갖는 칼럼은 하나만 정의해야 한다.
- (seed, increment)와 **AUTO_INCREMENT** = seed 는 같이 사용하지 않는다.

```
CREATE TABLE auto_tbl (id INT AUTO_INCREMENT, name VARCHAR);
INSERT INTO auto_tbl VALUES (NULL, 'AAA'), (NULL, 'BBB'), (NULL,
'CCC');
INSERT INTO auto_tbl (name) VALUES ('DDD'), ('EEE');
SELECT * FROM auto_tbl;
```

```
      id  name
=====
      1  'AAA'
      2  'BBB'
      3  'CCC'
      4  'DDD'
      5  'EEE'
```

```
CREATE TABLE tbl (id INT AUTO_INCREMENT, val string) AUTO_INCREMENT
= 3;
INSERT INTO tbl VALUES (NULL, 'cubrid');

SELECT * FROM tbl;
```

```
      id  val
=====
      3  'cubrid'
```

```
CREATE TABLE t (id INT AUTO_INCREMENT, id2 int AUTO_INCREMENT)
AUTO_INCREMENT = 5;
```

ERROR: To avoid ambiguity, the AUTO_INCREMENT table option requires the table to have exactly one AUTO_INCREMENT column **and** no seed/increment specification.

```
CREATE TABLE t (i INT AUTO_INCREMENT(100, 2)) AUTO_INCREMENT = 3;
```

ERROR: To avoid ambiguity, the AUTO_INCREMENT table option requires the table to have exactly one AUTO_INCREMENT column **and** no seed/increment specification.

참고

- 자동 증가 특성만으로는 **UNIQUE** 제약 조건을 가지지 않는다.
- 자동 증가 특성이 정의된 칼럼에 **NULL** 을 입력하면 자동 증가된 값이 저장된다.
- 자동 증가 특성이 정의된 칼럼에 값을 직접 입력해도 AUTO_INCREMENT 값은 변하지 않는다.
- 자동 증가 특성이 정의된 칼럼에 **SHARED** 또는 **DEFAULT** 속성을 설정할 수 없다.
- 초기값 및 자동 증가 특성에 의해 증가된 최종 값은 해당 타입에서 허용되는 최소/최대값을 넘을 수 없다.
- 자동 증가 특성은 순환되지 않으므로 타입의 최대값을 넘어갈 경우 오류가 발생하며, 이에 대한 롤백이 일어나지 않는다. 따라서 이와 같은 경우 해당 칼럼을 삭제 후 다시 생성해야 한다.

예를 들어, 아래와 같이 테이블을 생성했다면, A의 최대값은 32767이다. 32767이 넘어가는 경우 에러가 발생하므로, 초기 테이블 생성시에 칼럼 A의 최대값이 해당 타입의 최대값을 넘지 않는다는 것을 감안해야 한다.

```
CREATE TABLE tb1(A SMALLINT AUTO_INCREMENT, B CHAR(5));
```

ON UPDATE

특정 테이블의 해당 열의 다른 속성값이 변경되면 자동으로 변경되는 속성을 추가할 수 있다. **ON UPDATE** 의 값은 **ALTER** 문을 통해 변경할 수 있다. 의사 컬럼은 다음과 같은 방법으로 **ON UPDATE** 의 값을 허용한다. 갱신되는 필드의 목록에 의사 컬럼이 포함되는 경우, 정해진 **ON UPDATE** 값으로의 수정되지 않는다.

기본값	데이터 타입
SYS_TIMESTAMP	TIMESTAMP
UNIX_TIMESTAMP()	INTEGER
CURRENT_TIMESTAMP	TIMESTAMP
SYS_DATETIME	DATETIME
CURRENT_DATETIME	DATETIME
SYS_DATE	DATE
CURRENT_DATE	DATE

```
CREATE TABLE sales (sales_cnt INTEGER, last_sale TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP, product VARCHAR(100), product_id INTEGER);
INSERT INTO sales VALUES (0, NULL, 'bicycle', 1);
```

```
UPDATE sales set sales_cnt = sales_cnt + 1
WHERE product_id = 1;
```

```
ALTER TABLE sales MODIFY last_sale TIMESTAMP; -- removes ON UPDATE
UPDATE sales set sales_cnt = sales_cnt + 1
WHERE product_id = 1; -- last_sale will remain unupdated
```

제약 조건 정의

제약 조건으로 **NOT NULL**, **UNIQUE**, **PRIMARY KEY**, **FOREIGN KEY** 를 정의할 수 있다. 또한 제약 조건은 아니지만 **INDEX** 또는 **KEY** 를 사용하여 인덱스를 생성할 수도 있다.

```
<column_constraint> ::= [CONSTRAINT constraint_name] { NOT NULL |
UNIQUE | PRIMARY KEY | [FOREIGN KEY] [WITH <index_with_option>]
<referential_definition> }
```

```
<table_constraint> ::=
```

```

{
    {KEY|INDEX} index_name <index_col_desc> |
    [CONSTRAINT [constraint_name]]
    {
        UNIQUE [KEY|INDEX](column_name, ...) |
        PRIMARY KEY (column_name, ...) |
        <referential_constraint>
    }
}

<referential_constraint> ::= FOREIGN KEY [<foreign_key_name>]
(column_name, ...) [WITH <index_with_option>]
<referential_definition>

<referential_definition> ::=
    REFERENCES [schema_name.]referenced_table_name
(column_name, ...) [<referential_triggered_action> ...]

<referential_triggered_action> ::=
    ON UPDATE <referential_action> |
    ON DELETE <referential_action>

<referential_action> ::= CASCADE | RESTRICT | NO
ACTION | SET NULL

<index_with_option> ::= {DEDUPLICATE '=' deduplicate_level}

```

NOT NULL 제약

NOT NULL 제약 조건이 정의된 칼럼은 반드시 **NULL** 이 아닌 값을 가져야 한다. 모든 칼럼에 대해 **NOT NULL** 제약 조건을 정의할 수 있다. **INSERT**, **UPDATE** 구문을 통해 **NOT NULL** 속성 칼럼에 **NULL** 값을 입력하거나 갱신하면 에러가 발생한다.

아래 예에서 *id* 칼럼은 **NULL** 값을 가질 수 없으므로, **INSERT** 문에서 *id* 칼럼에 **NULL**을 입력하면 오류가 발생한다.

```
CREATE TABLE const_tbl1(id INT NOT NULL, INDEX i_index(id ASC),
phone VARCHAR);
```

```
CREATE TABLE const_tbl2(id INT NOT NULL PRIMARY KEY, phone VARCHAR);
INSERT INTO const_tbl2 VALUES (NULL, '000-0000');
```

```
Putting value 'null' into attribute 'id' returned: Attribute "id"
cannot be made NULL.
```

UNIQUE 제약

UNIQUE 제약 조건은 정의된 칼럼이 고유한 값을 갖도록 하는 제약 조건이다. 기존 레코드와 동일한 칼럼 값을 갖는 레코드가 추가되면 에러가 발생한다.

UNIQUE 제약 조건은 단일 칼럼뿐만 아니라 하나 이상의 다중 칼럼에 대해서도 정의가 가능하다.

UNIQUE 제약 조건이 다중 칼럼에 대해 정의되면 각 칼럼 값에 대해 고유성이 보장되는 것이 아니라, 다중 칼럼 값의 조합에 대해 고유성이 보장된다.

아래 예에서 두번째 INSERT 문의 *id* 칼럼의 값은 첫번째 INSERT 문의 *id* 칼럼 값과 동일한 1이므로 오류가 발생한다.

```
-- UNIQUE constraint is defined on a single column only
CREATE TABLE const_tbl5(id INT UNIQUE, phone VARCHAR);
INSERT INTO const_tbl5(id) VALUES (NULL), (NULL);
INSERT INTO const_tbl5 VALUES (1, '000-0000');
SELECT * FROM const_tbl5;
```

```
id  phone
=====
NULL NULL
NULL NULL
  1  '000-0000'
```

```
INSERT INTO const_tbl5 VALUES (1, '111-1111');
```

```
ERROR: Operation would have caused one or more unique constraint
violations.
```

아래 예에서 **UNIQUE** 제약 조건이 다중 칼럼에 대해 정의되면 칼럼 전체 값의 조합에 대해 고유성이 보장된다.

```
-- UNIQUE constraint is defined on several columns
CREATE TABLE const_tbl6(id INT, phone VARCHAR, CONSTRAINT UNIQUE
(id, phone));
INSERT INTO const_tbl6 VALUES (1, NULL), (2, NULL), (1, '000-0000'),
(1, '111-1111');
SELECT * FROM const_tbl6;
```

```
id  phone
=====
  1  NULL
  2  NULL
```

```
1 '000-0000'
1 '111-1111'
```

PRIMARY KEY 제약

테이블에서 키(key)란 각 행을 고유하게 식별할 수 있는 하나 이상의 칼럼들의 집합을 말한다. 후보 키(candidate key)는 테이블 내의 각 행을 고유하게 식별하는 칼럼들의 집합을 의미하며, 사용자는 이러한 후보 키 중 하나를 기본키(primary key)로 정의할 수 있다. 즉, 기본키로 정의된 칼럼 값은 각 행에서 고유하게 식별된다.

기본키를 정의하여 생성되는 인덱스는 기본적으로 오름차순으로 생성되며, 칼럼 뒤에 **ASC** 또는 **DESC** 키워드를 명시하여 키의 순서를 지정할 수 있다.

```
CREATE TABLE pk_tbl (a INT, b INT, PRIMARY KEY (a, b DESC));

CREATE TABLE const_tbl7 (
    id INT NOT NULL,
    phone VARCHAR,
    CONSTRAINT pk_id PRIMARY KEY (id)
);

-- CONSTRAINT keyword
CREATE TABLE const_tbl8 (
    id INT NOT NULL PRIMARY KEY,
    phone VARCHAR
);

-- primary key is defined on multiple columns
CREATE TABLE const_tbl8 (
    host_year    INT NOT NULL,
    event_code   INT NOT NULL,
    athlete_code INT NOT NULL,
    medal        CHAR (1) NOT NULL,
    score        VARCHAR (20),
    unit         VARCHAR (5),
    PRIMARY KEY (host_year, event_code, athlete_code, medal)
);
```

FOREIGN KEY 제약

외래키(foreign key)란 참조 관계에 있는 다른 테이블의 기본키를 참조하는 칼럼 또는 칼럼들의 집합을 말한다. 외래키와 참조되는 기본키는 동일한 데이터 타입을 가져야 한다. 외래키가 기본키를 참조함에 따라 연관되는 두 테이블 사이에는 일관성이 유지되는데, 이를 참조 무결성(referential integrity)이라 한다.

```
[CONSTRAINT constraint_name] FOREIGN KEY [foreign_key_name]
(<column_name_comma_list1>) [WITH <index_with_option>] REFERENCES
[schema_name.]referenced_table_name (<column_name_comma_list2>)
[<referential_triggered_action> ...]

    <referential_triggered_action> ::= ON UPDATE
<referential_action> | ON DELETE <referential_action>

    <referential_action> ::= CASCADE | RESTRICT | NO ACTION |
SET NULL

    <index_with_option> ::= {DEDUPLICATE '=' deduplicate_level}
```

- *constraint_name*: 제약 조건의 이름을 지정한다.
- *foreign_key_name*: **FOREIGN KEY** 제약 조건의 이름을 지정한다. 생략할 수 있으며, 이 값을 지정하면 *constraint_name* 을 무시하고 이 이름을 사용한다.
- *<column_name_comma_list1>*: **FOREIGN KEY** 키워드 뒤에 외래키로 정의하고자 하는 칼럼 이름을 명시한다. 정의되는 외래키의 칼럼 개수는 참조되는 기본키의 칼럼 개수와 동일해야 한다.
- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *referenced_table_name*: 참조되는 테이블의 이름을 지정한다.
- *<column_name_comma_list2>*: **REFERENCES** 키워드 뒤에 참조되는 기본키 칼럼 이름을 지정한다.
- *<referential_triggered_action>*: 참조 무결성이 유지되도록 특정 연산에 따라 대응하는 트리거 동작을 정의하는 것이며, **ON UPDATE**, **ON DELETE** 가 올 수 있다. 각각의 동작은 중복하여 정의 가능하며, 정의 순서는 무관하다.
 - **ON UPDATE**: 외래키가 참조하는 기본키 값을 갱신하려 할 때 수행할 작업을 정의한다. 사용자는 **NO ACTION**, **RESTRICT**, **SET NULL** 중 하나의 옵션을 지정할 수 있으며, 기본은 **RESTRICT** 이다.
 - **ON DELETE**: 외래키가 참조하는 기본키 값을 삭제하려 할 때 수행할 작업을 정의한다. 사용자는 **NO ACTION**, **RESTRICT**, **CASCADE**, **SET NULL** 중 하나의 옵션을 지정할 수 있으며, 기본은 **RESTRICT** 이다.
- *<referential_action>*: 기본키 값이 삭제 또는 갱신될 때 이를 참조하는 외래키의 값을 유지할 것인지 또는 변경할 것인지 지정할 수 있다.
 - **CASCADE**: 기본키가 삭제되면 외래키도 삭제한다. **ON DELETE** 연산에 대해서만 지원된다.
 - **RESTRICT**: 기본키 값이 삭제되거나 업데이트되지 않도록 제한한다. 삭제 또는 업데이트를 시도하는 트랜잭션은 롤백된다.

- **SET NULL**: 기본키가 삭제되거나 업데이트되면, 이를 참조하는 외래키 컬럼 값을 **NULL** 로 업데이트한다.

- **NO ACTION**: **RESTRICT** 옵션과 동일하게 동작한다.

- *deduplicate_level*: deduplicate 레벨을 지정한다(0 ~ 14). 자세한 내용은 **DEDUPLICATE**를 참고한다.

참조하는 테이블의 각 R1 행에 대해 참조되는 테이블의 R2 행이 있어야 하며, R1의 참조하는 각 컬럼의 값이 **NULL** 이거나 R2의 참조되는 해당 컬럼의 값과 동일해야 한다.

```
-- creating two tables where one is referencing the other
CREATE TABLE a_tbl (
    id INT NOT NULL DEFAULT 0 PRIMARY KEY,
    phone VARCHAR(10)
);

CREATE TABLE b_tbl (
    id INT NOT NULL,
    name VARCHAR(10) NOT NULL,
    CONSTRAINT pk_id PRIMARY KEY (id),
    CONSTRAINT fk_id FOREIGN KEY (id) REFERENCES a_tbl (id)
    ON DELETE CASCADE ON UPDATE RESTRICT
);

INSERT INTO a_tbl VALUES (1,'111-1111'), (2,'222-2222'), (3,
'333-3333');
INSERT INTO b_tbl VALUES (1,'George'),(2,'Laura'), (3,'Max');
SELECT a.id, b.id, a.phone, b.name FROM a_tbl a, b_tbl b WHERE a.id
= b.id;
```

id	id	phone	name
1	1	'111-1111'	'George'
2	2	'222-2222'	'Laura'
3	3	'333-3333'	'Max'

```
-- when deleting primary key value, it cascades foreign key value
DELETE FROM a_tbl WHERE id=3;
```

1 row affected.

```
SELECT a.id, b.id, a.phone, b.name FROM a_tbl a, b_tbl b WHERE a.id
= b.id;
```

id	id	phone	name
=====	=====	=====	=====
=			
1	1	'111-1111'	'George'
2	2	'222-2222'	'Laura'

```
-- when attempting to update primary key value, it restricts the
operation
```

```
UPDATE a_tbl SET id = 10 WHERE phone = '111-1111';
```

```
ERROR: Update/Delete operations are restricted by the foreign key
'fk_id'.
```

참고

- 참조 제약 조건에는 참조 대상이 되는 기본키 테이블의 이름 및 기본키와 일치하는 칼럼명들이 정의된다. 만약, 칼럼명 목록을 지정하지 않을 경우에는 기본키 테이블의 기본키가 원래 지정된 순서대로 지정된다.
- 참조 제약 조건의 기본키의 개수는 외래키의 개수와 동일해야 한다. 참조 제약 조건의 기본키는 동일한 칼럼명이 중복될 수 없다.
- 참조 제약 조건에 의해 CASCADE되는 작업은 트리거의 동작을 활성화하지 않는다.
- CUBRID HA 환경에서는 *referential_triggered_action* 을 사용하지 않는 것을 권장한다. CUBRID HA 환경에서는 트리거를 지원하지 않으므로, *referential_triggered_action* 을 사용하면 마스터 데이터베이스와 슬레이브 데이터베이스의 데이터가 일치하지 않을 수 있다. 자세한 내용은 CUBRID HA 를 참고한다.

KEY 또는 INDEX

KEY 와 **INDEX** 는 동일하며, 해당 칼럼을 키로 하는 인덱스를 생성한다.

```
CREATE TABLE const_tbl4(id INT, phone VARCHAR, KEY i_key(id DESC,
phone ASC));
```

참고

CUBRID 9.0 미만 버전에서는 인덱스 이름을 생략할 수 있었으나, CUBRID 9.0 버전부터는 인덱스 이름을 생략할 수 없다.

칼럼 옵션

특정 칼럼에 **UNIQUE** 또는 **INDEX** 를 정의할 때, 해당 칼럼 이름 뒤에 **ASC** 또는 **DESC** 옵션을 명시할 수 있다. 이 키워드는 오름차순 또는 내림차순 인덱스 값 저장을 위해 명시된다.

```
column_name [ASC | DESC]
```

```
CREATE TABLE const_tbl(
    id VARCHAR,
    name VARCHAR,
    CONSTRAINT UNIQUE INDEX(id DESC, name ASC)
);

INSERT INTO const_tbl VALUES('1000', 'john'), ('1000','johnny'),
('1000', 'jone');
INSERT INTO const_tbl VALUES('1001', 'johnny'), ('1001','john'),
('1001', 'jone');

SELECT * FROM const_tbl WHERE id > '100';
```

```
id      name
=====
1001    john
1001    johnny
1001    jone
1000    john
1000    johnny
1000    jone
```

테이블 옵션

테이블 옵션 중 **REUSE_OID** 와 **DONT_REUSE_OID** 은 생성하는 테이블이 참조 가능한 테이블인지 아닌지를 지정하는 옵션이다. 두개의 옵션은 함께 사용할 수 없으며 다른 옵션들과는 함께 사용할 수 있다. 테이블 생성시 옵션을 생략한 경우에는 **REUSE_OID** 테이블 옵션을 사용한다. 기본 옵션을 **DONT_REUSE_OID** 로 변경하려면, 시스템 파라미터인 **create_table_reuseoid** 값을 **no** 로 변경하면 된다. 자세한 내용은 [구문/타입 관련 파라미터](#) 를 참조하면 된다.

```
<table_options> ::= <table_option> [[,] <table_option> ...]
<table_option> ::= REUSE_OID | DONT_REUSE_OID |
```



```
COMMENT [=] 'table_comment_string' |
[CHARSET charset_name] [COLLATE
collation_name]
```

REUSE_OID

테이블 생성 시 **REUSE_OID** 옵션을 명시하면, 레코드 삭제(**DELETE**)로 인해 삭제된 OID를 새로운 레코드 삽입(**INSERT**) 시 재사용할 수 있다. **REUSE_OID** 옵션을 명시하여 생성된 테이블을 OID 재사용 테이블 또는 참조 불가능(non-referable)한 테이블이라고 한다.

OID(Object Identifier)는 블록 번호, 페이지 번호, 슬롯 번호와 같은 물리적 위치 정보로 표현되는 객체 식별자이다. CUBRID는 OID를 이용하여 객체의 참조 관계를 관리하고, 객체 조회, 저장, 삭제를 수행한다. OID를 이용하면 테이블을 참조하지 않고도 힙 파일 내의 해당 오브젝트에 직접 접근할 수 있어 접근성이 향상되지만, 객체가 삭제되더라도 참조 관계를 유지하기 위해 해당 객체의 OID를 보존하기 때문에 **DELETE** / **INSERT** 연산이 많은 경우 저장 공간 재사용률이 저하되는 문제가 있다.

테이블 생성 시 **REUSE_OID** 옵션을 명시하면, 해당 테이블 내의 데이터 삭제 시 해당 OID가 함께 삭제되며, **INSERT** 된 다른 데이터가 해당 OID를 재사용할 수 있다. 단, OID 재사용 테이블을 다른 테이블이 참조할 수 없고, OID 재사용 테이블 내 객체들의 OID 값을 조회할 수 없다.

```
-- creating table with REUSE_OID option specified
CREATE TABLE reuse_tbl (a INT PRIMARY KEY) REUSE_OID, COMMENT =
'reuse oid table';
INSERT INTO reuse_tbl VALUES (1);
INSERT INTO reuse_tbl VALUES (2);
INSERT INTO reuse_tbl VALUES (3);

-- an error occurs when column type is a OID reusable table itself
CREATE TABLE tbl_1 (a reuse_tbl);
```

```
ERROR: The class 'reuse_tbl' is marked as REUSE_OID and is non-
referable. Non-referable classes can't be the domain of an attribute
and their instances' OIDs cannot be returned.
```

테이블의 콜레이션과 같이 지정하는 경우 **REUSE_OID**를 콜레이션 앞 또는 뒤에 지정할 수 있다.

```
CREATE TABLE t3(a VARCHAR(20)) REUSE_OID, COMMENT = 'reuse oid
table', COLLATE euckr_bin;
CREATE TABLE t4(a VARCHAR(20)) COLLATE euckr_bin REUSE_OID;
```

참고

- 다른 테이블이 OID 재사용 테이블을 참조할 수 없다.
- OID 재사용 테이블에 대해 갱신 가능한(updatable) 뷰를 생성할 수 없다.
- 테이블의 컬럼 타입으로 OID 재사용 테이블을 지정할 수 없다.
- OID 재사용 테이블 객체들의 OID 값을 읽을 수 없다.
- OID 재사용 테이블에서 인스턴스 메서드를 호출할 수 없다. 메서드가 정의된 클래스를 상속받은 서브클래스가 OID 재사용 테이블로 정의되어도 마찬가지로 인스턴스 메서드를 호출할 수 없다.
- OID 재사용 테이블은 CUBRID 2008 R2.2 버전 이상에서만 지원되며, 하위 호환성을 보장하지 않는다. 즉, 더 낮은 버전의 데이터베이스 서버에서 OID 재사용 테이블이 존재하는 데이터베이스에 접근할 수 없다.
- OID 재사용 테이블은 분할 테이블로 관리될 수 있으며, 복제될 수 있다.
- OID 재사용 테이블이 포함된 뷰의 업데이트 질의는 아래와 같은 오류가 발생한다.

```
CREATE TABLE t1 (c1 INT) REUSE_OID;
CREATE VIEW v3 AS SELECT c1 FROM t1;

insert into v3(c1) values (1);
```

```
ERROR: Vclass v3 is not updatable. Please check if any of its
related classes are marked as REUSE_OID.
```

DONT_REUSE_OID

테이블 생성시 **DONT_REUSE_OID** 옵션을 명시하면, **REUSE_OID** 와 상반된 참조 가능(referable)한 테이블을 생성한다.

문자셋과 컬레이션

해당 테이블과 컬럼에 적용할 문자셋과 컬레이션을 **CREATE TABLE** 문에 명시할 수 있다. 다음 예시는 CREATE TABLE시 문자셋과 컬레이션을 지정하는 예문이다.

```
CREATE TABLE tblutf8(a int, b int) charset utf8;
CREATE TABLE tblutf8enci(a int, b int) COLLATE utf8_en_ci;

CREATE TABLE tbleuckr(a int, b int) charset euckr COLLATE euckr_bin;
CREATE TABLE tblcoleuckr(a int, c varchar(100) COLLATE euckr_bin)
charset euckr;

CREATE TABLE tblcol (s1 STRING COLLATE utf8_en_cs,s2 STRING COLLATE
utf8_bin);
CREATE TABLE tblcharcol (col STRING CHARSET utf8) COLLATE utf8_en_ci;
```

이에 관한 자세한 내용은 [테이블의 문자셋과 콜레이션](#) 또는 [문자열 리터럴의 문자셋과 콜레이션](#) 절을 참조하면 된다.

테이블의 커멘트

테이블의 커멘트를 다음과 같이 명시할 수 있다.

```
CREATE TABLE tbl (a INT, b INT) COMMENT = 'this is comment for table
tbl';
```

테이블의 커멘트는 다음 구문에서 확인할 수 있다.

```
SHOW CREATE TABLE table_name;
SELECT class_name, comment from db_class;
SELECT class_name, comment from _db_class;
```

또는 CSQL 인터프리터에서 테이블의 스키마를 출력하는 ;sc 명령으로 테이블의 커멘트를 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc tbl
```

테이블 암호화 (TDE)

다음과 같이 테이블을 암호화할 수 있다. TDE 암호화에 관한 자세한 내용은 [TDE \(Transparent Data Encryption\)](#) 절을 참고한다.

```
CREATE TABLE enc_tbl (a INT, b INT) ENCRYPT = AES;
```

암호화 알고리즘으로 **AES**, **ARIA** 를 지정할 수 있다. 다음과 같이 생략할 경우 시스템 파라미터 **tde_default_algorithm** 으로 지정된 암호화 알고리즘이 사용 된다. 기본 값은 **AES** 이다.

```
CREATE TABLE enc_tbl (a INT, b INT) ENCRYPT;
```

암호화 여부는 상속되지 않는다.

CREATE TABLE LIKE

CREATE TABLE ... LIKE 문을 사용하면, 이미 존재하는 테이블의 스키마와 동일한 스키마를 갖는 테이블을 생성할 수 있다. 기존 테이블에서 정의된 칼럼 속성, 테이블 제약 조건, 암호화 여부, 인덱스도 그대로 복제된다. 원본 테이블에서 자동 생성된 인덱스의 이름은 새로 생성된 테이블의 이름에 맞게 새로 생성되지만, 사용자에게 의해 지어진 인덱스 이름은 그대로 복제된다. 그러므로 인덱스 힌트 구문 ([인덱스 힌트 참고](#))으로 특정 인덱스를 사용하도록 작성된 질의문이 있다면 주의해야 한다.

CREATE TABLE ... LIKE 문은 스키마만 복제하므로 칼럼 정의문을 작성할 수 없다.

```
CREATE {TABLE | CLASS} [schema_name.]new_table_name LIKE
[schema_name.]source_table_name;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *new_table_name*: 새로 생성할 테이블 이름이다.
- *source_table_name*: 데이터베이스에 이미 존재하는 원본 테이블 이름이다. **CREATE TABLE ... LIKE** 문에서 아래의 테이블은 원본 테이블로 지정될 수 없다.
 - 분할 테이블
 - **AUTO_INCREMENT** 칼럼이 포함된 테이블
 - 상속 또는 메서드를 사용하는 테이블

```
CREATE TABLE a_tbl (
  id INT NOT NULL DEFAULT 0 PRIMARY KEY,
  phone VARCHAR(10)
);
INSERT INTO a_tbl VALUES (1,'111-1111'), (2,'222-2222'), (3,
'333-3333');

-- creating an empty table with the same schema as a_tbl
CREATE TABLE new_tbl LIKE a_tbl;
SELECT * FROM new_tbl;
```

There are no results.

```
csql> ;schema a_tbl
```

```
=== <Help: Schema of a Class> ===
```

```
<Class Name>
```

```
a_tbl
```

```
<Attributes>
```

```
id          INTEGER DEFAULT 0 NOT NULL
phone       CHARACTER VARYING(10)
```

```
<Constraints>
```

```
PRIMARY KEY pk_a_tbl_id ON a_tbl (id)
```

```
csql> ;schema new_tbl
```

```
=== <Help: Schema of a Class> ===
```

```
<Class Name>
```

```
new_tbl
```

```
<Attributes>
```

```
id          INTEGER DEFAULT 0 NOT NULL
phone       CHARACTER VARYING(10)
```

```
<Constraints>
```

```
PRIMARY KEY pk_new_tbl_id ON new_tbl (id)
```

CREATE TABLE AS SELECT

CREATE TABLE ... AS SELECT 문을 사용하여 **SELECT** 문의 결과 레코드를 포함하는 새로운 테이블을 생성할 수 있다. 새로운 테이블에 대해 칼럼 및 테이블 제약 조건을 정의할 수 있으며, 다음의 규칙을 적용하여 **SELECT** 결과 레코드를 반영한다.

- 새로운 테이블에 칼럼 *col_1* 이 정의되고, *select_statement* 에 동일한 칼럼 *col_1* 이 명시된 경우, **SELECT** 결과 레코드가 새로운 테이블 *col_1* 값으로 저장된다. 칼럼 이름은 같고 칼럼 타입이 다른 타입 변환을 시도한다.

- 새로운 테이블에 칼럼 *col_1*, *col_2* 가 정의되고, *select_statement* 의 칼럼 리스트에 *col_1*, *col_2*, *col_3* 이 명시되어 모두 포함 관계가 성립하는 경우, 새로 생성되는 테이블에는 *col_1*, *col_2*, *col_3* 이 생성되고, **SELECT** 결과 데이터가 모든 칼럼 값으로 저장된다. 칼럼 이름은 같고 칼럼 타입이 다르면 타입 변환을 시도한다.
- 새로운 테이블에 칼럼 *col_1*, *col_2* 가 정의되고, *select_statement* 의 칼럼 리스트에 *col_1*, *col_3* 이 명시되어 포함 관계가 성립하지 않는 경우, 새로 생성되는 테이블에는 *col_1*, *col_2*, *col_3* 이 생성되고, *select_statement* 에 명시된 칼럼 *col_1*, *col_3* 에 대해서만 **SELECT** 결과 데이터가 저장되고, *col_2* 에는 NULL이 저장된다.
- *select_statement* 의 칼럼 리스트에는 칼럼 별칭(alias)이 포함될 수 있으며, 이 경우 칼럼 별칭이 새로운 테이블 칼럼 이름으로 사용된다. 함수 호출이나 표현식이 사용된 경우 별칭이 없으면 유효하지 않은 칼럼 이름이 생성되므로, 이 경우에는 별칭을 사용하는 것이 좋다.
- **REPLACE** 옵션은 새로운 테이블의 칼럼(*col_1*)에 **UNIQUE** 제약 조건이 정의된 경우에만 유효하다. *select_statement* 의 결과 레코드에 중복된 값이 존재하는 경우, **REPLACE** 옵션이 명시되면 칼럼 *col_1* 에는 고유한 값이 저장되고, **REPLACE** 옵션이 생략되면 **UNIQUE** 제약 조건에 위배되므로 에러 메시지가 출력된다.

```
CREATE {TABLE | CLASS} [schema_name.]table_name
[( [{<table_constraint>}... ,] <column_definition> [{ ,
<column_definition> | <table_constraint>}...])] [COMMENT [=]
'comment_string'] [REPLACE] AS <select_statement>;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 새로 생성할 테이블 이름이다.
- *<column_definition>*: 칼럼을 정의한다. 생략하면 **SELECT** 문의 칼럼 스키마가 복제된다. **SELECT** 문의 칼럼 제약 조건이나 **AUTO_INCREMENT** 속성, 테이블/칼럼의 커멘트는 복제되지 않는다.
- *<table_constraint>*: 테이블 제약 조건을 정의한다.
- *<select_statement>*: 데이터베이스에 이미 존재하는 원본 테이블을 대상으로 하는 **SELECT** 문이다.
- *<select statement>*은 tbl@srver1처럼 원격 테이블을 포함할 수 있다.

```
CREATE TABLE a_tbl (
  id INT NOT NULL DEFAULT 0 PRIMARY KEY,
  phone VARCHAR(10)
);
INSERT INTO a_tbl VALUES (1,'111-1111'), (2,'222-2222'), (3,
'333-3333');

-- creating a table without column definition
```

```
CREATE TABLE new_tbl1 AS SELECT * FROM a_tbl;
SELECT * FROM new_tbl1;
```

```

id  phone
=====
1   '111-1111'
2   '222-2222'
3   '333-3333'
```

```
-- all of column values are replicated from a_tbl
CREATE TABLE new_tbl2 (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  phone VARCHAR
) AS SELECT * FROM a_tbl;

SELECT * FROM new_tbl2;
```

```

id  phone
=====
1   '111-1111'
2   '222-2222'
3   '333-3333'
```

```
-- some of column values are replicated from a_tbl and the rest is
NULL
CREATE TABLE new_tbl3 (
  id INT,
  name VARCHAR
) AS SELECT id, phone FROM a_tbl;

SELECT * FROM new_tbl3
```

```

name                                id  phone
=====
NULL                                1   '111-1111'
NULL                                2   '222-2222'
NULL                                3   '333-3333'
```

```
-- column alias in the select statement should be used in the column
definition
CREATE TABLE new_tbl4 (
  id1 INT,
  id2 INT
```

```
) AS SELECT t1.id id1, t2.id id2 FROM new_tbl1 t1, new_tbl2 t2;

SELECT * FROM new_tbl4;
```

id1	id2
1	1
1	2
1	3
2	1
2	2
2	3
3	1
3	2
3	3

```
-- REPLACE is used on the UNIQUE column
CREATE TABLE new_tbl5 (id1 int UNIQUE) REPLACE AS SELECT * FROM
new_tbl4;

SELECT * FROM new_tbl5;
```

id1	id2
1	3
2	3
3	3

아래와 같이 원격 테이블로 부터 데이터를 읽어서 테이블을 만들 수 있다.

```
-- column values are replicated from remote_tbl
CREATE TABLE new_tbl_rem (
  id INT NON NULL;
  phone VARCHAR;
  location VARCHAR;
) AS SELECT id, phone, 'remote' FROM a_tbl@server1 WHERE id < 3;

SELECT * FROM new_tbl_rem;
```

id	phone	location
1	'111-1111'	'remote'
2	'222-2222'	'remote'

ALTER TABLE

ALTER 구문을 이용하여 테이블의 구조를 변경할 수 있다. 대상 테이블에 칼럼 추가/삭제, 인덱스 생성/삭제, 기존 칼럼의 타입 변경, 테이블 이름 변경, 칼럼 이름 변경 등을 수행하거나 테이블 제약 조건을 변경한다. 또한 **AUTO_INCREMENT**의 초기값을 변경할 수 있다. **TABLE**은 **CLASS**와 동의어이다. **COLUMN**은 **ATTRIBUTE**와 동의어이다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name <alter_clause> [,
<alter_clause>] ... ;

<alter_clause> ::=
    ADD <alter_add> [INHERIT <resolution>, ...] |
    ADD {KEY | INDEX} <index_name> <index_col_desc> |
    ALTER [COLUMN] column_name SET DEFAULT <value_specification>
|
    DROP <alter_drop> [INHERIT <resolution>, ...] |
    DROP {KEY | INDEX} index_name |
    DROP FOREIGN KEY constraint_name |
    DROP PRIMARY KEY |
    RENAME <alter_rename> [INHERIT <resolution>, ...] |
    CHANGE <alter_change> |
    MODIFY <alter_modify> |
    INHERIT <resolution>, ... |
    AUTO_INCREMENT = <initial_value> |
    COMMENT [=] 'table_comment_string' |
    COMMENT ON {COLUMN | CLASS ATTRIBUTE}
<column_comment_definition> [, <column_comment_definition>] ;

<alter_add> ::=
    [ATTRIBUTE | COLUMN] [( <class_element>, ...[ ] ) [FIRST |
AFTER old_column_name] |
    CLASS ATTRIBUTE <column_definition>, ... |
    CONSTRAINT <constraint_name> <column_constraint>
(column_name) |
    QUERY <select_statement> |
    SUPERCLASS [schema_name.]superclass_name, ...

    <class_element> ::= <column_definition> |
<table_constraint>

    <column_constraint> ::= UNIQUE [KEY] | PRIMARY KEY |
FOREIGN KEY

    <alter_drop> ::=
    [ATTRIBUTE | COLUMN]
    {
        column_name, ... |
        QUERY [<unsigned_integer_literal>] |
        SUPERCLASS [schema_name.]superclass_name, ... |
        CONSTRAINT constraint_name
    }
```

```

<alter_rename> ::=
    [ATTRIBUTE | COLUMN]
    {
        old_column_name AS new_column_name |
        FUNCTION OF column_name AS function_name
    }

<alter_change> ::=
    [COLUMN | CLASS ATTRIBUTE] old_col_name new_col_name
<column_definition>
    [FIRST | AFTER col_name]

<alter_modify> ::=
    [COLUMN | CLASS ATTRIBUTE] col_name <column_definition>
    [FIRST | AFTER col_name2]

<table_option> ::=
    CHANGE [COLUMN | CLASS ATTRIBUTE] old_col_name
    new_col_name <column_definition>
    [FIRST | AFTER col_name2]
    | MODIFY [COLUMN | CLASS ATTRIBUTE] col_name
    <column_definition>
    [FIRST | AFTER col_name2]

<resolution> ::= column_name OF
[schema_name.]superclass_name [AS alias]

<index_col_name> ::= column_name [(length)] [ASC | DESC]

<column_comment_definition> ::= column_name [=]
'column_comment_string'

```

참고

칼럼의 커멘트는 <column_definition>에서 지정하거나 <column_comment_definition>에서 지정한다. <column_definition>은 위의 **CREATE TABLE 문법**을 참고한다.

경고

테이블의 소유자, **DBA**, **DBA**의 멤버만이 테이블 스키마를 변경할 수 있으며, 그 밖의 사용자는 소유자나 **DBA**로부터 이름을 변경할 수 있는 권한을 받아야 한다(권한 관련 사항은 **GRANT** 참조)

ADD COLUMN 절

ADD COLUMN 절을 사용하여 새로운 칼럼을 추가할 수 있다. **FIRST** 또는 **AFTER** 키워드를 사용하여 새로 추가할 칼럼의 위치를 지정할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
ADD [COLUMN | ATTRIBUTE] [(] <column_definition> [FIRST | AFTER
old_column_name] [)];

<column_definition> ::=
    column_name <data_type> [[<default_or_shared_or_ai>] |
    [<on_update>] | [<column_constraint>]] [COMMENT 'comment_string']

    <data_type> ::= <column_type> [<charset_modifier_clause>]
    [<collation_modifier_clause>]

    <charset_modifier_clause> ::= {CHARACTER_SET|CHARSET}
    {<char_string_literal>|<identifier>}

    <collation_modifier_clause> ::= COLLATE
    {<char_string_literal>|<identifier>}

    <default_or_shared_or_ai> ::=
        SHARED <value_specification> |
        DEFAULT <value_specification> |
        AUTO_INCREMENT [(seed, increment)]

    <on_update> ::= [ON UPDATE <value_specification>]

    <column_constraint> ::= [CONSTRAINT constraint_name] {NOT
    NULL | UNIQUE | PRIMARY KEY | [FOREIGN KEY] [WITH
    <index_with_option>] <referential_definition>}

    <referential_definition> ::=
        REFERENCES [schema_name.]referenced_table_name
    (column_name, ...) [<referential_triggered_action> ...]

    <referential_triggered_action> ::=
        ON UPDATE <referential_action> |
        ON DELETE <referential_action>

    <referential_action> ::= CASCADE | RESTRICT | NO
    ACTION | SET NULL

    <index_with_option> ::= {DEDUPLICATE '='
    deduplicate_level}
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 칼럼을 추가할 테이블의 이름을 지정한다.

- `<column_definition>`: 새로 추가할 칼럼의 이름(최대 254 바이트), 데이터 타입, 제약 조건을 정의한다.
- **AFTER** `old_column_name`: 새로 추가할 칼럼 앞에 위치하는 기존 칼럼 이름을 명시한다.
- `comment_string`: 칼럼의 커멘트를 지정한다.
- `deduplicate_level`: deduplicate 레벨을 지정한다(0 ~ 14). 자세한 내용은 **DEDUPLICATE**를 참고한다.

```
CREATE TABLE a_tbl;
ALTER TABLE a_tbl ADD COLUMN age INT DEFAULT 0 NOT NULL COMMENT 'age
comment';
ALTER TABLE a_tbl ADD COLUMN name VARCHAR FIRST;
ALTER TABLE a_tbl ADD COLUMN id INT NOT NULL AUTO_INCREMENT UNIQUE
FIRST;
INSERT INTO a_tbl(age) VALUES(20),(30),(40);

ALTER TABLE a_tbl ADD COLUMN phone VARCHAR(13) DEFAULT
'000-0000-0000' AFTER name;
ALTER TABLE a_tbl ADD COLUMN birthday VARCHAR(20) DEFAULT
TO_CHAR(SYSDATE, 'YYYY-MM-DD');
SELECT * FROM a_tbl;
```

id	name	phone	age
1	NULL	'000-0000-0000'	20
2	NULL	'000-0000-0000'	30
3	NULL	'000-0000-0000'	40

```
--adding multiple columns
ALTER TABLE a_tbl ADD COLUMN (age1 int, age2 int, age3 int);
```

새로 추가되는 칼럼에 어떤 제약 조건이 오느냐에 따라 다른 결과를 보여준다.

- 새로 추가되는 칼럼에 **DEFAULT** 제약 조건이 있으면 **DEFAULT** 값이 입력된다.
- 새로 추가되는 칼럼에 **DEFAULT** 제약 조건이 없고 **NOT NULL** 제약 조건이 있는 경우, 시스템 파라미터 **add_column_update_hard_default** 가 **yes** 이면 고정 기본값(hard default)을 갖게 되고, **no** 이면 에러를 반환한다.

add_column_update_hard_default 의 기본값은 **no** 이다.

DEFAULT 제약 조건 및 **add_column_update_hard_default** 값의 설정에 따라 해당 제약 조건을 위배하지 않는 한도 내에서 **PRIMARY KEY** 혹은 **UNIQUE** 제약 조건의 추가가 가능하다.

- 테이블에 데이터가 없거나 **NOT NULL** 이고 **UNIQUE** 인 값을 가지는 기존 칼럼에 **PRIMARY KEY** 제약 조건을 지정할 수 있다.
- 테이블에 데이터가 있고 새로 추가되는 칼럼에 **PRIMARY KEY** 제약 조건을 지정하는 경우, 에러를 반환한다.

```
CREATE TABLE tbl (a INT);
INSERT INTO tbl VALUES (1), (2);
ALTER TABLE tbl ADD COLUMN (b int PRIMARY KEY);
```

```
ERROR: NOT NULL constraints do not allow NULL value.
```

- 테이블에 데이터가 있고 새로 추가되는 칼럼에 **UNIQUE** 제약 조건을 지정하는 경우, **DEFAULT** 제약 조건이 없으면 **NULL**이 입력된다.

```
ALTER TABLE tbl ADD COLUMN (b int UNIQUE);
SELECT * FROM tbl;
```

a	b
1	NULL
2	NULL

- 테이블에 데이터가 있고 새로 추가되는 칼럼에 **UNIQUE** 제약 조건을 지정하는 경우, **DEFAULT** 제약 조건이 있으면 고유 키 위반 에러를 반환한다.

```
ALTER TABLE tbl ADD COLUMN (c int UNIQUE DEFAULT 10);
```

```
ERROR: Operation would have caused one or more unique constraint violations.
```

- 테이블에 데이터가 있고 새로 추가되는 칼럼에 **UNIQUE** 제약 조건을 지정하는 경우, **NOT NULL** 제약 조건이 있고 **add_column_update_hard_default**가 **yes**이면 고유 키 위반 에러를 반환한다.

```
SET SYSTEM PARAMETERS 'add_column_update_hard_default=yes';
ALTER TABLE tbl ADD COLUMN (c int UNIQUE NOT NULL);
```

```
ERROR: Operation would have caused one or more unique constraint violations.
```

add_column_update_hard_default 및 고정 기본값에 대해서는 **CHANGE/MODIFY 절** 을 참고한다.

ADD CONSTRAINT 절

ADD CONSTRAINT 절을 사용하여 새로운 제약 조건을 추가할 수 있다.

PRIMARY KEY 제약 조건을 추가할 때 생성되는 인덱스는 기본적으로 오름차순으로 생성되며, 칼럼 이름 뒤에 **ASC** 또는 **DESC** 키워드를 명시하여 키의 정렬 순서를 지정할 수 있다.

```
ALTER [TABLE | CLASS | VCLASS | VIEW] [schema_name.]table_name
ADD <table_constraint> ;

<table_constraint> ::=
{
    {KEY|INDEX} index_name (column_name, ...) |
    [CONSTRAINT [constraint_name]]
    {
        UNIQUE [KEY|INDEX](column_name, ...) |
        PRIMARY KEY (column_name, ...) |
        <referential_constraint>
    }
}

<referential_constraint> ::= FOREIGN KEY [foreign_key_name]
(column_name, ...) <referential_definition>

<referential_definition> ::=
REFERENCES [schema_name.]referenced_table_name
(column_name, ...) [<referential_triggered_action> ...]

<referential_triggered_action> ::=
    ON UPDATE <referential_action> |
    ON DELETE <referential_action>

<referential_action> ::= CASCADE | RESTRICT | NO
ACTION | SET NULL
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 제약 조건을 추가할 테이블의 이름을 지정한다.
- *constraint_name*: 새로 추가할 제약 조건의 이름(최대 254 바이트)을 지정할 수 있으며, 생략할 수 있다. 생략하면 자동으로 부여된다.

- *foreign_key_name*: **FOREIGN KEY** 제약 조건의 이름을 지정할 수 있다. 생략할 수 있으며, 지정하면 *constraint_name* 을 무시하고 이 이름을 사용한다.
- *<table_constraint>*: 지정된 테이블에 대해 제약 조건을 정의한다. 제약 조건에 대한 자세한 설명은 **ON UPDATE** 를 참고한다.

```
ALTER TABLE a_tbl ADD CONSTRAINT pk_a_tbl_id PRIMARY KEY(id);
ALTER TABLE a_tbl DROP CONSTRAINT pk_a_tbl_id;
ALTER TABLE a_tbl ADD CONSTRAINT pk_a_tbl_id PRIMARY KEY(id, name
DESC);
ALTER TABLE a_tbl ADD CONSTRAINT u_key1 UNIQUE (id);
```

ADD INDEX 절

ADD INDEX 절은 특정 칼럼에 대해 인덱스 속성을 추가로 정의할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name ADD {KEY | INDEX}
index_name (<index_col_name>) ;

<index_col_name> ::= column_name [(length)] [ ASC | DESC ]
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 변경하고자 하는 테이블의 이름을 지정한다.
- *index_name*: 인덱스의 이름을 지정한다(최대 254 바이트).
- *index_col_name*: 인덱스를 정의할 대상 칼럼을 지정하며, 이때 칼럼 옵션으로 **ASC** 또는 **DESC** 을 함께 지정할 수 있다.

```
ALTER TABLE a_tbl ADD INDEX i1(age ASC), ADD INDEX i2(phone DESC);
```

```
csql> ;schema a_tbl

=== <Help: Schema of a Class> ===

<Class Name>

    a_tbl

<Attributes>

    name          CHARACTER VARYING(1073741823) DEFAULT ''
    phone          CHARACTER VARYING(13)   DEFAULT '111-1111'
    age            INTEGER
```

```

    id                                INTEGER AUTO_INCREMENT NOT NULL

<Constraints>

    UNIQUE u_a_tbl_id ON a_tbl (id)
    INDEX i1 ON a_tbl (age)
    INDEX i2 ON a_tbl (phone DESC)

```

다음은 ALTER 문으로 인덱스 추가 시 인덱스 커멘트를 포함하는 예제이다.

```
ALTER TABLE tbl ADD index i_t_c (c) COMMENT 'index comment c';
```

ALTER COLUMN ... SET DEFAULT 절

ALTER COLUMN ... SET DEFAULT 절은 기본값이 없는 칼럼에 기본값을 지정하거나 기존의 기본값을 변경할 수 있다. **CHANGE/MODIFY 절** 을 이용하면, 단일 구문으로 여러 칼럼의 기본값을 변경할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name ALTER [COLUMN]
column_name SET DEFAULT value ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 기본값을 변경할 칼럼이 속한 테이블의 이름을 지정한다.
- *column_name*: 새로운 기본값을 적용할 칼럼의 이름을 지정한다.
- *value*: 새로운 기본값을 지정한다.

```

csql> ;schema a_tbl

=== <Help: Schema of a Class> ===

<Class Name>

    a_tbl

<Attributes>

    name                CHARACTER VARYING(1073741823)
    phone               CHARACTER VARYING(13) DEFAULT
'000-0000-0000'
    age                 INTEGER
    id                  INTEGER AUTO_INCREMENT NOT NULL

```


<Constraints>

```
UNIQUE u_a_tbl_id ON a_tbl (id)
```

```
ALTER TABLE a_tbl ALTER COLUMN name SET DEFAULT '';
ALTER TABLE a_tbl ALTER COLUMN phone SET DEFAULT '111-1111';
```

```
csql> ;schema a_tbl
```

```
=== <Help: Schema of a Class> ===
```

<Class Name>

```
a_tbl
```

<Attributes>

name	CHARACTER VARYING(1073741823) DEFAULT ''
phone	CHARACTER VARYING(13) DEFAULT '111-1111'
age	INTEGER
id	INTEGER AUTO_INCREMENT NOT NULL

<Constraints>

```
UNIQUE u_a_tbl_id ON a_tbl (id)
```

```
CREATE TABLE t1(id1 VARCHAR(20), id2 VARCHAR(20) DEFAULT '');
ALTER TABLE t1 ALTER COLUMN id1 SET DEFAULT TO_CHAR(SYS_DATETIME,
'yyyy/mm/dd hh:mi:ss');
```

```
csql> ;schema t1
```

```
=== <Help: Schema of a Class> ===
```

<Class Name>

```
t1
```

<Attributes>

id1	CHARACTER VARYING(20) DEFAULT
TO_CHAR(SYS_DATETIME, 'yyyy/mm/dd hh:mi:ss')	
id2	CHARACTER VARYING(20) DEFAULT ''

AUTO_INCREMENT 절

AUTO_INCREMENT 절은 기존에 정의한 자동 증가값의 초기값을 변경할 수 있다. 단, 테이블 내에 **AUTO_INCREMENT** 칼럼이 한 개만 정의되어 있어야 한다.

```
ALTER TABLE [schema_name.]table_name AUTO_INCREMENT = initial_value ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 테이블 이름
- *initial_value*: 새로 변경할 초기값

```
CREATE TABLE t (i int AUTO_INCREMENT);
ALTER TABLE t AUTO_INCREMENT = 5;

CREATE TABLE t (i int AUTO_INCREMENT, j int AUTO_INCREMENT);

-- when 2 AUTO_INCREMENT constraints are defined on one table, below
query returns an error.
ALTER TABLE t AUTO_INCREMENT = 5;
```

ERROR: To avoid ambiguity, the AUTO_INCREMENT table option requires the table to have exactly one AUTO_INCREMENT column **and** no seed/increment specification.

⚠ 경고

AUTO_INCREMENT의 초기값 변경으로 인해 **PRIMARY KEY**나 **UNIQUE**와 같은 제약 조건에 위배되는 경우가 발생하지 않도록 주의한다.

i 참고

AUTO_INCREMENT 칼럼의 타입을 변경하면 최대값도 변경된다. 예를 들어, INT 타입을 BIGINT 타입으로 변경하면 **AUTO_INCREMENT** 최대값이 INT의 최대값에서 BIGINT의 최대값으로 변경된다.

CHANGE/MODIFY 절

CHANGE 절은 칼럼의 이름, 타입, 크기 및 속성을 변경한다. 기존 칼럼의 이름과 새 칼럼의 이름이 같으면 타입, 크기 및 속성만 변경한다.

MODIFY 절은 칼럼의 타입, 크기 및 속성을 변경할 수 있으며, 칼럼의 이름은 변경할 수 없다.

CHANGE 절이나 **MODIFY** 절로 새 칼럼에 적용할 타입, 크기 및 속성을 설정할 때 기존에 정의된 속성은 새 칼럼의 속성에 전달되지 않는다.

CHANGE 절이나 **MODIFY** 절로 칼럼에 데이터 타입을 변경할 때, 기존의 칼럼 값이 변경되면서 데이터가 변형될 수 있다. 예를 들어 문자열 칼럼의 길이를 줄이면 문자열이 잘릴 수 있으므로 주의해야 한다. 단, **alter_table_change_type_strict** 설정 값이 **yes** 인 경우 에러가 발생한다. 마찬가지로 **allow_truncated_string** 설정 값이 **no** 인 경우에도 에러가 발생한다.

AUTO_INCREMENT 속성의 컬럼 타입을 변경할 경우, AUTO_INCREMENT 속성으로 사용할 수 없는 타입으로 변경할 수 없다. 예를 들면 아래와 같이 ALTER 구문으로 AUTO_INCREMENT 속성 컬럼인 a를 int에서 varchar 타입으로 변경시 에러가 발생한다.

```
CREATE a_tbl (a int AUTO_INCREMENT, b VARCHAR);
ALTER TABLE a_tbl MODIFY COLUMN a VARCHAR;
```

```
ERROR: before ' varchar; '
The domain of the attribute 'a' having an auto increment constraint
is invalid.
```

default값이 지정된 칼럼의 타입을 변경할 때, 지정된 default값이 변경된 타입으로 형변환이 불가능한 경우 아래의 예처럼 에러가 발생한다.

```
CREATE TABLE t_def (a bigint default 123456789012, b varchar(20));
ALTER TABLE t_def MODIFY a int;
```

```
ERROR: A domain conflict exists on attribute "a".
```

경고

- CUBRID 2008 R3.1 이하 버전에서 사용되었던 **ALTER TABLE** *[schema_name.]table_name* **CHANGE** *column_name* **DEFAULT** *default_value* 구문은 더 이상 지원하지 않는다.
- 숫자를 문자 타입으로 변환할 때, **alter_table_change_type_strict=no**이고 해당 문자열의 길이가 숫자의 길이보다 짧으면 변환되는 문자 타입의 길이에 맞추어 문자열이 잘린 상태로 저장된다. **alter_table_change_type_strict=yes**이면 오류를 발생한다.

- 테이블의 칼럼 타입, 컬레이션 등 칼럼 속성을 변경하는 경우 변경된 속성이 변경 전의 테이블을 이용하여 생성한 뷰에 반영되지는 않는다. 따라서 테이블의 칼럼 속성을 변경하는 경우 뷰를 재생성할 것을 권장한다.

```
ALTER [/**+ SKIP_UPDATE_NULL */] TABLE [schema_name.]tbl_name
<table_options> ;

<table_options> ::=
    <table_option>[, <table_option>, ...]

<table_option> ::=
    CHANGE [COLUMN | CLASS ATTRIBUTE] old_col_name
new_col_name <column_definition>
            [FIRST | AFTER col_name]
    | MODIFY [COLUMN | CLASS ATTRIBUTE] col_name
<column_definition>
            [FIRST | AFTER col_name]
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *tbl_name*: 변경할 칼럼이 속한 테이블의 이름을 지정한다.
- *old_col_name*: 기존 칼럼의 이름을 지정한다.
- *new_col_name*: 변경할 칼럼의 이름을 지정한다.
- *<column_definition>*: 변경할 칼럼의 타입, 크기 및 속성, 커멘트를 지정한다.
- *col_name*: 변경할 칼럼이 어느 칼럼 뒤에 위치할지를 지정한다.
- **SKIP_UPDATE_NULL**: 이 힌트가 추가되면 NOT NULL 제약 조건을 추가할 때 기존의 NULL 값을 검사하지 않는다. **SKIP_UPDATE_NULL** 을 참고한다.

```
CREATE TABLE t1 (a INTEGER);

-- changing column a's name into a1
ALTER TABLE t1 CHANGE a a1 INTEGER;

-- changing column a1's constraint
ALTER TABLE t1 CHANGE a1 a1 INTEGER NOT NULL;
---- or
ALTER TABLE t1 MODIFY a1 INTEGER NOT NULL;

-- changing column col1's type - "DEFAULT 1" constraint is removed.
CREATE TABLE t1 (col1 INT DEFAULT 1);
ALTER TABLE t1 MODIFY col1 BIGINT;
```

```
-- changing column col1's type - "DEFAULT 1" constraint is kept.
CREATE TABLE t1 (col1 INT DEFAULT 1, b VARCHAR(10));
ALTER TABLE t1 MODIFY col1 BIGINT DEFAULT 1;

-- changing column b's size
ALTER TABLE t1 MODIFY b VARCHAR(20);

-- changing the name and position of a column
CREATE TABLE t1 (i1 INT, i2 INT);
INSERT INTO t1 VALUES (1,11), (2,22), (3,33);

SELECT * FROM t1 ORDER BY 1;
```

i1	i2
1	11
2	22
3	33

```
ALTER TABLE t1 CHANGE i2 i0 INTEGER FIRST;
SELECT * FROM t1 ORDER BY 1;
```

i0	i1
11	1
22	2
33	3

```
ALTER TABLE t1 MODIFY i1 VARCHAR (200) DEFAULT TO_CHAR (SYS_DATE);
INSERT INTO t1(i0) VALUES (17);
SELECT * FROM t1 ORDER BY 1;
```

i0	i1
11	'1'
17	'05/24/2017'
22	'2'
33	'3'

```
-- adding NOT NULL constraint (strict)
SET SYSTEM PARAMETERS 'alter_table_change_type_strict=yes';

CREATE TABLE t1 (i INT);
```

```
INSERT INTO t1 VALUES (11), (NULL), (22);
```

```
ALTER TABLE t1 CHANGE i i1 INTEGER NOT NULL;
```

ERROR: Cannot add NOT NULL constraint **for** attribute "i1": there are existing NULL values **for** this attribute.

```
-- adding NOT NULL constraint
SET SYSTEM PARAMETERS 'alter_table_change_type_strict=no';

CREATE TABLE t1 (i INT);
INSERT INTO t1 VALUES (11), (NULL), (22);

ALTER TABLE t1 CHANGE i i1 INTEGER NOT NULL;

SELECT * FROM t1;
```

```
      i1
=====
      22
       0
      11
```

```
-- change the column's data type (no errors)

CREATE TABLE t1 (i1 INT);
INSERT INTO t1 VALUES (1), (-2147483648), (2147483647);

ALTER TABLE t1 CHANGE i1 s1 CHAR(11);
SELECT * FROM t1;
```

```
      s1
=====
'2147483647 '
'-2147483648'
'1          '
```

```
-- change the column's data type (errors), strict mode
SET SYSTEM PARAMETERS 'alter_table_change_type_strict=yes';

CREATE TABLE t1 (i1 INT);
INSERT INTO t1 VALUES (1), (-2147483648), (2147483647);
```

```
ALTER TABLE t1 CHANGE i1 s1 CHAR(4);
```

```
ERROR: ALTER TABLE .. CHANGE : changing to new domain : cast failed,
current configuration doesn't allow truncation or overflow.
```

```
-- change the column's data type (errors)
SET SYSTEM PARAMETERS 'alter_table_change_type_strict=no';

CREATE TABLE t1 (i1 INT);
INSERT INTO t1 VALUES (1), (-2147483648), (2147483647);

ALTER TABLE t1 CHANGE i1 s1 CHAR(4);
SELECT * FROM t1;
```

```
-- hard default values have been placed instead of signaling overflow

s1
=====
'1'
'-214'
'2147'
```

참고

NULL을 NOT NULL로 제약 조건을 변경하는 경우 hard default로 값을 업데이트하는 과정으로 인해 많은 시간이 소요되는데, 이를 해소하기 위한 방법으로 이미 존재하는 NULL 값의 UPDATE는 생략하는 **SKIP_UPDATE_NULL** 힌트를 사용할 수 있다. 단, 이 힌트 사용 이후 사용자는 제약 조건과 불일치되는 NULL 값이 존재할 수 있음을 인지해야 한다.

```
ALTER /*+ SKIP_UPDATE_NULL */ TABLE foo MODIFY col INT NOT NULL;
```

칼럼의 타입 변경에 따른 테이블 속성의 변경

- 타입 변경: 시스템 파라미터 **alter_table_change_type_strict**의 값이 **no**이면 다른 타입으로 값 변경을 허용하고, **yes**이면 허용하지 않는다. 기본값은 **yes**이며, **CAST** 연산자로 허용되는 모든 타입으로 변경이 허용된다. 객체 타입의 변경은 객체의 상위 클래스(테이블)에 의해서만 허용된다. 또한 **char**, **varchar**처럼 스트링 타입으로 변경 시 시스템 파라미터 **allow_truncated_string**이 **no**인 경우 오버플로된 스트링은 허용하지 않는다. 기본값은 **no**이다.

· NOT NULL

- 변경할 칼럼에 **NOT NULL** 제약 조건이 지정되지 않더라도 새 테이블에서 해당 제약 조건이 제거되지 않고 그대로 남아있게 된다. 새 테이블에서 해당 제약 조건을 제거하려면 구문에서 **NULL**을 지정하면 된다.
- 변경할 칼럼에 **NOT NULL** 제약 조건이 지정되면 시스템 파라미터 **alter_table_change_type_strict** 의 설정에 따라 결과가 달라진다.
 - **alter_table_change_type_strict** 가 yes이면 해당 칼럼의 값을 검사하여 **NULL** 이 존재하면 오류가 발생하고 변경을 수행하지 않는다.
 - **alter_table_change_type_strict** 가 no이면 존재하는 모든 **NULL** 값을 변경할 타입의 고정 기본값(hard default value)으로 변경한다.
- **DEFAULT**: 변경할 칼럼에 **DEFAULT** 속성이 지정되지 않더라도 새 테이블에서 해당 속성이 제거되지 않고 그대로 남아있게 된다. 새 테이블에서 해당 속성을 제거하려면 구문에서 **DEFAULT NULL**을 지정하면 된다.
- **COMMENT**: 변경할 칼럼에 **COMMENT** 속성이 지정되지 않더라도 새 테이블에서 해당 속성이 제거되지 않고 그대로 남아있게 된다. 새 테이블에서 해당 속성을 제거하려면 구문에서 **COMMENT** "을 지정하면 된다.
- **AUTO_INCREMENT**: 변경할 칼럼에 **AUTO_INCREMENT** 속성이 지정되지 않더라도 새 테이블에서 해당 속성이 제거되지 않고 남아있게 된다.
 - 주의) **AUTO_INCREMENT** 속성은 CREATE나 ALTER로 한 번 설정한 후에는 제거할 수 없다.
- **ON UPDATE**: 변경할 칼럼에 **ON UPDATE** 속성이 지정되지 않더라도 새 테이블에서 해당 속성이 제거되지 않고 남아있게 된다.
 - 주의) **ON UPDATE** 속성은 CREATE나 ALTER로 한 번 설정한 후에는 제거할 수 없다.
- **FOREIGN KEY**: 참조되고 있거나 참조하고 있는 외래키(foreign key) 제약 조건을 지닌 칼럼은 변경할 수 없다.
- 단일 칼럼 **PRIMARY KEY**
 - 변경할 칼럼에 **PRIMARY KEY** 제약 조건이 지정되면, 기존 칼럼에 **PRIMARY KEY** 제약 조건이 존재하고 타입이 업그레이드되는 경우에만 **PRIMARY KEY** 가 재생성된다.
 - 변경할 칼럼에 **PRIMARY KEY** 제약 조건이 지정되었으나 기존 칼럼에는 존재하지 않으면 **PRIMARY KEY** 가 생성된다.
 - 기존 칼럼에는 **PRIMARY KEY** 제약 조건이 존재하나 변경할 칼럼에는 지정되지 않으면 **PRIMARY KEY** 는 유지된다.
- 멀티 칼럼 **PRIMARY KEY**: 변경할 칼럼에 **PRIMARY KEY** 제약 조건이 지정되고 타입이 업그레이드 되면 **PRIMARY KEY** 가 재생성된다.

· 단일 칼럼 **UNIQUE KEY**

- 타입이 업그레이드되면 **UNIQUE KEY** 가 재생성된다.
- 기존 칼럼에 존재하고 변경할 칼럼에 지정되지 않으면 **UNIQUE KEY** 가 유지된다.
- 기존 칼럼에 존재하지 않고 변경할 칼럼에 지정되면 **UNIQUE KEY** 가 생성된다.

· 멀티 칼럼 **UNIQUE KEY**: 해당 칼럼의 타입이 변경되면 인덱스가 재생성된다.

- 유일하지 않은(non-unique) 인덱스가 있는 칼럼: 해당 칼럼의 타입이 변경되면 인덱스가 재생성된다.
- 파티션 기준 칼럼: 테이블이 해당 칼럼에 의해 파티션되어 있으면, 칼럼을 변경할 수 없다. 파티션을 추가할 수 없다.
- 클래스 계층이 있는 테이블의 칼럼: 하위 클래스가 없는 테이블만 변경할 수 있다. 상위 클래스에서 상속받은 하위 클래스는 변경할 수 없다. 상속받은 속성은 변경할 수 없다.
- 트리거와 뷰: 트리거와 뷰는 변경할 칼럼의 정의에 따라 변경되지 않으므로 사용자가 직접 재정의해야 한다.
- 칼럼 순서: 칼럼 순서를 변경할 수 있다.
- 이름 변경: 이름이 충돌하지 않는 한 이름을 변경할 수 있다.

칼럼의 타입 변경에 따른 값의 변경

alter_table_change_type_strict 파라미터는 타입 변경에 따른 값의 변환을 허용하는지 여부를 결정한다. 값이 **no**이면 칼럼의 타입을 변경하거나 **NOT NULL** 제약 조건을 추가할 때 값이 변경될 수 있다. 기본값은 **yes** 이다.

alter_table_change_type_strict 파라미터의 값이 **no**이면 상황에 따라 다음과 같이 동작한다.

- 숫자 또는 문자열을 숫자로 변환 중 오버플로우 발생: 결과 타입의 부호에 따라 음수면 최소값, 양수면 최대값으로 정해지고 오버플로우가 발생한 레코드에 대한 경고 메시지가 로그에 기록된다. 문자열은 **DOUBLE** 타입으로 변환한 후 같은 법칙을 따른다. 다만, **allow_truncated_string** 설정 값이 **no** 인 경우 오버플로우 에러가 반환될 수 있다.
- 문자열을 더 짧은 문자열로 변환: 레코드는 정의한 타입의 고정 기본값(hard default value)으로 업데이트되고 경고 메시지가 로그에 기록된다. 단, **allow_truncated_string** 설정 값이 **no**인 경우 허용되지 않을 수 있다.
- 그 밖의 이유로 인한 변환 실패: 레코드는 정의한 타입의 고정 기본값(hard default value)으로 업데이트되고 경고 메시지가 로그에 기록된다.

alter_table_change_type_strict 파라미터의 값이 **yes** 혹은 **allow_truncated_string** 파라미터 값이 **no**이면 에러 메시지를 출력하고 변경 내용이 롤백될 수 있다.

ALTER CHANGE 문은 레코드를 업데이트하기 전에 해당 타입 변환이 가능한지 검사하지만, 특정 값은 타입 변환에 실패할 수도 있다. 예를 들어, **VARCHAR** 를 **DATE** 로 변환할 때 값의 형식이 올바르지 않으면 변환에 실패할 수 있으며, 이때에는 **DATE** 타입의 고정 기본값(hard default value)이 지정된다.

고정 기본값(hard default value)은 **ALTER TABLE ... ADD COLUMN** 문에 의한 칼럼 추가 혹은 **ALTER TABLE ... CHANGE/MODIFY** 문에 의한 타입 변환으로 인해 값이 추가되거나 변경될 때 사용되는 값이다. **ADD COLUMN** 문에서는 **add_column_update_hard_default** 시스템 파라미터에 따라 동작이 달라진다.

타입별 고정 기본값

타입	고정 기본값 유무	고정 기본값
INTEGER	유	0
FLOAT	유	0
DOUBLE	유	0
SMALLINT	유	0
DATE	유	date'01/01/0001'
TIME	유	time'00:00'
DATETIME	유	datetime'01/01/0001 00:00'
TIMESTAMP	유	timestamp'00:00:01 01/01/1970' (GMT) AM
NUMERIC	유	0
CHAR	유	"

타입	고정 기본값 유무	고정 기본값
VARCHAR	유	"
SET	유	{}
MULTISET	유	{}
SEQUENCE	유	{}
BIGINT	유	0
BIT	무	
VARBIT	무	
OBJECT	무	
BLOB	무	
CLOB	무	

칼럼의 커멘트

칼럼의 커멘트는 ADD/MODIFY/CHANGE 구문 뒤에 위치하는 `<column_definition>` 에서 지정하거나 COMMENT ON COLUMN 구문 뒤에 위치하는 `<column_comment_definition>` 에서 지정한다. `<column_definition>`은 위의 CREATE TABLE 문법을 참고한다.

COMMENT ON COLUMN 구문에서는 하나 이상의 칼럼을 지정하여 칼럼 커멘트를 변경할 수 있다. 다음은 COMMENT ON COLUMN 구문을 이용해서 칼럼의 커멘트를 변경하는 예제이다.

```
ALTER TABLE t1 COMMENT ON COLUMN c1 = 'changed table column c1
comment';
```

```
ALTER TABLE t1 COMMENT ON COLUMN c2 = 'changed table column c2
comment', c3 = 'changed table column c3 comment';
```

다음은 칼럼의 커멘트를 확인하는 예제이다.

```
SHOW CREATE TABLE t1 /* table_name */ ;

SELECT attr_name, class_name, comment
FROM db_attribute
WHERE class_name = 't1' /* lowercase_table_name */ ;

SHOW FULL COLUMNS FROM t1 /* table_name */ ;
```

CSQL 인터프리터에서 “;sc table_name” 명령으로도 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc t1
```

RENAME COLUMN 절

RENAME COLUMN 절을 사용하여 칼럼의 이름을 변경할 수 있다.

```
ALTER [TABLE | CLASS | VCLASS | VIEW] [schema_name.]table_name
RENAME [COLUMN | ATTRIBUTE] old_column_name {AS | TO} new_column_name
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 이름을 변경할 칼럼의 테이블 이름을 지정한다.
- *old_column_name*: 현재의 칼럼 이름을 지정한다.
- *new_column_name*: 새로운 칼럼 이름을 **AS** 키워드 뒤에 명시한다(최대 254 바이트).

```
CREATE TABLE a_tbl (id INT, name VARCHAR(50));
ALTER TABLE a_tbl RENAME COLUMN name AS name1;
```

DROP COLUMN 절

DROP COLUMN 절을 사용하여 테이블에 존재하는 칼럼을 삭제할 수 있다. 삭제하고자 하는 칼럼들을 쉼표(,)로 구분하여 여러 개의 칼럼을 한 번에 삭제할 수 있다.

```
ALTER [TABLE | CLASS | VCLASS | VIEW] [schema_name.]table_name
DROP [COLUMN | ATTRIBUTE] column_name, ... ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 삭제할 칼럼의 테이블 이름을 명시한다.
- *column_name*: 삭제할 칼럼의 이름을 명시한다. 쉼표로 구분하여 여러 개의 칼럼을 지정할 수 있다.

```
ALTER TABLE a_tbl DROP COLUMN age1,age2,age3;
```

DROP CONSTRAINT 절

DROP CONSTRAINT 절을 사용하여, 테이블에 이미 정의된 **UNIQUE, PRIMARY KEY, FOREIGN KEY** 제약 조건을 삭제할 수 있다. 삭제할 제약 조건 이름을 지정해야 하며, 이는 CSQL 명령어(**;schema table_name**)를 사용하여 확인할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
DROP CONSTRAINT constraint_name ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 제약 조건을 삭제할 테이블의 이름을 지정한다.
- *constraint_name*: 삭제할 제약 조건의 이름을 지정한다.

```
CREATE TABLE a_tbl (
  id INT NOT NULL DEFAULT 0 PRIMARY KEY,
  phone VARCHAR(10)
);

CREATE TABLE b_tbl (
  ID INT NOT NULL,
  name VARCHAR (10) NOT NULL,
  CONSTRAINT u_name UNIQUE (name),
  CONSTRAINT pk_id PRIMARY KEY (id),
  CONSTRAINT fk_id FOREIGN KEY (id) REFERENCES a_tbl (id)
  ON DELETE CASCADE ON UPDATE RESTRICT
);

ALTER TABLE b_tbl DROP CONSTRAINT pk_id;
ALTER TABLE b_tbl DROP CONSTRAINT fk_id;
ALTER TABLE b_tbl DROP CONSTRAINT u_name;
```

DROP INDEX 절

DROP INDEX 절을 사용하여 인덱스를 삭제할 수 있다. 고유 인덱스는 **DROP CONSTRAINT** 절로도 삭제할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name DROP INDEX  
index_name ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 제약 조건을 삭제할 테이블의 이름을 지정한다.
- *index_name*: 삭제할 인덱스의 이름을 지정한다.

```
ALTER TABLE a_tbl DROP INDEX i_a_tbl_age;
```

DROP PRIMARY KEY 절

DROP PRIMARY KEY 절을 사용하여 테이블에 정의된 기본키 제약 조건을 삭제할 수 있다. 하나의 테이블에는 하나의 기본키만 정의될 수 있으므로 기본키 제약 조건 이름을 지정하지 않아도 된다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name DROP PRIMARY KEY ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 기본키 제약 조건을 삭제할 테이블의 이름을 지정한다.

```
ALTER TABLE a_tbl DROP PRIMARY KEY;
```

DROP FOREIGN KEY 절

DROP FOREIGN KEY 절을 사용하여 테이블에 정의된 외래키 제약 조건을 모두 삭제할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name DROP FOREIGN KEY  
constraint_name ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 제약 조건을 삭제할 테이블의 이름을 지정한다.
- *constraint_name*: 삭제할 외래키 제약 조건의 이름을 지정한다.

```
ALTER TABLE b_tbl ADD CONSTRAINT fk_id FOREIGN KEY (id) REFERENCES
a_tbl (id);
ALTER TABLE b_tbl DROP FOREIGN KEY fk_id;
```

DROP TABLE

DROP 구문을 이용하여 기존의 테이블을 삭제할 수 있다. 하나의 **DROP** 구문으로 여러 개의 테이블을 삭제할 수 있으며 테이블이 삭제되면 포함된 행도 모두 삭제된다. **IF EXISTS** 절을 함께 사용하면 해당 테이블이 존재하지 않더라도 에러가 발생하지 않는다.

```
DROP [TABLE | CLASS] [IF EXISTS] <table_specification_comma_list>
[CASCADE CONSTRAINTS] ;

<table_specification_comma_list> ::=
    <single_table_spec> | (<table_specification_comma_list>)

<single_table_spec> ::=
    | [ONLY] [schema_name.]table_name
    | ALL [schema_name.]table_name [( EXCEPT
[schema_name.]table_name, ... )]
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 삭제할 테이블의 이름을 지정한다. 쉼표로 구분하여 여러 개의 테이블을 한 번에 삭제할 수 있다.
- **ONLY** 키워드 뒤에 수퍼클래스 이름이 명시되면, 해당 수퍼클래스만 삭제하고 이를 상속받는 서브클래스는 삭제하지 않는다.
- **ALL** 키워드 뒤에 수퍼클래스 이름이 지정되면, 해당 수퍼클래스 및 이를 상속받는 서브클래스를 모두 삭제한다.
- **EXCEPT** 키워드 뒤에 삭제하지 않을 서브클래스 리스트를 명시할 수 있다.
- **CASCADE CONSTRAINTS**: 테이블이 DROP되고 이 테이블을 참조하는 다른 테이블들의 외래 키도 DROP된다.

```
CREATE TABLE b_tbl (i INT);
CREATE TABLE a_tbl (i INT);

-- DROP TABLE IF EXISTS
DROP TABLE IF EXISTS b_tbl, a_tbl;

SELECT * FROM a_tbl;
```

```
ERROR: Unknown class "a_tbl".
```

- **CASCADE CONSTRAINTS** 가 명시되면 다른 테이블들이 DROP할 테이블의 기본 키를 참조하더라도 지정된 테이블은 DROP되며, 이 테이블을 참조하는 다른 테이블들의 외래 키 역시 DROP된다. 단, 참조하는 테이블들의 데이터는 삭제되지 않는다.

다음은 b_child 테이블이 참조하는 a_parent 테이블을 DROP하는 예이다. b_child의 외래 키 역시 DROP되며, b_child의 데이터는 유지된다.

```
CREATE TABLE a_parent (
    id INTEGER PRIMARY KEY,
    name VARCHAR(10)
);
CREATE TABLE b_child (
    id INTEGER PRIMARY KEY,
    parent_id INTEGER,
    CONSTRAINT fk_parent_id FOREIGN KEY(parent_id) REFERENCES
a_parent(id) ON DELETE CASCADE ON UPDATE RESTRICT
);

DROP TABLE a_parent CASCADE CONSTRAINTS;
```

RENAME TABLE

RENAME TABLE 구문을 사용하여 테이블 이름을 변경할 수 있으며, 여러 개의 테이블 이름을 변경하는 경우 테이블 이름 리스트를 명시할 수 있다.

```
RENAME [TABLE | CLASS] [schema_name.]old_table_name {AS | TO}
[schema_name.]new_table_name [{, [schema_name.]old_table_name {AS |
TO} [schema_name.]new_table_name}];
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다. 변경할 테이블의 스키마와 새로운 테이블의 스키마가 동일해야 한다.
- *old_table_name*: 변경할 테이블의 이름을 지정한다.
- *new_table_name*: 새로운 테이블 이름을 지정한다(최대 254 바이트).

```
RENAME TABLE a_tbl AS aa_tbl;
RENAME TABLE aa_tbl TO a1_tbl, b_tbl TO b1_tbl;
```

참고

테이블의 소유자, **DBA**, **DBA**의 멤버만이 테이블의 이름을 변경할 수 있으며, 그 밖의 사용자는 소유자나 **DBA**로부터 이름을 변경할 수 있는 권한을 받아야 한다(권한 관련 사항은 **GRANT** 참조).

인덱스 정의문

CREATE INDEX

CREATE INDEX 구문을 이용하여 지정한 테이블에 인덱스를 생성한다. 인덱스 이름 작성 원칙은 **식별자**를 참고한다.

인덱스 힌트 구문, 내림차순 인덱스, 커버링 인덱스, 인덱스 스kip 스캔, **ORDER BY** 최적화, **GROUP BY** 최적화 등 인덱스를 이용하는 방법과 필터링된 인덱스, 함수 인덱스를 생성하는 방법에 대해서는 **조인 방식**을 참고한다.

```
CREATE [UNIQUE] INDEX index_name ON [schema_name.]table_name
<index_col_desc2> ;

<index_col_desc2> ::=
    {
        ( {column_name | function_name (argument_list)}
[ASC | DESC]
        [{, {column_name | function_name (argument_list)}
[ASC | DESC]} ...] )
    }
    [WHERE <filter_predicate> ]
    [WITH <index_with_clause> [{,
<index_with_clause>}] ...]
    [INVISIBLE]
    [COMMENT 'index_comment_string']

    <index_with_clause> ::= {ONLINE [PARALLEL parallel_count]}
| <index_with_option>
    <index_with_option> ::= {DEDUPLICATE '=' deduplicate_level}
```

- **UNIQUE**: 유일한 값을 갖는 고유 인덱스를 생성한다.
- *index_name*: 생성하려는 인덱스의 이름을 명시한다. 인덱스 이름은 테이블 안에서 고유한 값이어야 한다.
- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 인덱스를 생성할 테이블의 이름을 명시한다.

- *column_name*: 인덱스를 적용할 컬럼의 이름을 명시한다. 다중 컬럼 인덱스를 생성할 경우 둘 이상의 컬럼 이름을 명시한다.
- **ASC | DESC**: 컬럼의 정렬 방향을 설정한다.
- *deduplicate_level*: deduplicate 레벨을 지정한다(0 ~ 14). 자세한 내용은 **DEDUPLICATE**를 참고한다.

참고

*deduplicate_level*은 0부터 14까지의 정수이다. 0은 **DEDUPLICATE** 옵션이 없었던 CUBRID 11.2 또는 이하 버전과 동일한 구성의 인덱스를 의미한다.

- *<filter_predicate>*: 필터링된 인덱스를 만드는 조건을 명시한다. 컬럼과 상수 간 비교 조건이 여러 개인 경우 **AND** 로 연결된 경우에만 필터링이 될 수 있다. 자세한 내용은 **필터링된 인덱스**를 참고한다.
- *function_name (argument_list)*: 함수 기반 인덱스를 만드는 조건을 명시한다. 이와 관련하여 **함수 기반 인덱스**를 반드시 참고한다.
- **WITH ONLINE**: 다른 트랜잭션들에 의해 테이블 데이터가 변경중에 인덱스의 생성을 허용한다. **PARALLEL** 이 선언되지 않은 경우, 인덱스는 동일 트랜잭션 스레드에서 생성된다. *<parallel_count>*는 인덱스를 생성하기 위해 사용되는 스레드의 개수이며 1부터 16사이의 정수이다.
- **INVISIBLE**: 인덱스를 생성할 때 인덱스의 상태를 **INVISIBLE** 로 설정하면, 질의 실행시 해당 인덱스를 사용하지 않는다는 것을 의미한다. **INVISIBLE** 이 생략된 경우 생성되는 인덱스의 상태는 **NORMAL_INDEX** 로 설정된다.
- *index_comment_string*: 인덱스의 커멘트를 지정한다.

참고

- CUBRID 9.0 버전부터는 CREATE/ALTER INDEX 구문에서 인덱스 이름을 생략할 수 없다.
- prefix 인덱스 기능은 제거될 예정(deprecated)이므로, 더 이상 사용을 권장하지 않는다.
- 데이터베이스의 **TIMESTAMP**, **TIMESTAMP WITH LOCAL TIME ZONE** 또는 **DATETIME WITH LOCAL TIME ZONE** 타입 컬럼에 인덱스 또는 함수 인덱스가 포함되어 있는 경우 세션 및 서버 타임존 (**타임존 파라미터**)을 변경하면 안 된다.
- 데이터베이스의 **TIMESTAMP** 또는 **TIMESTAMP WITH LOCAL TIME ZONE** 타입 컬럼에 인덱스 또는 함수 인덱스가 포함되어 있는 경우 윤초를 지원하는 파라미터(**타임존 파라미터**)를 변경하면 안 된다.

- **PARALLEL** 옵션에 설정된 스레드의 개수는 인덱스 로드 과정에서만 적용된다. 수집 과정은 해당 트랜잭션에서 단일 스레드로 실행된다. 이 과정에서 레코드들은 특정 단위의 묶음으로 수집되고, 묶음들의 크기가 16M에 도달한 경우 인덱스 로딩 스레드에 전달하고 (또는 큐에 입력한다), 수집 프로세스는 계속 진행된다. 한 묶음의 크기는 인덱스 키 길이와 열의 식별자 길이 (8 bytes)로 결정한다.
- stand-alone 모드에서 WITH ONLINE 옵션은 무시된다 (정상적인 인덱스가 생성된다).
- 온라인 인덱스 생성은 다음 3 단계로 실행된다:
 - INDEX IS IN ONLINE BUILDING 상태로 스키마에 인덱스를 추가한다. 이 과정은 테이블에 SCH_M_LOCK (다른 모든 스키마 변경 때와 같이)을 확보한 상태에서 실행된다. 이후, 잠금은 IX_LOCK으로 강등된다.
 - 인덱스 채우기: 힙파일을 묶음 단위로 검사, 묶음들을 정렬하고 인덱스에 키를 추가한다. 이 과정 중에 다른 트랜잭션들은 테이블 데이터를 수정할 수있다 (인덱스도 커밋된 다른 트랜잭션에 의해서 수정된 데이터와 함께 변경된다).
 - 인덱스 상태를 NORMAL INDEX로 변경; 이 과정은 테이블 잠금을 다시 SCH_M_LOCK으로 승격한 후에 수행된다.
- 구성중인 온라인 인덱스는 다른 트랜잭션에서 SHOW문으로 표시된다. 다른 트랜잭션에서는 MVCC 스냅샷 때문에 `_db_index` 시스템 테이블에서는 보이지 않는다 (다른 트랜잭션은 이 테이블에서 커밋된 항목만 볼 수있다).
- 온라인 인덱스 구성과 병렬로 실행중인 트랜잭션이 인덱스에 대하여 고유 키 위반을 유발하는 명령을 실행하는 경우, 그 트랜잭션은 커밋이 허용된다. 온라인 인덱스 구성은 계속 진행되고 최종 단계 (스키마에 NORMAL INDEX 상태로 설정)에 이르기 전에 고유키 제약의 유효성을 검사한다. 고유 위반인 경우 인덱스 생성이 중단된다. 사용자는 고유 제약 조건이 보장된 후에 이 작업을 다시 시작해야한다.

다음은 내림차순으로 정렬된 인덱스를 생성하는 예제이다.

```
CREATE INDEX gold_index ON participant(gold DESC);
```

다음은 다중 칼럼 인덱스를 생성하는 예제이다.

```
CREATE INDEX name_nation_idx ON athlete(name, nation_code) COMMENT
'index comment';
```

인덱스의 커멘트

인덱스의 커멘트를 다음과 같이 지정할 수 있다.

```
CREATE TABLE tbl (a int default 0, b int, c int);

CREATE INDEX i_tbl_b on tbl (b) COMMENT 'index comment for i_tbl_b';

CREATE TABLE tbl2 (a INT, index i_tbl_a (a) COMMENT 'index comment',
b INT);

ALTER TABLE tbl2 ADD INDEX i_tbl2_b (b) COMMENT 'index comment b';
```

지정된 인덱스의 커멘트는 다음 구문에서 확인할 수 있다.

```
SHOW CREATE TABLE table_name;
SELECT index_name, class_name, comment from db_index WHERE
class_name ='classname';
SHOW INDEX FROM table_name;
```

또는 CSQL 인터프리터에서 테이블의 스키마를 출력하는 ;sc 명령으로 인덱스의 커멘트를 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc tbl
```

온라인 인덱스 생성

다른 트랜잭션이 테이블에 데이터를 추가하거나 갱신을 허용하면서 인덱스를 생성할 수 있다.

```
CREATE TABLE t1 (i1 int, i2 int);

CREATE INDEX i_t1_i1 on t1 (i1) WITH ONLINE PARALLEL 10;
```

다른 트랜잭션에서 온라인 인덱스 출력

다른 트랜잭션은 스키마 관련 문장으로 온라인 인덱스를 볼 수 있다:

```
csql> show index in t1;

=== <Result of SELECT Command in Line 1> ===

Table                Non_unique  Key_name
```

```

Seq_in_index  Column_name  Collation
Cardinality   Sub_part  Packed
Null          Index_type  Func
Comment       Visible
=====
=====
=====
=====
      't1'                                1
      'i_t1'                            1  'i1'
      'A'                                0      NULL
NULL          'YES'                     'BTREE'
NULL          NULL                      'NO'

1 row selected. (0.020779 sec) Committed.

1 command(s) successfully processed.
csql> desc t1;

=== <Result of SELECT Command in Line 1> ===

      Field          Type          Null
      Key            Default        Extra
=====
=====
      'i1'            'INTEGER'    'YES'
      'MUL'           NULL         ''
      'i2'            'INTEGER'    'YES'
      ''              NULL         ''

csql> ;schema t1

=== <Help: Schema of a Class> ===

<Class Name>

      t1

<Attributes>

      i1              INTEGER
      i2              INTEGER

<Constraints>

      INDEX i_t1 ON t1 (i1) IN PROGRESS

```

다른 트랜잭션이 고유키 위반을 유발하는 삽입을 실행하는 도중의 온라인 인덱스

session 1

session 2

```
CREATE TABLE t1 (i1 int, i2
int);
```

```
COMMIT WORK;
```

```
INSERT INTO t1 VALUES (1, 10);
```

```
CREATE UNIQUE INDEX i_t1_i1 on
t1 (i1) WITH ONLINE;
```

```
csql> ;schema t1

=== <Help: Schema of a Class>
===

<Class Name>

    t1

<Attributes>

    i1
    INTEGER
    i2
    INTEGER

<Constraints>

    UNIQUE i_t1 ON t1 (i1)
    IN PROGRESS
```

session 1

session 2

```
INSERT INTO t1 VALUES (1, 20);

COMMIT WORK;
```

```
COMMIT WORK;
```

```
ERROR: Operation would have
caused one or more unique
constraint
```

```
violations. INDEX
i_t1(B+tree: 0|3456|3457) ON
```

```
CLASS t1(CLASS_OID: 0|202|7).
key: *UNKNOWN-KEY*.
```

DEDUPLICATE

DEDUPLICATE 옵션을 사용하면 특정 키값에 치우쳐진 비대칭 데이터(skewed data)인덱스의 데이터 수정 시 발생할 수 있는 성능 저하를 개선할 수 있다. 이 옵션 값으로 특정 키 값에 대한 오버플로우 페이지의 연결 리스트가 너무 길게 만들어지지 않게 완화하도록 조정할 수 있으며 그에 따라 삽입/삭제/갱신 및 vacuum 성능을 개선한다. 하지만 시스템에 의해 숨겨진 인덱스 칼럼이 추가되었으므로 인덱스의 단말 노드의 개수와 트리의 높이가 증가 할 수 있어 인덱스의 크기가 커지며 조회 성능에 영향을 줄 수 있으므로 주의해야 한다. 특히, 인덱스 데이터가 키 값에 대하여 균일하게 분포하는 경우에는 성능 개선 효과 없이 인덱스의 크기만 커질 수 있으므로 주의해야 한다.

*deduplicate level*의 값이 1 이상 지정되면 인덱스 생성 시 사용자가 명시한 인덱스 칼럼에 더하여 시스템 내부적으로 사용하는 숨겨진 인덱스 칼럼이 하나 추가되는데, 이 값을 사용하여 특정 키값에 치우치지 않도록 중복도를 완화한다. *deduplicate level*값이 커질수록 중복도를 더 완화하게 되며, 특정 키값의 오버플로우 페이지의 연결 리스트 길이가 짧아지게 된다.

참고

- 오버플로우 페이지 연결 리스트는 추가, 삭제, 갱신 성능 면에서는 불리하지만 조회 성능 면에서는 유리하다. 그러므로, *deduplicate level*을 무조건 크게 해서 오버플로우 페이지 자체가 생

성되지 않도록 유도하는 것은 바람직하지 않으며 중복도를 고려해서 적당한 값을 지정해야 한다. (하나의 키에 대한 오버플로우 페이지의 길이는 수십 혹은 수백 페이지 이내가 되도록 조정한다)

인덱스를 생성할 때 *deduplicate level*를 지정해 주는 방법은 두 가지이다.

묵시적인 방법

SQL 구문에 명시적인 DEDUPLICATE 옵션 지정이 없는 경우에 *deduplicate level*를 자동으로 지정해 주는 방식이다. 이 방식은 시스템 파라미터 **deduplicate_key_level** 설정값의 영향을 받는다. **deduplicate_key_level**이 1 이상이면 *deduplicate level*은 자동으로 **deduplicate_key_level**값으로 지정된다.

명시적인 방법

SQL 구문에 사용자가 명시적으로 **DEDUPLICATE** 옵션을 지정하는 방식이다. **deduplicate_key_level** 설정값과 상관없이 사용자가 지정한 *deduplicate level*를 적용한다. 아래 예시와 같이 DEDUPLICATE 구문을 직접 기술해서 명시한다.

```
CREATE TABLE tbl (a int default 0, b int, c int);
CREATE INDEX i_tbl_b on tbl (b) WITH DEDUPLICATE=3 COMMENT
'for deduplicate level 3';
CREATE INDEX i_tbl_b_c on tbl (b,c) WITH DEDUPLICATE=7
COMMENT 'for deduplicate level 1';
```

⚠ 경고

- **deduplicate_key_level**이 -1인 경우는 명시적인 방법으로 지정을 해도 내부적으로 무시되어 적용되지 않는다. 즉 이 경우에는 모든 인덱스는 *deduplicate level*이 0으로 생성된다.
- “_dedup_”로 시작되는 컬럼명을 생성할 수 없다.

📖 참고

- 인덱스 생성시 키 필드의 구성이 **UNIQUE**를 보장 받는다면 사용자가 설정한 **DEDUPLICATE** 설정이 무시되며 *deduplicate level*은 0으로 생성된다.
 - 키 필드가 특정한 Primary Key 또는 Unique Index를 구성하는 키필드를 모두 포함하고 있는 경우

- 단, 인덱스의 함수 인수로 Primary Key 또는 Unique Index의 컬럼이 사용된 경우는 제외

동일한 구성을 갖는 복수의 인덱스 허용

*deduplicate level*을 제외한 모든 키 필드 및 필터 조건이 동일한 복수개의 인덱스를 정의할 수 있다.

```
CREATE TABLE tbl (a int primary key, b int, c int);
CREATE INDEX idx1 ON tbl(b, c) WITH DEDUPLICATE=3;
CREATE INDEX idx2 ON tbl(b, c) WITH DEDUPLICATE=5;
CREATE UNIQUE INDEX idx_uk ON tbl(b);
CREATE INDEX idx3 ON tbl(b, c) WITH DEDUPLICATE=7;
```

위 예시에서 idx1과 idx2는 지정된 *deduplicate level*을 갖는다. 그렇지만 idx3는 b 컬럼이 idx_uk에 의해 Unique 할 것을 보장 받기 때문에 사용자의 지정을 무시하고 *deduplicate level*이 0으로 생성된다.

참고

- *deduplicate level*만 다른 동일한 인덱스 컬럼을 가진 FK가 이미 존재한다면 중복해서 생성할 수 없다.
- 중복된 인덱스가 허용되더라도 동일한 구성의 Primary Key나 Unique Index가 있다면 중복 생성되지 않는다.
- ALTER INDEX REBUILD 구문으로는 인덱스의 *deduplicate level*을 변경할 수 없다. 필요하다면 인덱스 삭제 후 재생성해야 한다.

오버플로우 페이지

• 오버플로우 페이지란?

인덱스는 크게 비단말 노드, 단말 노드로 구성되는데 단말 노드는 인덱스 키 정보의 집합으로 구성된다. 이때 하나의 인덱스 키 정보는 키값과 그 키값에 해당하는 레코드의 OID들의 집합을 쌍으로 묶은 것이다. 특정 키값을 갖는 레코드가 많아지면 그 모든 레코드의 OID들을 단말노드에 담을 수 없기 때문에 별도의 저장 구조로 분리해서 관리하는데 이것을 오버플로우 페이지라고 부른다.

또한 오버플로우 페이지마저도 가득 차게 되면 새로운 오버플로우 페이지를 생성하고 이 페이지들을 연결 리스트로 유지 관리한다.

ALTER INDEX

ALTER INDEX 문은 인덱스의 특성을 변경한다. 주석 또는 상태만 변경된 경우를 제외하고 인덱스가 재구성된다. 인덱스 재구성은 인덱스를 제거하고 다시 생성하는 작업이다.

다음은 인덱스를 재구성하는 구문이다.

```
ALTER INDEX index_name ON [schema_name.]table_name {[COMMENT
index_comment_string] REBUILD} | {COMMENT index_comment_string} |
VISIBLE | INVISIBLE};
```

- *index_name*: 변경 혹은 재구성하려는 인덱스의 이름을 명시한다. 인덱스 이름은 테이블 안에서 고유한 값이어야 한다.
- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 인덱스를 재구성할 테이블의 이름을 명시한다.
- **REBUILD**: 기존과 동일한 구조의 인덱스로 재구성한다.
- *index_comment_string*: 인덱스의 커멘트를 지정한다.

참고

- CUBRID 9.0 버전부터는 인덱스 이름을 생략할 수 없다.
- CUBRID 10.0 버전부터는 테이블 이름을 생략할 수 없다.
- CUBRID 10.0 버전부터는 테이블 이름 뒤에 칼럼 이름을 추가하더라도 이는 무시되며, 예전 인덱스와 동일한 칼럼으로 재생성된다.
- CUBRID 11.4 버전부터는 테이블 이름 뒤에 칼럼 이름을 추가하면 에러가 발생한다.

다음은 인덱스를 재생성하는 구문이다.

```
CREATE INDEX i_game_medal ON game(medal);
ALTER INDEX i_game_medal ON game COMMENT 'rebuild index comment'
REBUILD ;
```

인덱스를 재구성하지 않고 인덱스의 커멘트를 추가하거나 변경하려는 경우 다음과 같이 **COMMENT** 절을 추가하고 **REBUILD** 키워드를 제거한다.

```
ALTER INDEX index_name ON table_name COMMENT 'index_comment_string' ;
```

다음은 인덱스 재구성 없이 커멘트만 추가 또는 변경하는 구문이다.

```
ALTER INDEX i_game_medal ON game COMMENT 'change index comment' ;
```

다음은 인덱스의 상태를 **INVISIBLE/VISIBLE** 로 변경하기 위한 구문이다. 인덱스의 상태가 **INVISIBLE** 인 경우, 질의 실행은 인덱스가 없는 것처럼 수행된다. 이 방법으로 인덱스의 성능 측정이 가능하며, 실제로 인덱스를 제거하지 않고 인덱스 제거에 따른 영향도를 측정할 수 있다.

```
CREATE INDEX i_game_medal ON game(medal);
ALTER INDEX i_game_medal ON game VISIBLE;
ALTER INDEX i_game_medal ON game INVISIBLE;
```

DROP INDEX

DROP INDEX 문을 사용하여 인덱스를 삭제할 수 있다. 고유 인덱스는 **DROP CONSTRAINT** 절로도 삭제할 수 있다.

```
DROP INDEX index_name ON [schema_name.]table_name ;
```

- *index_name*: 삭제할 인덱스의 이름을 지정한다.
- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 삭제할 인덱스가 지정된 테이블 이름을 지정한다.

경고

CUBRID 10.0 버전부터는 테이블 이름을 생략할 수 없다.

다음은 인덱스를 삭제하는 예제이다.

```
DROP INDEX i_game_medal ON game;
```

뷰 정의문

CREATE VIEW

뷰(가상 테이블)는 물리적으로 존재하지 않는 가상의 테이블이며, 기존의 테이블이나 뷰에 대한 질의문을 이용하여 뷰를 생성할 수 있다. **VIEW** 와 **VCLASS** 는 동의어로 사용된다.

CREATE VIEW 문을 이용하여 뷰를 생성한다. 기존 테이블 또는 동의어와 이름이 같은 뷰는 생성할 수 없다. 뷰 이름 작성 원칙은 **식별자**를 참고한다.

```
CREATE [OR REPLACE] {VIEW | VCLASS} [schema_name.]view_name
[<subclass_definition>]
[(view_column_name [COMMENT 'column_comment_string'], ...)]
[INHERIT <resolution>, ...]
[AS <select_statement>]
[WITH CHECK OPTION]
[COMMENT [=] 'view_comment_string'];

<subclass_definition> ::= {UNDER | AS SUBCLASS OF}
[schema_name.]superclass_name, ...
<resolution> ::= [CLASS | TABLE] {column_name} OF
[schema_name.]superclass_name [AS alias]
```

- **OR REPLACE:** **CREATE** 뒤에 **OR REPLACE** 키워드가 명시되면, *view_name*이 기존의 뷰와 이름이 중복되더라도 에러를 출력하지 않고 기존의 뷰를 새로운 뷰로 대체한다.
- *schema_name:* 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *view_name:* 생성하려는 뷰의 이름을 지정한다. 뷰의 이름은 데이터베이스 내에서 고유해야 한다.
- *view_column_name:* 생성하려는 뷰의 칼럼 이름을 지정한다.
- **AS <select_statement>:** 유효한 **SELECT** 문이 명시되어야 한다. 이를 기반으로 뷰가 생성된다.
- **WITH CHECK OPTION:** 이 옵션이 명시되면 <select_statement> 내 **WHERE** 절에 명시된 조건식을 만족하는 경우에만 업데이트 또는 삽입이 가능하다. 조건식을 위반하는 가상 테이블에 대한 갱신을 허용하지 않기 위해서 사용한다.
- *view_comment_string:* 뷰의 커멘트를 지정한다.
- *column_comment_string:* 칼럼의 커멘트를 지정한다.

```
CREATE TABLE a_tbl (
    id INT NOT NULL,
    phone VARCHAR(10)
);
INSERT INTO a_tbl VALUES(1, '111-1111'), (2, '222-2222'), (3,
```

```
'333-3333'), (4, NULL), (5, NULL);

--creating a new view based on AS select_statement from a_tbl
CREATE VIEW b_view AS SELECT * FROM a_tbl WHERE phone IS NOT NULL
WITH CHECK OPTION;
SELECT * FROM b_view;
```

```
      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
```

```
--WITH CHECK OPTION doesn't allow updating column value which
violates WHERE clause
UPDATE b_view SET phone=NULL;
```

```
ERROR: Check option exception on view b_view.
```

다음은 기존 뷰의 정의를 갱신한다. 이와 함께 뷰에 커멘트를 추가하고 있다.

```
--creating view which name is as same as existing view name
CREATE OR REPLACE VIEW b_view AS SELECT * FROM a_tbl ORDER BY id
DESC COMMENT 'changed view';

--the existing view has been replaced as a new view by OR REPLACE
keyword
SELECT * FROM b_view;
```

```
      id  phone
=====
      5  NULL
      4  NULL
      3  '333-3333'
      2  '222-2222'
      1  '111-1111'
```

다음은 뷰의 칼럼에 커멘트를 추가한다.

```
CREATE OR REPLACE VIEW b_view(a COMMENT 'column id', b COMMENT
'column phone') AS SELECT * FROM a_tbl ORDER BY id DESC;
```

업데이트 가능한 VIEW의 생성 조건

다음의 조건을 만족한다면 해당 뷰를 업데이트할 수 있다.

- **FROM** 절은 반드시 업데이트 가능한 테이블이나 뷰만 포함해야 한다.

CUBRID 9.0 미만 버전에서는 **FROM** 절에 업데이트 가능한 테이블을 포함할 경우 반드시 하나의 테이블만 포함해야 했다. 단, FROM (class_x, class_y)와 같이 괄호에 포함된 두 테이블은 하나의 테이블로 표현되므로 업데이트할 수 있었다. CUBRID 9.0 이상 버전에서는 업데이트 가능한 두 개 이상의 테이블을 허용한다.

- **JOIN** 구문을 포함할 수 있다.

참고

CUBRID 10.0 미만 버전에서는 뷰에 **JOIN** 구문을 포함한 뷰를 업데이트 할 수 없다.

- **DISTINCT, UNIQUE** 구문을 포함하지 않는다.
- **GROUP BY ... HAVING** 구문을 포함하지 않는다.
- **SUM (), AVG ()**와 같은 집계 함수를 포함하지 않는다.
- **UNION** 이 아닌 **UNION ALL** 을 사용하여 업데이트 가능한 질의만으로 질의를 구성한 경우 업데이트할 수 있다. 단, 테이블은 **UNION ALL** 을 구성하는 질의 중 어느 한 질의에만 존재해야 한다.
- **UNION ALL** 구문을 사용하여 생성된 뷰에 레코드를 입력하는 경우, 레코드가 입력될 테이블은 시스템이 결정한다. 레코드가 입력될 테이블을 사용자가 제어하는 것은 불가능하므로 사용자가 제어하기 원한다면 테이블에 직접 입력하거나 입력을 위한 별도의 뷰를 생성해야 한다.

뷰가 위의 규칙을 모두 충족해도, 해당 뷰의 다음과 같은 칼럼은 업데이트할 수 없다.

- 경로 표현식(예: `tbl_name.col_name`)
- 산술 연산자가 포함된 숫자 타입의 칼럼

뷰에 정의된 칼럼이 업데이트 가능하더라도 **FROM** 구문에 포함된 테이블에 대해 업데이트를 위한 적절한 권한이 있어야 하며 뷰에 대한 접근 권한이 있어야 한다. 뷰에 접근 권한을 부여하는 방법은 테이블에 접근 권한을 부여하는 방식과 동일하다. 권한 부여에 대한 자세한 내용은 **GRANT** 를 참조한다.

뷰의 커멘트

뷰의 커멘트를 다음과 같이 명시할 수 있다.

```
CREATE OR REPLACE VIEW b_view AS SELECT * FROM a_tbl ORDER BY id
DESC COMMENT 'changed view';
```

명시된 뷰의 커멘트는 다음 구문에서 확인할 수 있다.

```
SHOW CREATE VIEW view_name;
SELECT vclass_name, comment from db_vclass;
```

또는 CSQL 인터프리터에서 스키마를 출력하는 ;sc 명령으로 뷰의 커멘트를 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc b_view
```

뷰의 각 칼럼에도 커멘트 추가가 가능하다.

```
CREATE OR REPLACE VIEW b_view (a COMMENT 'a comment', b COMMENT 'b
comment')
AS SELECT * FROM a_tbl ORDER BY id DESC COMMENT 'view comment';
```

뷰 커멘트의 변경은 아래의 ALTER VIEW 구문을 참고한다.

ALTER VIEW

ADD QUERY 절

ALTER VIEW 문에 **ADD QUERY** 절을 사용하여 뷰의 질의 명세부에 질의를 추가할 수 있다. 뷰 생성 시 정의된 질의문에는 1이 부여되고, **ADD QUERY** 절에서 추가한 질의문에는 2가 부여된다.

```
ALTER [VIEW | VCLASS] [schema_name.]view_name
ADD QUERY <select_statement>
[INHERIT <resolution> , ...] ;

    <resolution> ::= {column_name} OF [schema_name.]superclass_name
[AS alias]
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *view_name*: 질의를 추가할 뷰의 이름 명시한다.
- *<select_statement>*: 추가할 질의를 명시한다.

```
SELECT * FROM b_view;
```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      4  NULL
      5  NULL

```

```
ALTER VIEW b_view ADD QUERY SELECT * FROM a_tbl WHERE id IN (1,2);
SELECT * FROM b_view;
```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      4  NULL
      5  NULL
      1  '111-1111'
      2  '222-2222'

```

AS SELECT 절

ALTER VIEW 문에 **AS SELECT** 절을 사용하여 가상 테이블에 정의된 **SELECT** 질의를 변경할 수 있다. 이는 **CREATE OR REPLACE** 문과 유사하게 동작한다. **ALTER VIEW** 문의 **CHANGE QUERY** 절에 질의 번호 1을 명시하여 질의를 변경할 수도 있다.

```
ALTER [VIEW | VCLASS] [schema_name.]view_name AS <select_statement> ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *view_name*: 변경할 가상 테이블의 이름을 명시한다.
- *<select_statement>*: 가상 테이블 생성 시 정의된 **SELECT** 문을 대체할 새로운 질의문을 명시한다.

```
ALTER VIEW b_view AS SELECT * FROM a_tbl WHERE phone IS NOT NULL;
SELECT * FROM b_view;
```



```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'

```

CHANGE QUERY 절

ALTER VIEW 문의 **CHANGE QUERY** 절을 사용하여 뷰 질의 명세부에 정의된 질의를 변경할 수 있다.

```

ALTER [VIEW | VCLASS] [schema_name.]view_name
CHANGE QUERY [integer] <select_statement> ;

```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *view_name*: 변경할 뷰의 이름을 명시한다.
- *integer*: 변경할 질의의 번호를 명시한다. 기본값은 1이다.
- *<select_statement>*: 질의 번호가 *integer* 인 질의를 대치할 새로운 질의를 명시한다.

```

--adding select_statement which query number is 2 and 3 for each
ALTER VIEW b_view ADD QUERY SELECT * FROM a_tbl WHERE id IN (1,2);
ALTER VIEW b_view ADD QUERY SELECT * FROM a_tbl WHERE id = 3;
SELECT * FROM b_view;

```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      4  NULL
      5  NULL
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'

```

```

--altering view changing query number 2
ALTER VIEW b_view CHANGE QUERY 2 SELECT * FROM a_tbl WHERE phone IS
NULL;
SELECT * FROM b_view;

```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      4  NULL
      5  NULL
      4  NULL
      5  NULL
      3  '333-3333'

```

DROP QUERY 절

ALTER VIEW 문의 **DROP QUERY** 예약어를 이용하여 뷰 질의 명세부에 정의된 질의를 삭제할 수 있다.

```

ALTER VIEW b_view DROP QUERY 2,3;
SELECT * FROM b_view;

```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      4  NULL
      5  NULL

```

COMMENT 절

ALTER VIEW 문의 **COMMENT** 절을 이용하여 뷰와 칼럼들, 어트리뷰트들의 커멘트를 변경할 수 있다.

```

ALTER [VIEW | VCLASS] [schema_name.]view_name
COMMENT [=] 'view_comment_string' |
COMMENT ON {COLUMN | CLASS ATTRIBUTE} <column_comment_definition> [,
<column_comment_definition>] ;

<column_comment_definition> ::= column_name [=]
'column_comment_string'

```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *view_name*: 변경할 뷰의 이름을 명시한다.
- *column_name*: 변경할 칼럼의 이름을 명시한다.

- `view_comment_string`: 뷰의 커멘트를 지정한다.
- `column_comment_string`: 칼럼의 커멘트를 지정한다.

다음은 뷰의 커멘트를 변경하는 예제이다.

```
ALTER VIEW v1 COMMENT = 'changed view v1 comment';
```

ON COLUMN 키워드 뒤에 하나 이상의 칼럼을 지정하여 칼럼의 커멘트를 변경할 수 있다. 다음은 칼럼의 커멘트를 변경하는 예제이다.

```
ALTER VIEW v1 COMMENT ON COLUMN c1 = 'changed view column c1 comment';
ALTER VIEW v1 COMMENT ON COLUMN c2 = 'changed view column c2 comment', c3 = 'changed view column c3 comment';
```

다음은 뷰와 칼럼의 커멘트를 확인하는 예제이다. 하지만 SHOW CREATE VIEW 구문에서는 뷰 커멘트만 확인할 수 있다.

```
SHOW CREATE VIEW v1 /* view_name */ ;

SELECT attr_name, class_name, comment
FROM db_attribute
WHERE class_name = 'v1' /* lowercase_view_name */ ;

SHOW FULL COLUMNS FROM v1 /* view_name */ ;
```

CSQL 인터프리터에서 “;sc view_name” 명령으로도 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc v1
```

DROP VIEW

뷰는 **DROP VIEW** 문을 이용하여 삭제할 수 있다. 뷰를 삭제하는 방법은 일반 테이블을 삭제하는 방법과 동일하다. IF EXISTS 절을 함께 사용하면 해당 뷰가 존재하지 않더라도 에러가 발생하지 않는다.

```
DROP [VIEW | VCLASS] [IF EXISTS] [schema_name.]view_name [{,
[schema_name.]view_name}] ;
```

- `schema_name`: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.

- *view_name* : 삭제하려는 뷰의 이름을 지정한다.

```
DROP VIEW b_view;
```

RENAME VIEW

뷰의 이름은 **RENAME VIEW** 문을 사용하여 변경할 수 있다.

```
RENAME [VIEW | VCLASS] [schema_name.]old_view_name {AS | TO}
[schema_name.]new_view_name [{, [schema_name.]old_view_name {AS |
TO} [schema_name.]new_view_name}] ;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다. 변경할 뷰의 스키마와 새로운 뷰의 스키마는 동일해야 한다.
- *old_view_name* : 변경할 뷰의 이름을 지정한다.
- *new_view_name* : 뷰의 새로운 이름을 지정한다.

다음은 *game_2004* 뷰의 이름을 *info_2004* 로 변경하는 예제이다.

```
RENAME VIEW game_2004 AS info_2004;
```

시리얼 정의문

CREATE SERIAL

시리얼(**SERIAL**)은 고유한 순번을 생성하는 객체이다. 시리얼은 다음과 같은 특성을 갖는다.

- 시리얼은 다중 사용자 환경에서 고유한 순번을 생성하는데 용이하다.
- 시리얼 번호는 테이블과 독립적으로 생성된다. 따라서 하나 이상의 테이블에 동일한 시리얼을 사용할 수 있다.
- **PUBLIC** 을 포함한 모든 사용자가 시리얼 객체를 생성할 수 있다. 일단 생성되면 모든 사용자들이 **CURRENT_VALUE (CURRVAL과 동일)**, **NEXT_VALUE (NEXTVAL과 동일)** 를 통해 시리얼 값을 조회할 수 있다.
- 시리얼 객체의 소유자와 **DBA** 만 시리얼 객체를 갱신하고 삭제할 수 있다. 소유자가 **PUBLIC** 이면 모든 사용자가 갱신하거나 삭제할 수 있다.

CREATE SERIAL 문을 이용하여 데이터베이스에 시리얼 객체를 생성한다. 시리얼 이름 작성 원칙은 **식별자**를 참고한다.

```
CREATE SERIAL [schema_name.]serial_name
[START WITH initial]
[INCREMENT BY interval]
[MINVALUE min | NOMINVALUE]
[MAXVALUE max | NOMAXVALUE]
[CYCLE | NOCYCLE]
[CACHE cached_num | NOCACHE]
[COMMENT 'comment_string'];
```

- *schema_name*: 시리얼의 스키마 이름을 지정한다(최대 31 바이트). 생략하면 현재 세션의 스키마 이름을 사용한다.
- *serial_name*: 생성할 시리얼의 이름을 지정한다(최대 222 바이트).
- **START WITH** *initial*: 처음 생성되는 시리얼 숫자를 지정한다. 이 값의 범위는 $-1,000,000,000,000,000,000,000,000,000,000,000(-10^{36})$ 와 $9,999,999,999,999,999,999,999,999,999,999(10^{37}-1)$ 사이이다. 오름차순 시리얼의 경우 기본값은 1이며 내림차순 시리얼의 경우 기본값은 -1이다.
- **INCREMENT BY** *interval*: 시리얼 숫자 간의 간격을 지정한다. *interval* 값으로 0을 제외하고 $-9,999,999,999,999,999,999,999,999,999,999(-10^{37}+1)$ 와 $9,999,999,999,999,999,999,999,999,999,999(10^{37}-1)$ 사이의 어느 정수라도 올 수 있다. *interval*의 절대값은 **MAXVALUE**와 **MINVALUE**의 차이와 같거나 작아야 한다. 음수가 설정되면 시리얼은 내림차순이 되고 양수가 설정되면 오름차순이 된다. 기본값은 1이다.
- **MINVALUE**: 시리얼의 최소값을 지정한다. 이 값의 범위는 $-1,000,000,000,000,000,000,000,000,000,000,000(-10^{36})$ 와 $9,999,999,999,999,999,999,999,999,999,999(10^{37}-1)$ 사이이다. **MINVALUE**는 초기값보다 작거나 같아야 하고 최대값보다 작아야 한다.
- **NOMINVALUE**: 오름차순 시리얼에 대해서는 1, 내림차순 시리얼에 대해서는 $-1,000,000,000,000,000,000,000,000,000,000,000(-10^{36})$ 이 최소값으로 자동 지정된다.
- **MAXVALUE**: 시리얼의 최대값을 지정한다. 이 값의 범위는 $-999,999,999,999,999,999,999,999,999,999(-10^{36}+1)$ 와 $10,000,000,000,000,000,000,000,000,000,000,000(10^{37})$ 사이이다. **MAXVALUE**는 초기값보다 크거나 같아야 하고 최소값보다 커야 한다.
- **NOMAXVALUE**: 오름차순 시리얼에 대해서는 $10,000,000,000,000,000,000,000,000,000,000,000(10^{37})$, 내림차순 시리얼에 대해서는 -1이 최대값으로 자동 지정된다.

- **CYCLE**: 시리얼 값이 최대 또는 최소값에 도달한 후에 연속적으로 값을 생성하도록 지정한다. 오름차순 시리얼은 최대값에 도달한 후에 다음 값으로 최소값이 생성된다. 내림차순 시리얼은 최소값에 도달한 후에 다음 값으로 최대값이 생성된다.
- **NOCYCLE**: 시리얼이 최대 또는 최소값에 도달한 후에 시리얼 값이 더 이상 생성되지 않도록 지정한다. 기본값은 **NOCYCLE** 이다.
- **CACHE**: 시리얼 성능을 향상시키기 위하여 *cached_num* 에 지정된 개수만큼의 시리얼을 캐시에 저장하고, 시리얼 값을 요청받으면 캐시된 시리얼 값을 가져온다. 또한, 메모리에 캐시된 시리얼을 전부 사용하게 되면, *cached_num* 개수 만큼의 시리얼을 디스크로부터 메모리로 다시 가져오며, 데이터베이스 서버가 중간에 종료되면 캐시된 시리얼 값들은 삭제된다. 따라서 데이터베이스 서버가 재시작되기 이전과 이후의 시리얼 값은 비연속적일 수 있다. 캐시된 시리얼은 트랜잭션 롤백되지 않으므로, 롤백을 수행하더라도 다음에 요청하는 시리얼은 이전에 최종 요청한 시리얼 값의 다음 값이 된다. **CACHE** 키워드 뒤에 *cached_num* 는 생략할 수 없으며, 1 이하의 숫자가 지정되면 시리얼 캐시가 적용되지 않는다.
- **NOCACHE**: 시리얼 캐시 기능을 사용하지 않으며, 매번 시리얼 값을 업데이트한다. 기본값은 **NOCACHE** 이다.
- *comment_string*: 시리얼의 커멘트를 지정한다.

```
--creating serial with default values
CREATE SERIAL order_no;

--creating serial within a specific range
CREATE SERIAL order_no START WITH 10000 INCREMENT BY 2 MAXVALUE
20000;

--creating serial with specifying the number of cached serial values
CREATE SERIAL order_no START WITH 10000 INCREMENT BY 2 MAXVALUE
20000 CACHE 3;

--selecting serial information from the db_serial
SELECT name, current_val, increment_val, max_val, min_val, cyclic,
started, cached_num, attr_name FROM db_serial;
```

name	current_val	increment_val		
max_val	min_val	cyclic	started	
cached_num	attr_name			
=====				
=====				
=====				
'order_no'	10006	2		
20000	10000	0		1
3	NULL			

다음은 선수 번호와 이름을 저장하는 `athlete_idx` 테이블을 생성하고 `order_no` 시리얼을 이용하여 인스턴스를 생성하는 예제이다. `order_no.CURRENT_VALUE`는 시리얼의 현재 값을 반환하고, `order_no.NEXT_VALUE`는 시리얼 값을 증가시킨 후 값을 반환한다.

```
CREATE TABLE athlete_idx( code INT, name VARCHAR(40) );
CREATE SERIAL order_no START WITH 10000 INCREMENT BY 2 MAXVALUE
20000;
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Park');
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Kim');
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Choo');
INSERT INTO athlete_idx VALUES (order_no.CURRENT_VALUE, 'Lee');

SELECT * FROM athlete_idx;
```

```
      code  name
=====
    10000  'Park'
    10002  'Kim'
    10004  'Choo'
    10004  'Lee'
```

시리얼의 커멘트

다음은 시리얼 생성 시 커멘트를 추가한다.

```
CREATE SERIAL order_no
START WITH 100 INCREMENT BY 2 MAXVALUE 200
COMMENT 'from 100 to 200 by 2';
```

시리얼의 커멘트를 확인하려면 다음의 구문을 실행한다.

```
SELECT name, comment FROM db_serial;
```

시리얼 커멘트의 변경은 ALTER SERIAL 문의 설명을 참고한다.

ALTER SERIAL

ALTER SERIAL 문을 이용하면 시리얼 값의 증가치를 갱신하고 시작 값, 최소 값, 최대 값을 설정하거나 제거할 수 있으며, 순환 속성을 설정할 수 있다.

```
ALTER SERIAL [schema_name.]serial_name
[INCREMENT BY interval]
[START WITH initial_value]
```

```
[MINVALUE min | NOMINVALUE]
[MAXVALUE max | NOMAXVALUE]
[CYCLE | NOCYCLE]
[CACHE cached_num | NOCACHE]
[COMMENT 'comment_string'];
```

- *schema_name*: 시리얼의 스키마 이름을 지정한다(최대 31 바이트). 생략하면 현재 세션의 스키마 이름을 사용한다.
- *serial_name*: 생성할 시리얼의 이름을 지정한다(최대 222 바이트).
- **INCREMENT BY interval**: 시리얼 숫자간의 간격을 지정한다. *interval* 값으로 0을 제외한 38자리 이하의 어떤 정수도 지정할 수 있다. *interval*의 절대값은 **MAXVALUE**와 **MINVALUE**의 차이보다 작아야 한다. 음수가 설정되면 시리얼은 내림차순이 되고 양수가 설정되면 오름차순이 된다. 기본값은 1이다.
- **START WITH initial_value**: 시리얼의 시작 값을 변경한다.
- **MINVALUE**: 시리얼의 최소값을 지정한다. 이 값은 38자리 이하의 숫자이다. **MINVALUE**는 초기값보다 작거나 같아야 하고 최대값보다 작아야 한다.
- **NOMINVALUE**: 오름차순 시리얼에 대해서는 1, 내림차순 시리얼에 대해서는 $-(10)^{38}$ 이 최소값으로 자동 지정된다.
- **MAXVALUE**: 시리얼의 최대값을 지정한다. 이 값은 38자리 이하의 숫자이다. **MAXVALUE**는 초기값보다 크거나 같아야 하고 최소값보다 커야 한다.
- **NOMAXVALUE**: 오름차순 시리얼에 대해서는 $(10)^{37}$, 내림차순 시리얼에 대해서는 -1이 최대값으로 자동 지정된다.
- **CYCLE**: 시리얼 값이 최대 또는 최소값에 도달한 후에 연속적으로 값을 생성하도록 지정한다. 오름차순 시리얼은 최대값에 도달한 후에 다음 값으로 최소값이 생성된다. 내림차순 시리얼은 최소값에 도달한 후에 다음 값으로 최대값이 생성된다.
- **NOCYCLE**: 시리얼이 최대 또는 최소값에 도달한 후에 시리얼 값이 더 이상 생성되지 않도록 지정한다. 기본값은 **NOCYCLE**이다.
- **CACHE**: 시리얼 성능을 향상시키기 위하여 *cached_num*에 지정된 개수만큼의 시리얼을 캐시에 저장하고, 시리얼 값을 요청받으면 캐시된 시리얼 값을 가져온다. **CACHE** 키워드 뒤에 *cached_num*는 생략할 수 없으며, 1 이하의 숫자가 지정되면 시리얼 캐시가 적용되지 않는다.
- **NOCACHE**: 시리얼 캐시 기능을 사용하지 않으며, 매번 시리얼 값이 업데이트된다. 기본값은 **NOCACHE**이다.
- *comment_string*: 시리얼의 커멘트를 지정한다.

```
--altering serial by changing start and incremental values
ALTER SERIAL order_no START WITH 100 MINVALUE 100 INCREMENT BY 2;
```



```
--altering serial to operate in cache mode
ALTER SERIAL order_no CACHE 5;

--altering serial to operate in common mode
ALTER SERIAL order_no NOCACHE;
```

⚠ 경고

CUBRID 2008 R1.x 버전에서는 시스템 카탈로그인 **db_serial** 테이블을 업데이트하는 방식으로 시리얼 값을 변경할 수 있었으나, CUBRID 2008 R2.0 이상 버전부터는 **db_serial** 테이블의 수정은 허용되지 않고 **ALTER SERIAL** 구문을 이용하는 방식만 허용된다. 따라서 CUBRID 2008 R2.0 이상 버전에서 내보내기(unloaddb)한 데이터에 **ALTER SERIAL** 구문이 포함된 경우에는 이를 CUBRID 2008 R1.x 이하 버전에서 가져오기(loaddb)할 수 없다.

⚠ 경고

ALTER SERIAL 이후 첫번째 **NEXT_VALUE** 값을 구하면 CUBRID 9.0 미만 버전에서는 **ALTER SERIAL** 로 설정한 초기값의 다음 값을 반환했으나, CUBRID 9.0 이상 버전에서는 **ALTER_SERIAL** 의 설정 값을 반환한다.

```
CREATE SERIAL s1;
SELECT s1.NEXTVAL;

ALTER SERIAL s1 START WITH 10;

SELECT s1.NEXTVAL;
-- From 9.0, above query returns 10
-- In the version less than 9.0, above query returns 11
```

다음은 시리얼의 커멘트를 변경한다.

```
ALTER SERIAL order_no COMMENT 'new comment';
```

DROP SERIAL

DROP SERIAL 문으로 시리얼 객체를 데이터베이스에서 삭제할 수 있다. **IF EXISTS** 절을 함께 지정하는 경우, 대상 시리얼이 없어도 에러가 발생하지 않는다.

```
DROP SERIAL [ IF EXISTS ] [schema_name.]serial_name ;
```

- *schema_name*: 시리얼의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *serial_name*: 삭제할 시리얼의 이름을 지정한다.

다음 예는 *order_no* 시리얼을 삭제하는 예제이다.

```
DROP SERIAL order_no;
DROP SERIAL IF EXISTS order_no;
```

시리얼 사용

의사 칼럼

시리얼 이름과 의사 칼럼(pseudo column)을 통해서 해당 시리얼을 읽고 갱신할 수 있다.

```
[schema_name.]serial_name.CURRENT_VALUE
[schema_name.]serial_name.CURRVAL

[schema_name.]serial_name.NEXT_VALUE
[schema_name.]serial_name.NEXTVAL
```

- *schema_name*: 시리얼의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *[schema_name.]serial_name.CURRENT_VALUE*, *[schema_name.]serial_name.CURRVAL*: 시리얼의 현재 값을 반환한다.
- *[schema_name.]serial_name.NEXT_VALUE*, *[schema_name.]serial_name.NEXTVAL*: 시리얼 값을 증가시키고 그 값을 반환한다.

다음은 선수 번호와 이름을 저장하는 *athlete_idx* 테이블을 생성하고 *order_no* 시리얼을 이용하여 인스턴스를 생성하는 예제이다.

```
CREATE TABLE athlete_idx (code INT, name VARCHAR (40));
CREATE SERIAL order_no START WITH 10000 INCREMENT BY 2 MAXVALUE
20000;
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Park');
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Kim');
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Choo');
INSERT INTO athlete_idx VALUES (order_no.NEXT_VALUE, 'Lee');
SELECT * FROM athlete_idx;
SELECT order_no.CURRENT_VALUE;
```

```

      code  name
=====
      10000  'Park'
      10002  'Kim'
      10004  'Choo'
      10006  'Lee'

      serial_current_value(order_no)
=====
      10006

```

참고

시리얼을 생성하고 처음 사용할 때 **NEXT_VALUE** 를 이용하면 초기 값을 반환한다. 그 이후에는 현재 값에 증가 값이 추가되어 반환된다.

함수

SERIAL_CURRENT_VALUE(*[schema_name.]serial_name*)

SERIAL_NEXT_VALUE(*[schema_name.]serial_name, number*)

시리얼 함수에는 **SERIAL_CURRENT_VALUE** 함수와 **SERIAL_NEXT_VALUE** 함수가 있다.

매개변수:

- **schema_name** – 시리얼의 스키마 이름
- **serial_name** – 시리얼 이름
- **number** – 얻고자 하는 시리얼 개수

반환 형식:

NUMERIC(38,0)

SERIAL_CURRENT_VALUE 함수는 현재의 시리얼 값을 반환하며, *serial_name.current_value* 와 동일한 값을 반환한다.

SERIAL_NEXT_VALUE 함수는 현재의 시리얼 값에서 지정한 개수의 시리얼 간격만큼 증가시킨 값을 반환한다. 시리얼 간격은 **CREATE SERIAL ... INCREMENT BY** 절로 지정한 값을 따른다. **SERIAL_NEXT_VALUE** (*serial_name, 1*)은 *serial_name.next_value* 와 동일한 값을 반환한다.

한꺼번에 많은 개수의 시리얼을 얻고자 할 때에는, `serial_name.next_value` 를 반복하여 호출하는 것보다 원하는 개수를 인자로 하여 **SERIAL_NEXT_VALUE** 함수를 한 번만 호출하는 것이 성능상 유리하다.

즉, 어떤 응용 프로세스가 한꺼번에 N 개의 시리얼을 얻고자 한다면 N 번 `serial_name.next_value` 를 호출하여 값들을 구하는 것보다는, 한 번 **SERIAL_NEXT_VALUE** (`serial_name`, N)을 호출하여 반환하는 값을 가지고 (함수를 호출한 시점의 시리얼 시작값)과 (반환 값) 사이의 시리얼 값들을 계산하는 것이 낫다. (함수를 호출한 시점의 시리얼 시작값)은 (반환 값) - (얻고자 하는 시리얼 개수-1) * (시리얼 간격)이다.

예를 들어, 101로 시작하며 1씩 증가하는 시리얼을 처음에 생성하였을 경우, 처음 **SERIAL_NEXT_VALUE** (`serial_name`, 10)을 호출하면 110이 반환된다. 이 시점의 시작값을 구하면 $110 - (10-1)*1 = 101$ 이므로 101, 102, 103, ... 110까지 10개의 시리얼 값을 해당 응용 프로세스에서 사용할 수 있다. 한 번 더 **SERIAL_NEXT_VALUE** (`serial_name`, 10)을 호출하면 120이 반환되며, 이 시점의 시작값은 $120 - (10-1)*1 = 111$ 이다.

```
CREATE SERIAL order_no START WITH 101 INCREMENT BY 1 MAXVALUE 20000;
SELECT SERIAL_CURRENT_VALUE(order_no);
```

101

```
-- At first, the first serial value starts with the initial serial
value, 10000. So the 10'th serial value will be 10009.
SELECT SERIAL_NEXT_VALUE(order_no, 10);
```

110

```
SELECT SERIAL_NEXT_VALUE(order_no, 10);
```

120

참고

시리얼을 생성하고 **SERIAL_NEXT_VALUE** 함수를 처음 호출하면, 첫 번째 값은 초기값을 반환하므로 한 개의 값이 빠져 현재의 시리얼 값에 (시리얼 간격) * (얻고자 하는 시리얼 개수-1)만큼 증가한 값이 반환된다. 이후 **SERIAL_NEXT_VALUE** 함수를 호출하면 현재 값에 (시리얼 간격) * (얻고자 하는 시리얼 개수)만큼 증가한 값이 반환된다. 위의 예제를 참고한다.

동의어 정의문

CREATE SYNONYM

이미 데이터베이스에 존재하는 객체를 사용할 수 있는 또 다른 새로운 이름을 만든다. 사용자는 데이터베이스 객체를 생성할 때 지정한 이름으로 객체를 사용할 수 있고, 동의어로도 객체를 사용할 수 있다.

1. 사용자는 전용(Private) 동의어만 생성할 수 있으며, 전용(Private) 동의어는 사용자의 스키마에서만 사용할 수 있다.
2. 생성된 동의어의 정보는 **DB_SYNONYM** 시스템 가상 클래스에서 조회할 수 있다.
3. 같은 이름의 테이블이나 뷰가 이미 존재하는 경우 해당 이름으로 동의어를 생성할 수 없다.
4. 동의어를 생성할 때 대상 객체의 스키마가 존재하는지는 확인하지만 대상 객체가 존재하는지는 확인하지 않는다. 대상 객체가 존재하지 않으면 동의어를 사용할 때 에러가 발생한다.
5. 사용자는 해당 사용자의 스키마에서만 동의어를 생성할 수 있지만 DBA와 DBA의 멤버는 스키마를 지정해서 동의어를 생성할 수 있다.

```
CREATE [OR REPLACE] [PRIVATE] SYNONYM
[schema_name_of_synonym.]synonym_name
FOR [schema_name_of_target.]target_name
[COMMENT 'synonym_comment_string'] ;
```

- **OR REPLACE:** 스키마에 *synonym_name*이 이미 존재하더라도 에러를 출력하지 않고 기존의 동의어가 새로운 동의어로 대체된다.
 - 새로운 동의어를 생성하기 전에 기존 동의어는 삭제된다.
- **PRIVATE:** 전용(Private) 동의어를 생성하도록 지정한다. 생략해도 기본값으로 전용(Private) 동의어를 생성한다.
- *schema_name_of_synonym:* 동의어의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *synonym_name:* 동의어의 이름을 지정한다.
- *schema_name_of_target:* 대상 객체의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *target_name:* 대상 객체의 이름을 지정한다.

• `synonym_comment_string`: 동의어의 커멘트를 지정한다.

⚠ 경고

동의어의 대상 객체는 테이블과 뷰만 지원한다. 다른 객체를 대상 객체로 사용할 수는 있지만 동의어를 사용할 때 에러가 발생한다.

⚠ 경고

아직은 공용(Public) 동의어를 지원하지 않는다.

1. 전용(Private) 동의어

아래 예제에서 u1 사용자와 u2 사용자는 같은 이름으로 동의어를 생성했지만 다른 대상 객체를 사용한다.

```
call login ('dba') on class db_user;

/* current_user: dba */
create user u1;
create user u2;

create table t1 (c1 varchar(100));
insert into t1 values ('private synonym for user u1. ');
grant select on t1 to u1;
grant select on t1 to u2;

create table t2 (c1 varchar(100));
insert into t2 values ('private synonym for user u2. ');
grant select on t2 to u1;
grant select on t2 to u2;
```

```
call login ('u1') on class db_user;

/* current_user: u1 */
create synonym s1 for dba.t1;
select * from s1;
```

```

c1
=====
'private synonym for user u1.'

```

```

call login ('u2') on class db_user;

/* current_user: u2 */
create synonym s1 for dba.t2;
select * from s1;

```

```

c1
=====
'private synonym for user u2.'

```

2. 동의어 정보

아래 예제에서 사용자는 **DB_SYNONYM** 시스템 가상 클래스에서 동의어의 정보를 확인할 수 있다.

```

/* There should be the result of example 1. */

call login ('dba') on class db_user;

/* current_user: dba */
select * from db_synonym;

```

synonym_name target_name	synonym_owner_name target_owner_name	is_public_synonym comment
's1'	'U1'	'NO'
't1'	'DBA'	NULL
's1'	'U2'	'NO'
't2'	'DBA'	NULL

```

call login ('u1') on class db_user;

/* current_user: u1 */
select * from db_synonym;

```

synonym_name target_name	synonym_owner_name target_owner_name	is_public_synonym comment

```
=====
's1'                'U1'                'NO'
't1'                'DBA'                NULL
```

```
call login ('u2') on class db_user;

/* current_user: u2 */
select * from db_synonym;
```

```
synonym_name      synonym_owner_name  is_public_synonym
target_name       target_owner_name  comment
=====
's1'              'U2'              'NO'
't2'              'DBA'              NULL
```

```
/* clean */

call login ('dba') on class db_user;

/* current_user: dba */
drop synonym if exists u1.s1;
drop synonym if exists u2.s1;
drop user u1;
drop user u2;

drop table if exists t1;
drop table if exists t2;
```

3. 동의어 이름

같은 이름의 테이블이나 뷰가 이미 존재하는 경우 해당 이름으로 동의어를 생성할 수 없다.

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('table for user public.');
```

```
create table s1 (c1 varchar(100));
create view s2 as select * from t1;
```

```
/* Already used as a table name. */
create synonym s1 for t1;
```



```
ERROR: before ' ; '
Class public.s1 already exists.
```

```
/* Already used as a view name. */
create synonym s2 for t1;
```

```
ERROR: before ' ; '
Class public.s2 already exists.
```

```
create synonym s3 for t1;
select * from s3;
```

```
c1
=====
'table for user public.'
```

```
/* clean */

call login ('public') on class db_user;

/* current_user: public */
drop synonym if exists s1;
drop synonym if exists s2;
drop synonym if exists s3;
drop view if exists s2;
drop table if exists t1;
drop table if exists s1;
```

4. 동의어에 대한 동의어

동의어를 생성할 때 대상 객체가 존재하는지는 확인하지 않으므로, 사용자가 기존 동의어를 대상 객체로 지정하여 새로운 동의어를 만들 수 있다. 하지만 동의어를 사용할 때 대상 객체인 동의어의 대상 객체를 다시 찾지 않는다.

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('synonym for synonym.');
```

```
create synonym s1 for t1;
```

```
create synonym s2 for s1;
select * from db_synonym;
```

synonym_name target_name	synonym_owner_name target_owner_name	is_public_synonym comment
's1'	'PUBLIC'	'NO'
't1'	'PUBLIC'	NULL
's2'	'PUBLIC'	'NO'
's1'	'PUBLIC'	NULL

```
select * from s2;
```

```
ERROR: before ' ; '
Unknown class "public.s1".
```

```
select * from s1;
```

```
c1
=====
'synonym for synonym.'
```

```
/* clean */

call login ('public') on class db_user;

/* current_user: public */
drop synonym if exists s1;
drop synonym if exists s2;
drop table if exists t1;
```

5. 스키마 지정 동의어 생성

DBA와 **DBA**의 멤버가 스키마를 지정하여 동의어를 생성하면, 동의어는 지정한 스키마에 만들어진 다.

```
call login ('dba') on class db_user;

/* current_user: dba */
create user u1;
```

```

create user u2;

create table t1 (c1 varchar(100));
insert into t1 values ('private synonym for user u1. ');
grant select on t1 to u1;
grant select on t1 to u2;

create table t2 (c1 varchar(100));
insert into t2 values ('private synonym for user u2. ');
grant select on t2 to u1;
grant select on t2 to u2;

create synonym u1.s1 for dba.t1;
create synonym u2.s1 for dba.t2;

select * from db_synonym;

```

synonym_name target_name	synonym_owner_name target_owner_name	is_public_synonym comment
's1'	'U1'	'NO'
't1'	'DBA'	NULL
's1'	'U2'	'NO'
't2'	'DBA'	NULL

```

call login ('u1') on class db_user;

/* current_user: u1 */
select * from s1;

```

```

c1
=====
'private synonym for user u1.'

```

```

call login ('u2') on class db_user;

/* current_user: u2 */
select * from s1;

```

```

c1
=====
'private synonym for user u2.'

```

사용자는 해당 사용자의 스키마에서만 동의어를 생성할 수 있다.

```
call login ('u1') on class db_user;

/* current_user: u1 */
create synonym u2.s2 for dba.t1;
```

```
ERROR: before ' ; '
DBA, members of DBA group, and owner can perform CREATE SYNONYM.
```

```
/* clean */

call login ('dba') on class db_user;

/* current_user: dba */
drop synonym if exists u1.s1;
drop synonym if exists u2.s1;
drop user u1;
drop user u2;

drop table if exists t1;
drop table if exists t2;
```

ALTER SYNONYM

동의어의 대상 객체나 커멘트를 변경한다. 사용 중인 동의어는 변경할 수 없다.

```
ALTER [PRIVATE] SYNONYM [schema_name_of_synonym.]synonym_name
{
    FOR [<schema_name_of_target>.<target_name>] [COMMENT
'comment_string'] |
    COMMENT 'synonym_comment_string'
} ;
```

- **PRIVATE:** 전용(Private) 동의어를 변경하도록 지정한다. 생략해도 기본값으로 전용(Private) 동의어를 생성한다.
- *schema_name_of_synonym:* 동의어의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *synonym_name:* 동의어의 이름을 지정한다.
- *schema_name_of_target:* 대상 객체의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.

- *target_name*: 대상 객체의 이름을 지정한다.
- *synonym_comment_string*: 동의어의 커멘트를 지정한다.

⚠ 경고

동의어에 대한 **ALTER**, **DROP**, **RENAME** 문이 실행되면 쿼리 계획 캐시에서 대상 객체를 사용하는 쿼리 계획을 모두 삭제하므로 주의해야 한다.

하지만 **ALTER** 문을 실행할 때 같은 대상 객체로 변경하거나 커멘트만 변경하는 경우에는 쿼리 계획을 삭제하지 않는다.

대상 객체 변경

아래 예제에서 대상 객체를 변경한다.

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('target table before change. ');

create table t2 (c1 varchar(100));
insert into t2 values ('target table after change. ');

create synonym s1 for t1;
select * from db_synonym;
select * from s1;
```

synonym_name	synonym_owner_name	is_public_synonym
target_name	target_owner_name	comment
's1'	'PUBLIC'	'NO'
't1'	'PUBLIC'	NULL

c1
'target table before change.'

```
alter synonym s1 for t2;
select * from db_synonym;
select * from s1;
```

```

synonym_name      synonym_owner_name  is_public_synonym
target_name       target_owner_name    comment
=====
's1'              'PUBLIC'             'NO'
't2'              'PUBLIC'             NULL

c1
=====
'target table after change.'

```

```

/* clean */

call login ('public') on class db_user;

/* current_user: public */
drop synonym if exists s1;
drop table if exists t1;
drop table if exists t2;

```

커멘트 변경

아래 예제에서 사용자는 동의어의 커멘트를 변경한다.

```

call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('change comment.');
```

```

create synonym s1 for t1 comment 'It is a synonym for the t1 table.';
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;

```

```

synonym_name      synonym_owner_name  is_public_synonym
comment
=====
's1'              'PUBLIC'             'NO'
'It is a synonym for the t1 table.'

```

대상 객체를 지정하지 않고 커멘트를 변경할 수 있다.

```
alter synonym s1 comment 'the comment was changed.';
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
```

```
synonym_name      synonym_owner_name  is_public_synonym
comment
=====
=====
's1'              'PUBLIC'          'NO'
'the comment was changed.'
```

대상 객체와 커멘트를 모두 지정하지 않으면 에러가 발생한다.

```
alter synonym s1;
```

```
ERROR: No options specified for ALTER SYNONYM.
```

커멘트를 **NULL**로 변경하려면, 커멘트를 빈 문자열로 변경하면 된다.

```
alter synonym s1 comment '';
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
```

```
synonym_name      synonym_owner_name  is_public_synonym
comment
=====
=====
's1'              'PUBLIC'          'NO'
NULL
```

```
/* clean */

call login ('public') on class db_user;

/* current_user: public */
drop synonym if exists s1;
drop table if exists t1;
```

DROP SYNONYM

동의를 삭제한다. 사용 중인 동의어는 삭제할 수 없다. 동의를 삭제해도 대상 객체는 삭제되지 않는다.

```
DROP [PRIVATE] SYNONYM [IF EXISTS] [schema_name.]synonym_name ;
```

- **PRIVATE:** 전용(Private) 동의를 삭제하도록 지정한다. 생략해도 기본값으로 전용(Private) 동의를 생성한다.
- **IF EXISTS:** 스키마에 *synonym_name*이 존재하지 않더라도 에러가 발생하지 않는다.
- *schema_name:* 동의어의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *synonym_name:* 동의어의 이름을 지정한다.

⚠ 경고

동의를에 대한 **ALTER, DROP, RENAME** 문이 실행되면 쿼리 계획 캐시에서 대상 객체를 사용하는 쿼리 계획을 모두 삭제하므로 주의해야 한다.

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('The target object of the to-be-deleted
synonym.');
```

```
create synonym s1 for t1;
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
select * from s1;
```

```
synonym_name      synonym_owner_name  is_public_synonym
comment
=====
=====
's1'              'PUBLIC'            'NO'
NULL
c1
=====
'The target object of the to-be-deleted synonym.'
```



```
drop synonym s1;
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
```

```
There are no results.
0 row selected.
```

```
select * from s1;
```

```
ERROR: before ' ; '
Unknown class "public.s1".
```

```
select * from t1;
```

```
c1
=====
'The target object of the to-be-deleted synonym.'
```

```
/* clean */

call login ('public') on class db_user;

/* current_user: public */
drop synonym if exists s1;
drop table if exists t1;
```

RENAME SYNONYM

동의어의 이름을 변경한다. 사용 중인 동의어의 이름은 변경할 수 없다.

1. 사용자는 동의어의 이름을 변경하면서 동의어의 스키마를 변경할 수 없다.
2. 변경하는 이름의 테이블이나 뷰, 동의어가 이미 존재하는 경우 이름을 변경할 수 없다.

```
RENAME [PRIVATE] SYNONYM
[schema_name_of_old_synonym.]old_synonym_name
{AS | TO} [schema_name_of_new_synonym.]new_synonym_name ;
```

- **PRIVATE**: 전용(Private) 동의어를 변경하도록 지정한다. 생략해도 기본값으로 전용(Private) 동의어를 생성한다.
- *schema_name_of_old_synonym*: 이름을 바꿀 동의어의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *old_synonym_name*: 이름을 바꿀 동의어의 이름을 지정한다.
- *schema_name_of_new_synonym*: 새로운 이름의 동의어에 대한 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *new_synonym_name*: 동의어의 새로운 이름을 지정한다.

⚠ 경고

동의어에 대한 **ALTER**, **DROP**, **RENAME** 문이 실행되면 쿼리 계획 캐시에서 대상 객체를 사용하는 쿼리 계획을 모두 삭제하므로 주의해야 한다.

1. 스키마를 변경할 수 없음

아래 예제에서 rename 시 스키마 이름을 다르게 지정할 때 에러가 발생한다.

```
call login ('dba') on class db_user;
```

```
/* current_user: dba */
```

```
create user u1;
```

```
create user u2;
```

```
call login ('u1') on class db_user;
```

```
/* current_user: u1 */
```

```
create table t1 (c1 varchar(100));
```

```
insert into t1 values ('private synonym for user u1.');
```

```
create synonym s1 for t1;
```

```
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
```

```
select * from s1;
```

```

    synonym_name      synonym_owner_name  is_public_synonym
comment
=====
=====
    's1'              'U1'                'NO'
NULL

    c1
=====
    'private synonym for user u1.'
```

```

rename synonym s1 as u2.s2;
rename synonym u1.s1 as u2.s2;
```

```

ERROR: before ' ; '
Rename cannot change owner.
```

```

call login ('dba') on class db_user;

/* current_user: dba */
rename synonym u1.s1 as u2.s2;
```

```

ERROR: before ' ; '
Rename cannot change owner.
```

```

call login ('u1') on class db_user;

/* current_user: u1 */
rename synonym s1 as s2;
select synonym_name, synonym_owner_name, is_public_synonym, comment
from db_synonym;
select * from s2;
```

```

    synonym_name      synonym_owner_name  is_public_synonym
comment
=====
=====
    's2'              'U1'                'NO'
NULL

    c1
```

```
=====
'private synonym for user u1.'
```

```
/* clean */

call login ('dba') on class db_user;

/* current_user: dba */
drop synonym if exists u1.s1;
drop synonym if exists u1.s2;
drop synonym if exists u2.s2;
drop table if exists u1.t1;
drop user u1;
drop user u2;
```

2. 이미 사용 중인 이름

아래 예제에서 변경할 이름이 이미 사용중이기 때문에 에러가 발생한다.

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('first table for user u1.');
```

```
create table t2 (c1 varchar(100));
insert into t2 values ('second table for user u1.');
```

```
create table s_t1 (c1 varchar(100));
create table s_v1 as select * from t1;
create synonym s_s1 for t2;
```

```
create synonym s1 for t1;
select * from db_synonym;
select * from s1;
```

synonym_name	synonym_owner_name	is_public_synonym
target_name	target_owner_name	comment
's_s1'	'PUBLIC'	'NO'
't2'	'PUBLIC'	NULL
's1'	'PUBLIC'	'NO'
't1'	'PUBLIC'	NULL

c1

```
=====
'first table for user u1.'
```

```
rename synonym s1 as s_t1;
```

```
ERROR: before ' ; '
Class dba.s_t1 already exists.
```

```
rename synonym s1 as s_v1;
```

```
ERROR: before ' ; '
Class dba.s_v1 already exists.
```

```
rename synonym s1 as s_s1;
```

```
ERROR: before ' ; '
Synonym "dba.s_s1" already exists.
```

```
rename synonym s1 as s2;
select * from db_synonym;
select * from s2;
```

```

synonym_name      synonym_owner_name  is_public_synonym
target_name      target_owner_name  comment
=====
=====
's_s1'           'PUBLIC'           'NO'
't2'             'PUBLIC'           NULL
's2'             'PUBLIC'           'NO'
't1'             'PUBLIC'           NULL

c1
=====
'first table for user u1.'
```

```
/* clean */

call login ('public') on class db_user;
```

```
/* current_user: public */
drop synonym if exists s1;
drop synonym if exists s2;
drop synonym if exists s_s1;
drop table if exists t1;
drop table if exists t2;
drop table if exists s_t1;
drop table if exists s_v1;
```

동의어 사용

동의어는 테이블 이름과 뷰 이름을 사용할 수 있는 경우에만 사용할 수 있다.

1. 동의어를 대상으로 **GRANT** 및 **REVOKE**를 할 수 없다. 동의어의 스키마 이름이 지정되면 다른 스키마에 존재하는 동의어도 사용할 수 있다.
2. 대상 객체를 변경하는 **ALTER**, **DROP**, **RENAME** 문과 **TRUNCATE** 문에서는 동의어를 사용할 수 없다.

1. 다른 스키마의 동의어 사용

```
call login ('dba') on class db_user;
```

```
/* current_user: dba */
create user u1;
create user u2;
```

```
call login ('u1') on class db_user;

/* current_user: u1 */
create table t1 (c1 varchar(100));
insert into t1 values ('first table for user u1. ');
grant select on t1 to u2;

create synonym s1 for t1;
select * from s1;
```

```
c1
=====
'first table for user u1.'
```

```
call login ('u2') on class db_user;
```

```
/* current_user: u2 */
select * from s1;
select * from u1.s1;
```

```
ERROR: before ' ; '
Unknown class "u2.s1".
```

```
c1
=====
'first table for user u1.'
```

```
/* clean */

call login ('dba') on class db_user;

/* current_user: dba */
drop synonym if exists u1.s1;
drop table if exists u1.t1;
drop user u1;
drop user u2;
```

2. 동의어를 사용할 수 없는 구문

```
call login ('public') on class db_user;

/* current_user: public */
create table t1 (c1 varchar(100));
insert into t1 values ('first table for user public.');
```

```
create synonym s1 for t1;
select * from db_synonym;
select * from s1;
```

synonym_name	synonym_owner_name	is_public_synonym
target_name	target_owner_name	comment
's1'	'PUBLIC'	'NO'
't1'	'PUBLIC'	NULL

```
c1
=====
'first table for user public.'
```

```
alter table s1 add column c2 int;
```

```
ERROR: before '  add column c2 int; '  
Class public.s1 does not exist.
```

```
drop table s1;
```

```
ERROR: before ' ; '  
Class public.s1 does not exist.
```

```
rename table s1 to s2;
```

```
ERROR: before ' ; '  
Class public.s1 does not exist.
```

```
truncate s1;
```

```
ERROR: before ' ; '  
Class public.s1 does not exist.
```

```
select * from s1;
```

```
c1  
=====  
'first table for user public.'
```

```
/* clean */  
  
call login ('public') on class db_user;  
  
/* current_user: public */  
drop synonym if exists s1;  
drop table if exists t1;  
drop table if exists s2;
```


서버 정의문

CREATE SERVER

서버 정의

CREATE SERVER 문을 사용하여 원격 접속 정보를 생성한다. 생성된 서버는 **DBLINK**를 이용하는 **SELECT** 질의를 수행 할 때 원격 서버를 지정하는데 사용한다. 준비된 서버의 사용 방법은 **원격 테이블**을 참고한다.

```
CREATE SERVER <dblink_server_name> (<connect_info>) ;

<dblink_server_name> ::= [owner_name.] server_name

<connect_info> ::=
    <connect_info>, <connect_item>
    | <connect_item>
<connect_item> ::=
    HOST = host_string
    | PORT = port_number
    | DBNAME = db_name
    | USER = user_name
    | PASSWORD = [password_string]
    | PROPERTIES = [properties_string]
    | COMMENT = [server_comment_string]
```

- *owner_name*: 생성할 서버의 소유자 이름을 지정한다.
- *server_name*: 생성할 서버의 이름을 지정한다(최대 254바이트).
- *<connect_info>*: 접속 정보 리스트로 *<connect_item>* 항목 중 **HOST, PORT, DBNAME, USER**는 필수 항목이다.
- *<connect_item>*: HOST, PORT, DBNAME, USER, PASSWOED, PROPERTIES, COMMENT 항목으로 구성되며, 동일한 항목이 중복될 수 없다.
 - *host_string*: 원격 접속할 DBMS 정보를 가지고 있는 Broker 서버의 호스명 또는 IP 주소이다.
 - *port_number*: 원격 접속할 DBMS 정보를 가지고 있는 Broker 서버의 포트 번호이다.
 - *db_name*: 원격 접속할 데이터베이스 이름.
 - *user_name*: 원격 접속할 데이터베이스에 접속할 때 사용할 사용자 이름.
 - *password_string*: 원격 접속할 데이터베이스에 접속할 때 사용하는 *user_name*에 대한 패스워드 문자열.

- *properties_string*: 원격 데이터베이스 사용을 위해 broker(또는 gateway)에 접속시 사용하는 property 정보 문자열 (최대 2047 바이트). 상세한 정보는 *cci_connect_with_url*를 참고한다.
- *server_comment_string*: 서버 정보에 대한 커멘트를 지정한다.(최대 1023바이트)

참고

db_name과 user_name은 식별자 형식과 문자열 리터럴 형식으로 모두 기술할 수 있다.

· 식별자 형식의 예

t123db, "123db", 123db, [124db]

· 문자열 형식의 예

't123db', '123db'

예제 1 다음은 필수항목(HOST,PORT,DBNAME,USER) 및 PROPERTIES, COMMENT를 사용한 예제이다.

```
CREATE SERVER dblink_srv1 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dba,
  PASSWORD='password1234',
  PROPERTIES='?rcTime=600',
  COMMENT='this is dblink_srv1'
);
```

예제 2 다음은 서버 생성 시 최소한의 정보만 포함하는 예제이다. 원격지의 demodb에 비밀번호가 없는 dev1 계정으로 접속할 것임을 나타내고 있다. srv1, srv2, srv3는 동일한 의미이다.

```
CREATE SERVER srv1 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dev1
);

CREATE SERVER srv2 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dev1,
  PASSWORD=
);
```

```
CREATE SERVER srv3 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dev1,
  PASSWORD=''
);
```

예제 3 다음은 서버 생성 시 소유자를 지정하는 예제이다. 소유자 지정 없이 CREATE하는 경우에는 현재 사용자가 소유자가 된다. 추후 ALTER SERVER 구문을 이용해서 소유자를 변경할 수 있다. 아래 예제에서 두 서버의 이름이 동일하게 *srv2*로 같지만 소유자는 각각 *dba*와 *cub*로 다르다.

```
-- When the current account is dba
CREATE SERVER srv2 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dev1,
  PASSWORD='dev1-password',
  COMMENT='The owner of this server is dba'
);

CREATE SERVER cub.srv2 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dev2,
  PASSWORD='dev2-password',
  COMMENT='The owner of this server is cub.'
);
```

예제 4 다음은 HA 환경으로 구성된 원격 데이터베이스 사용시 PROPERTIES에 altHosts 속성을 사용하여 원격 데이터베이스의 fail-over에 대비한 설정 예제이다. 자세한 내용은 [cci_connect_with_url](#)을 참고한다.

```
CREATE SERVER dblink_srv1 (
  HOST='192.168.1.8',
  PORT=3300,
  DBNAME=demodb,
  USER=dba,
  PASSWORD='password1234',
  PROPERTIES='?altHosts=192.168.1.9:33000',
  COMMENT='this is dblink_srv1'
);
```

예제 5 다음은 서버명(<dblink_server_name>)를 잘 못 사용한 예제이다.

```
CREATE SERVER srv1 ( HOST='localhost', PORT=3300, DBNAME=demodb,
USER=dev1 );
CREATE SERVER "srv 1" ( HOST='localhost', PORT=3300, DBNAME=demodb,
USER=dev1 );
CREATE SERVER "srv.1" ( HOST='localhost', PORT=3300, DBNAME=demodb,
USER=dev1 );
CREATE SERVER cub.srv1 ( HOST='localhost', PORT=3300, DBNAME=demodb,
USER=dev1 );

CREATE SERVER "cub"."srv 2" ( HOST='localhost', PORT=3300,
DBNAME=demodb, USER=dev1 );
CREATE SERVER [cub].[srv.2] ( HOST='localhost', PORT=3300,
DBNAME=demodb, USER=dev1 );
```

위의 예시에서 “srv.1” 과 [cub].[srv.2] 이름으로 생성은 실패한다. cub.srv1 의 경우 cub 는 사용자 이름으로 인식되고 서버명은 srv1 으로 인식 될 것이다.

참고

서버명에는 점(".")을 사용 할 수 없다. 따옴표나 []로 묶은 경우라고 해도 서버명에 점(".")을 사용할 수는 없다.

RENAME SERVER

RENAME SERVER 구문을 사용하여 서버 이름을 변경할 수 있다.

```
RENAME SERVER [owner_name.] old_server_name {AS | TO}
new_server_name ;
```

- *owner_name*: 이름을 변경할 대상 서버의 소유자 이름을 지정한다.
- *old_server_name*: 이름을 변경할 대상 서버의 이름을 지정한다.
- *new_server_name*: 새롭게 부여될 서버의 이름을 지정한다(최대 254바이트).

참고

서버의 소유자 또는 소유 그룹의 멤버만 정보를 변경 할 수 있다. 특별히 **DBA** 또는 **DBA** 의 멤버는 모든 서버 정보를 변경할 수 있다.

변경 후에도 소유자는 동일하게 유지된다. 소유자를 변경하기 위해서는 **ALTER SERVER** 구문을 참고한다.

```
-- When the current account is dba
RENAME SERVER srv1 AS srv2;
RENAME SERVER dev1.srv1 AS srv3;
```

위 예제를 *dba* 계정에서 수행하더라도, *srv2*의 소유자는 변경되지 않고 *srv1* 서버의 소유자인 *cub*로 유지한다. 또한, *srv3* 서버의 소유자는 *dev1*으로 계속 유지 된다.

DROP SERVER

DROP SERVER 구문을 이용하여 기존의 서버를 제거할 수 있다. **IF EXISTS** 절을 함께 사용하면 해당 서버가 존재하지 않더라도 에러가 발생하지 않는다.

```
DROP SERVER [IF EXISTS] [owner_name.] server_name ;
```

- *owner_name*: 제거할 서버의 소유자 이름을 지정한다.
- *server_name*: 제거할 서버의 이름을 지정한다.

```
DROP SERVER srv1;
DROP SERVER cub.srv1;
DROP SERVER IF EXISTS srv2;
```

⚠ 경고

DROP USER 구문으로 사용자를 삭제 할 때 해당 사용자가 소유하고 있는 서버가 있다면 에러 처리되고 사용자는 삭제되지 않는다. 먼저 사용자가 소유하고 있는 서버를 **DROP SERVER** 구문으로 제거한 후에 사용자 계정을 삭제해야 한다.

```
-- When the current account is dba
csql> create user cub;
Execute OK. (0.000371 sec) Committed.

1 command(s) successfully processed.
csql> create server cub.tsrv (HOST='localhost', PORT=3300,
DBNAME=demdb, USER=dev1);
Execute OK. (0.000761 sec) Committed.
```

```
1 command(s) successfully processed.
csql> drop user cub;
```

In the command **from** line 1,

```
ERROR: Cannot drop the user who owns database objects(class/trigger/
serial/server etc).
```

```
0 command(s) successfully processed.
csql> drop server cub.tsrv;
Execute OK. (0.000761 sec) Committed.
```

```
1 command(s) successfully processed.
csql> drop user cub;
Execute OK. (0.001650 sec) Committed.
```

```
1 command(s) successfully processed.
csql>
```

위 예시를 보면 *cub* 계정 소유의 *tsrv* 서버를 생성한 상태에서 `drop user cub` 구문이 실패하고 있음을 볼 수 있다. 이후 *cub.tsrv* 서버를 제거한 후에는 정상적으로 *cub* 계정을 삭제할 수 있었음을 볼 수 있다.

ALTER SERVER

ALTER 구문을 이용하여 서버의 정보를 변경할 수 있다. 대상 서버의 소유자를 변경하거나, HOST, PORT, DBNAME, USER, PASSWOED, PROPERTIES, COMMENT에 대한 정보를 갱신 할 수 있다.

```
ALTER SERVER <dblink_server_name> <alter_server_list> ;

<dblink_server_name> ::= [owner_name.] server_name

<alter_server_list> ::=
    <alter_server_list>, <alter_server_item>
    | <alter_server_item>
<alter_server_item> ::=
    OWNER TO owner_name
    | CHANGE <connect_item>
<connect_item> ::=
    HOST = host_string
    | PORT = port_number
    | DBNAME = db_name
    | USER = user_name
    | PASSWORD = [password_string]
    | PROPERTIES = [properties_string]
    | COMMENT = [server_comment_string]
```

참고

서버의 소유자 또는 소유 그룹의 멤버만 정보를 변경 할 수 있다. 특별히 **DBA** 또는 **DBA** 의 멤버는 모든 서버 정보를 변경할 수 있다.

경고

HOST, PORT, DBNAME, USER에 대한 값을 제거하는 갱신은 할 수 없다.

OWNER TO 절

OWNER TO 절을 사용하여 서버의 소유자를 변경할 수 있다.

```
ALTER SERVER [owner_name.] server_name OWNER TO new_owner_name ;
```

- *owner_name*: 소유자를 변경할 대상 서버의 소유자 이름을 지정한다.
- *server_name*: 소유자를 변경할 대상 서버의 이름을 지정한다.
- *new_owner_name*: 새로운 소유자 이름을 지정한다.

경고

- 하나의 ALTER SERVER 구문에 OWNER TO 절은 오직 한번만 지정되어야 한다.

```
CREATE SERVER srv1 (HOST='broker-server-name', PORT=3300,
DBNAME=demodb, USER=dev1);
ALTER SERVER srv1 OWNER TO usr1;
ALTER SERVER usr1.srv1 OWNER TO usr2;
```

CHANGE 절

CHANGE 절은 HOST, PORT, DBNAME, USER, PASSWOED, PROPERTIES, COMMENT 항목의 값을 변경 하는데 사용한다.

⚠ 경고

- 하나의 ALTER SERVER 구문에 CHANGE 절은 콤마(,)로 구분하여 여러개가 나열 될 수 있다. 다만 이때 동일한 항목에 대한 CHANGE절은 오직 하나만 지정되어야 한다.
- ALTER SERVER 구문 수행시 언급이 없었던 항목은 그 값이 초기화 되거나 삭제되는 것이 아니고 기존의 값을 그대로 유지하게 된다.

```
ALTER SERVER [owner_name.] server_name CHANGE <connect_item> [,
CHANGE <connect_item>] ... ;
```

```
<connect_item> ::=
    HOST = host_string
    | PORT = port_number
    | DBNAME = db_name
    | USER = user_name
    | PASSWORD = [password_string]
    | PROPERTIES = [properties_string]
    | COMMENT = [server_comment_string]
```

- *owner_name*: 생성할 서버의 소유자 이름을(user name)지정한다.
- *server_name*: 생성할 서버의 이름을 지정한다(최대 254바이트).
- *host_string*: 원격 접속할 DBMS 정보를 가지고 있는 Broker 서버의 호스트명 또는 IP 주소이다.
- *port_number*: 원격 접속할 DBMS 정보를 가지고 있는 Broker 서버의 포트 번호이다.
- *db_name*: 원격 접속할 데이터베이스 이름.
- *user_name*: 원격 접속할 데이터베이스에 접속할 때 사용할 사용자 이름.
- *password_string*: 원격 접속할 데이터베이스에 접속할 때 사용하는 *user_name* 에 대한 패스워드 문자열.
- *properties_string*: 원격 접속할 데이터베이스에 접속할 때 사용하는 property 정보 문자열.
- *server_comment_string*: 서버 정보에 대한 커멘트를 지정한다.

```
CREATE SERVER srv1 ( HOST='localhost', PORT=3300, DBNAME=demodb,
USER=dev1 );
```

```
ALTER SERVER srv1 CHANGE HOST='127.0.0.1';
ALTER SERVER srv1 CHANGE HOST='127.0.0.1', OWNER TO usr1;
```



```
ALTER SERVER srv1 CHANGE USER=dev2, CHANGE PASSWORD='dev2-pawword',
CHANGE PORT=3500;
```

위 예시는 **CHANGE** 절은 여러개를 한번에 나열해서 사용할 수도 있고 **OWNER TO** 절과 함께 사용 될 수도 있음을 보여 준다.

```
ALTER SERVER srv1 CHANGE PORT=;
ALTER SERVER srv1 CHANGE DBNAME=;
ALTER SERVER srv1 CHANGE USER=;
ALTER SERVER srv1 CHANGE HOST=;
ALTER SERVER srv1 CHANGE HOST='';
```

위 예시는 모두 지원 되지 않는 경우에 대한 예시이다. 서버 정보의 구성에서 필수요소인 HOST, PORT, DBNAME, USER는 반드시 값을 가지고 있어야 하기 때문에 값을 삭제하는 설정 변경은 지원되지 않는다. 특히, HOST의 경우 빈 문자열로 설정하는 것도 허용 되지 않는다.

```
ALTER SERVER srv1 CHANGE PASSWORD=;
ALTER SERVER srv1 CHANGE PASSWORD='';

ALTER SERVER srv1 CHANGE PROPERTIES=;
ALTER SERVER srv1 CHANGE PROPERTIES='';

ALTER SERVER srv1 CHANGE COMMENT=;
ALTER SERVER srv1 CHANGE COMMENT='';
```

위 예시는 모두 지원되는 예시이다. 서버 정보의 구성에서 필수요소가 아닌 PASSWORD, PROPERTIES, COMMENT는 반드시 값을 가지고 있어야 할 필요가 없기 때문에 값을 삭제하는 설정 변경이 가능하다.

저장 프로시저/함수 정의문

CREATE PROCEDURE

CREATE PROCEDURE 문을 사용하여 저장 프로시저를 등록한다.

```
CREATE [OR REPLACE] PROCEDURE [schema_name.]procedure_name
([<parameter_definition> [, <parameter_definition>] ...])
[<procedure_properties>]
{IS | AS} LANGUAGE <lang>
    <body>
COMMENT 'procedure_comment';

<parameter_definition> ::= parameter_name [mode] sql_type [ {
```

```

DEFAULT | = } <default_expr> ] [COMMENT 'parameter_comment_string']
<lang> ::= [PLCSQL | JAVA]
<mode> ::= IN | OUT | IN OUT | INOUT
<procedure_properties> ::= <authid>
<authid> ::= AUTHID {DEFINER | OWNER | CALLER | CURRENT_USER}

```

OR REPLACE 구문을 사용하여 현재의 저장 함수/프로시저를 대체 혹은 새로 생성하는 문장을 작성할 수 있다.

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *procedure_name*: 생성할 저장 프로시저의 이름을 지정한다. (최대 222바이트)
- *parameter_name*: 인자의 이름을 지정한다. (최대 254바이트)
- *sql_type*: 인자의 데이터 타입을 지정한다. 지정할 수 있는 데이터 타입은 **지원하는 인자와 결과 데이터 타입**을 참고한다.
- *default_arg*: 인자의 기본값을 지정한다. **기본값 인자 지정**를 참고한다.
- *authid*: 저장 프로시저의 실행 권한을 지정한다. 자세한 내용은 **소유자 권한과 호출자 권한의 차이**를 참고한다.
- *parameter_comment_string*: 인자 커멘트 문자열을 지정한다.
- *body*: 저장 프로시저의 본문을 지정한다.
- *procedure_comment*: 저장 프로시저의 커멘트 문자열을 지정한다.

저장 프로시저의 커멘트

저장 프로시저의 커멘트를 다음과 같이 제일 뒤에 지정할 수 있다.

```

CREATE FUNCTION Hello() RETURN VARCHAR
AS LANGUAGE JAVA
NAME 'SpCubrid.HelloCubrid() return java.lang.String'
COMMENT 'function comment';

```

저장 프로시저의 인자 뒤에는 다음과 같이 지정할 수 있다.

```

CREATE OR REPLACE FUNCTION test(i in int COMMENT 'arg i')
RETURN int AS LANGUAGE JAVA NAME 'SpTest.testInt(int) return int'
COMMENT 'function test';

```

저장 프로시저의 커멘트는 다음 구문을 실행하여 확인할 수 있다.

```
SELECT sp_name, comment FROM db_stored_procedure;
```

저장 프로시저 인자의 커멘트는 다음 구문을 실행하여 확인할 수 있다.

```
SELECT sp_name, arg_name, comment FROM db_stored_procedure_args;
```

등록된 저장 프로시저의 정보 확인

등록된 저장 프로시저의 정보는 **db_stored_procedure** 시스템 가상 클래스와 **db_stored_procedure_args** 시스템 가상 클래스에서 확인할 수 있다. **db_stored_procedure** 시스템 가상 클래스에서는 저장 프로시저의 이름과 타입, 인자의 수, Java 클래스에 대한 명세, 저장 프로시저의 소유자에 대한 정보를 확인할 수 있다. **db_stored_procedure_args** 시스템 가상 클래스에서는 저장 프로시저에서 사용하는 인자에 대한 정보를 확인할 수 있다.

```
SELECT * FROM db_stored_procedure WHERE sp_type = 'PROCEDURE';
```

```

sp_name          pkg_name          sp_type
return_type      arg_count  lang
authid           is_deterministic
target
owner    code    comment
=====
'athlete_add'    NULL          'PROCEDURE'
'void'           4    'JAVA'
'DEFINER'        'NO'
'Athlete.Athlete(java.lang.String, java.lang.String,
java.lang.String, java.lang.String)'  'DBA'    NULL    NULL

```

```
SELECT * FROM db_stored_procedure_args WHERE sp_name = 'athlete_add';
```

```

sp_name          owner_name          pkg_name
index_of  arg_name          data_type
mode      is_optional          default_value
comment
=====
'athlete_add'    'DBA'

```

```

NULL                                0  'name'
'String'                            'IN'      'NO'
NULL                                NULL
'athlete_add'                       'DBA'
NULL                                1  'gender'
'String'                            'IN'      'NO'
NULL                                NULL
'athlete_add'                       'DBA'
NULL                                2  'nation_code'
'String'                            'IN'      'NO'
NULL                                NULL
'athlete_add'                       'DBA'
NULL                                3  'event'
'String'                            'IN'      'NO'
NULL                                NULL

```

CREATE FUNCTION

CREATE FUNCTION 문을 사용하여 저장 함수를 등록한다.

```

CREATE [OR REPLACE] FUNCTION [schema_name.]function_name
([<parameter_definition> [, <parameter_definition>] ...])
RETURN sql_type
[<function_properties>]
{IS | AS} LANGUAGE <lang>
    <body>
COMMENT 'function_comment';

<parameter_definition> ::= parameter_name [mode] sql_type
[<default_arg>] [COMMENT 'param_comment_string']
    <default_arg> ::= { DEFAULT | = } <default_expr>
<function_properties> ::= [<authid>] [<deterministic>]
    <authid> ::= AUTHID {DEFINER | OWNER | CALLER | CURRENT_USER}
    <deterministic> ::= [NOT DETERMINISTIC | DETERMINISTIC]
<lang> ::= [PLCSQL | JAVA]
<mode> ::= IN | OUT | IN OUT | INOUT

```

- *schema_name*: 스키마 이름을 지정한다(최대 31바이트). 생략하면 현재 세션의 스키마 이름을 사용한다.
- *function_name*: 생성할 저장 함수의 이름을 지정한다(최대 222바이트).
- *parameter_name*: 인자의 이름을 지정한다(최대 254바이트).
- *sql_type*: 인자 또는 리턴 값의 데이터 타입을 지정한다. 지정할 수 있는 데이터 타입은 **지원하는 인자와 결과 데이터 타입**을 참고한다.
- *default_arg*: 인자의 기본값을 지정한다. **기본값 인자 지정**를 참고한다.

- *param_comment_string*: 인자 커멘트 문자열을 지정한다.
- *authid*: 저장 함수의 실행 권한을 지정한다. 자세한 내용은 [소유자 권한과 호출자 권한의 차이](#) 을 참고한다.
- *deterministic*: 저장 함수가 결정적 함수인지 여부를 지정한다. 자세한 내용은 [결정적 함수 사용](#) 을 참고한다.
- *body*: 저장 함수의 본문을 지정한다.
- *function_comment*: 저장 함수의 커멘트 문자열을 지정한다.

저장 함수의 커멘트

저장 함수의 커멘트를 다음과 같이 제일 뒤에 지정할 수 있다.

```
CREATE FUNCTION Hello() RETURN VARCHAR
AS LANGUAGE JAVA
NAME 'SpCubrid.HelloCubrid() return java.lang.String'
COMMENT 'function comment';
```

저장 함수의 인자 뒤에는 다음과 같이 지정할 수 있다.

```
CREATE OR REPLACE FUNCTION test(i in int COMMENT 'arg i')
RETURN int AS LANGUAGE JAVA NAME 'SpTest.testInt(int) return int'
COMMENT 'function test';
```

저장 함수의 커멘트는 다음 구문을 실행하여 확인할 수 있다.

```
SELECT sp_name, comment FROM db_stored_procedure;
```

함수 인자의 커멘트는 다음 구문을 실행하여 확인할 수 있다.

```
SELECT sp_name, arg_name, comment FROM db_stored_procedure_args;
```

등록된 저장 함수의 정보 확인

등록된 저장 함수의 정보는 **db_stored_procedure** 시스템 가상 클래스와 **db_stored_procedure_args** 시스템 가상 클래스에서 확인할 수 있다. **db_stored_procedure** 시스템 가상 클래스에서는 저장 함수의 이름과 타입, 반환 타입, 인자의 수, Java 클래스에 대한 명세, 저장 함수의 소유자에 대한 정보를 확인할 수 있다. **db_stored_procedure_args** 시스템 가상 클래스에서는 저장 함수에서 사용하는 인자에 대한 정보를 확인할 수 있다.

```
SELECT * FROM db_stored_procedure WHERE sp_type = 'FUNCTION';
```

sp_name	pkg_name	sp_type
return_type	arg_count	lang
authid	is_deterministic	
target	owner	
code	comment	
=====		
=====		
=====		
=====		
'hello'	NULL	'FUNCTION'
'STRING'	0	'JAVA'
'DEFINER'	'NO'	'SpCubrid.HelloCubrid()'
return java.lang.String	'DBA'	NULL
'sp_int'	NULL	'FUNCTION'
'INTEGER'	1	'JAVA'
'DEFINER'	'NO'	'SpCubrid.SpInt(int)'
return int	'DBA'	NULL

```
SELECT * FROM db_stored_procedure_args WHERE sp_name = 'sp_int';
```

sp_name	owner_name	pkg_name
index_of	arg_name	data_type
mode	is_optional	default_value
comment		
=====		
=====		
=====		
'sp_int'	'DBA'	
NULL	0	'i'
'INTEGER'	'IN'	'NO'
NULL	NULL	

CREATE FUNCTION DETERMINISTIC

NOT DETERMINISTIC 키워드는 저장 함수가 동일한 입력값에 대해 다른 결과를 반환하는 함수이다. NOT DETERMINISTIC으로 설정된 함수는 부질의 결과 캐시 최적화의 대상에서 제외되며, 매 호출 시 결과가 재계산된다. 기본값은 NOT DETERMINISTIC이다.

DETERMINISTIC 키워드는 저장 함수가 동일한 입력값에 대해 항상 동일한 결과를 반환하는 함수이다. DETERMINISTIC으로 설정된 함수는 상관 부질의 결과 캐시 최적화의 대상으로 처리한다.

상관 부질의 캐시 동작 방식에 대한 자세한 내용은 **서브 쿼리 캐시 (correlated)**을 참고한다.

다음은 DETERMINISTIC을 사용한 저장 함수의 예시이다. 이 예시에서는 상관 부질의를 사용할 때 결과를 캐시하여 성능을 최적화하는 과정을 보여준다.

```
CREATE TABLE dummy_tbl (col1 INTEGER);
INSERT INTO dummy_tbl VALUES (1), (2), (1), (2);

CREATE OR REPLACE FUNCTION pl_csql_not_deterministic (n INTEGER)
RETURN INTEGER AS
BEGIN
    return n + 1;
END;

CREATE OR REPLACE FUNCTION pl_csql_deterministic (n INTEGER) RETURN
INTEGER DETERMINISTIC AS
BEGIN
    return n + 1;
END;

SELECT sp_name, owner, sp_type, is_deterministic from
db_stored_procedure;
```

```
      sp_name                owner
sp_type                is_deterministic
=====
=====
      'pl_csql_not_deterministic'  'DBA'
'FUNCTION'                'NO'
      'pl_csql_deterministic'      'DBA'
'FUNCTION'                'YES'
```

위 예시에서 pl_csql_not_deterministic 함수는 NOT DETERMINISTIC이므로 상관 부질의에서 캐시를 사용하지 않는다. 반면, pl_csql_deterministic 함수는 DETERMINISTIC 키워드가 지정되어 있으므로 상관 부질의 결과를 캐시하여 성능을 최적화할 수 있다.

```
;trace on
SELECT (SELECT pl_csql_not_deterministic (t1.col1) FROM dual) AS
results FROM dummy_tbl t1;
```

```
      results
=====
              2
              3
              2
              3
```

```

=== Auto Trace ===
...
Trace Statistics:
  SELECT (time: 3, fetch: 44, fetch_time: 0, ioread: 0)
    SCAN (table: dba.dummy_tbl), (heap time: 0, fetch: 20,
ioread: 0, readrows: 4, rows: 4)
    SUBQUERY (correlated)
      SELECT (time: 3, fetch: 24, fetch_time: 0, ioread: 0)
        SCAN (table: dual), (heap time: 0, fetch: 16, ioread: 0,
readrows: 4, rows: 4)

```

pl_csql_not_deterministic 함수는 NOT DETERMINISTIC이므로 부질의 결과를 캐시하지 않는다.

```

;trace on
SELECT (SELECT pl_csql_deterministic (t1.col1) FROM dual) AS results
FROM dummy_tbl t1;

```

```

      results
=====
          2
          3
          2
          3

=== Auto Trace ===
...
Trace Statistics:
  SELECT (time: 3, fetch: 36, fetch_time: 0, ioread: 0)
    SCAN (table: dba.dummy_tbl), (heap time: 0, fetch: 20,
ioread: 0, readrows: 4, rows: 4)
    SUBQUERY (correlated)
      SELECT (time: 3, fetch: 16, fetch_time: 0, ioread: 0)
        SCAN (table: dual), (heap time: 0, fetch: 8, ioread: 0,
readrows: 2, rows: 2)
        SUBQUERY_CACHE (hit: 2, miss: 2, size: 150808, status:
enabled)

```

pl_csql_deterministic 함수의 Trace 결과에서는 SUBQUERY_CACHE 항목이 표시되며(hit: 2, miss: 2, size: 150808, status: enabled), 첫 번째 결과 (2), (3)은 캐시에서 miss되었고, 이후 동일한 결과부터는 캐시에서 hit된 것을 확인할 수 있다.

ALTER PROCEDURE

ALTER PROCEDURE 문을 사용하여 저장 프로시저를 재컴파일할 수 있다. 저장 프로시저와 연관된 테이블의 스키마가 변경되더라도 자동으로 재컴파일되지 않으므로, 변경 사항을 반영하려면 사용자가 직접 재컴파일해야 한다.


```
ALTER PROCEDURE [schema_name.]procedure_name COMPILE;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *procedure_name*: 재컴파일할 프로시저의 이름을 지정한다.

참고

소유자를 변경하는 경우, 변경된 소유자로 저장 프로시저를 자동으로 재컴파일한다. 소유자를 변경하기 위해서는 **ALTER ... OWNER**을 참고한다.

다음은 테이블 스키마 변경 후 PL/CSQL을 재컴파일하여 정상적으로 실행할 수 있게 만드는 예이다.

PL/CSQL에 Static SQL을 사용하는 저장 프로시저를 생성한 후 정상적으로 실행되는지 확인한다.

```
CREATE OR REPLACE PROCEDURE proc_stadium_code() AS
  n INTEGER;
BEGIN
  SELECT code INTO n FROM stadium LIMIT 1;
  DBMS_OUTPUT.put_line('code : ' || n);
END;

;server-output on
CALL proc_stadium_code();
```

```
Result
=====
NULL

<DBMS_OUTPUT>
=====
code :30140
```

stadium 테이블의 code 컬럼 타입을 INTEGER에서 VARCHAR로 변경한 후 저장 프로시저를 실행하면 아래와 같은 에러가 발생한다.

```
ALTER TABLE public.stadium MODIFY code VARCHAR;

CALL proc_stadium_code();
```

```
ERROR: Stored procedure execute error:
(line 4, column 3) internal server error
```

컬럼 타입 변경 정보가 기존에 컴파일된 PL/CSQL의 실행코드에 반영되지 않았기 때문에, 저장 프로시저를 재컴파일해야 정상적으로 실행할 수 있다.

```
ALTER PROCEDURE proc_stadium_code COMPILE;

CALL proc_stadium_code();
```

```
Result
=====
NULL

<DBMS_OUTPUT>
=====
code :30140
```

ALTER FUNCTION

ALTER FUNCTION 문을 사용하여 저장 함수를 재컴파일할 수 있다. 저장 함수와 연관된 테이블의 스키마가 변경되더라도 자동으로 재컴파일되지 않으므로, 변경 사항을 반영하려면 사용자가 직접 재컴파일해야 한다.

```
ALTER FUNCTION [schema_name.]function_name COMPILE;
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *function_name*: 재컴파일할 함수의 이름을 지정한다.

참고

소유자를 변경하는 경우, 변경된 소유자로 저장 함수를 자동으로 재컴파일한다. 소유자를 변경하기 위해서는 **ALTER ... OWNER**을 참고한다.

다음은 테이블 스키마 변경 후 PL/CSQL을 재컴파일하여 정상적으로 실행할 수 있게 만드는 예이다.

PL/CSQL에 Static SQL을 사용하는 저장 함수를 생성한 후 정상적으로 실행되는지 확인한다.

```
CREATE OR REPLACE FUNCTION func_stadium_code() RETURN INTEGER AS
n INTEGER;
```

```
BEGIN
  SELECT code INTO n FROM stadium LIMIT 1;
  RETURN n;
END;

CALL func_stadium_code();
```

```
Result
=====
30140
```

stadium 테이블의 code 컬럼 타입을 INTEGER에서 VARCHAR로 변경한 후 저장 함수를 실행하면 아래와 같은 에러가 발생한다.

```
ALTER TABLE public.stadium MODIFY code VARCHAR;

CALL func_stadium_code();
```

```
ERROR: Stored procedure execute error:
(line 4, column 3) internal server error
```

컬럼 타입 변경 정보가 기존에 컴파일된 PL/CSQL의 실행코드에 반영되지 않았기 때문에, 저장 함수를 재컴파일을 수행해야 정상적으로 실행할 수 있다.

```
ALTER FUNCTION func_stadium_code COMPILE;

CALL func_stadium_code();
```

```
Result
=====
30140
```

DROP PROCEDURE

CUBRID에서는 등록된 저장 프로시저를 **DROP PROCEDURE** 구문을 사용하여 삭제할 수 있다. 이 때, 여러 개의 *procedure_name* 을 콤마(,)로 구분하여 한꺼번에 여러 개의 저장 프로시저를 삭제할 수 있다.

```
DROP PROCEDURE [schema_name.]procedure_name [{ ,
[schema_name.]procedure_name , ... }];
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *procedure_name*: 제거할 프로시저의 이름을 지정한다.

```
DROP PROCEDURE hello, public.sp_int;
```

저장 프로시저의 삭제는 프로시저를 등록한 사용자와 DBA의 구성원만 삭제할 수 있다. 예를 들어 'sp_int' 저장 프로시저를 **PUBLIC** 이 등록했다면, **PUBLIC** 또는 **DBA** 의 구성원만이 'sp_int' 저장 프로시저를 삭제할 수 있다.

DROP FUNCTION

CUBRID에서는 등록한 저장 함수를 **DROP FUNCTION** 구문을 사용하여 삭제할 수 있다. 이 때, 여러 개의 *function_name* 을 콤마(,)로 구분하여 한꺼번에 여러 개의 저장 함수를 삭제할 수 있다.

```
DROP FUNCTION [schema_name.]function_name [{,
[schema_name.]function_name} ... ];
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *function_name*: 제거할 함수의 이름을 지정한다.

```
DROP FUNCTION hello, public.sp_int;
```

저장 함수의 삭제는 함수를 등록한 사용자와 DBA의 구성원만 삭제할 수 있다. 예를 들어 'sp_int' 저장 함수를 **PUBLIC** 이 등록했다면, **PUBLIC** 또는 **DBA** 의 구성원만이 'sp_int' 저장 함수를 삭제할 수 있다.

연산자와 함수

논리 연산자

논리 연산자(logical operator)는 피연산자로 불리언(boolean) 연산식이 지정되며, 연산 결과로 **TRUE**, **FALSE**, **NULL** 을 반환한다. 불리언 값이 수식에 사용될 때에는 **TRUE** 는 1, **FALSE** 는 0으로 해석된다.

참고

11.2 버전부터 논리식이 아닌 값(정수·컬럼 등)을 피연산자로 사용하면 “operand must be logical expression.” 오류가 발생한다. 비논리 값을 직접 조건절에 사용하던 구문은 명시적인 비교 연산으로 바뀌어야 한다. 상수 조건 **WHERE 1** 은 **WHERE 1=1** 로, 컬럼 조건 **WHERE column_name** 은 **WHERE column_name <> 0** 으로 작성한다.

논리 연산자의 종류 및 진리표는 아래와 같다.

논리 연산자

논리 연산자	설명	조건식
AND, &&	피연산자가 모두 TRUE 이면 TRUE 를 반환한다.	a AND b
OR, 	피연산자가 모두 NULL 이 아니고, 하나 이상의 피연산자가 TRUE 이면 TRUE 를 반환한다. SQL 구문 관련 파라미터인 pipes_as_concat 파라미터가 no이면, 이중 파이프 기호()를 OR 연산자로 사용할 수 있다.	a OR b
XOR	피연산자가 모두 NULL 이 아니고, 두 피연산자의 값이 다르면 TRUE 를 반환한다.	a XOR b
NOT, !	단항 연산자이며, 피연산자가 FALSE 이면 TRUE , 피연산자가 TRUE 이면 FALSE 를 반환한다.	NOT a

논리 연산자의 진리표

a	b	a AND b	a OR b	NOT a	a XOR b
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
TRUE	NULL	NULL	TRUE	FALSE	NULL
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
FALSE	NULL	FALSE	NULL	TRUE	NULL

비교 연산자

비교 연산자(comparison operator)는 왼쪽 피연산자와 오른쪽 피연산자를 비교하여 1 또는 0을 반환한다. 비교 연산의 피연산자들은 같은 데이터 타입이어야 하므로, 시스템에 의해서 묵시적으로 타입이 변환되거나 사용자에게 의해 명시적으로 타입이 변환되어야 한다.

구문 1

```
<expression> <comparison_operator> <expression>
```

```
<expression> ::=
    bit_string |
    character_string |
    numeric_value |
    date-time_value |
    collection_value |
    NULL
```

```
<comparison_operator> ::=
    = |
    <=> |
    <> |
    != |
    > |
    < |
```

```
>= |
<=
```

구문 2

```
<expression> IS [NOT] NULL
<logical_expression> IS [NOT] <truth_value>

<expression> ::=
    bit_string |
    character_string |
    numeric_value |
    date-time_value |
    collection_value |
    NULL

<truth_value> ::=
    TRUE |
    FALSE |
    UNKNOWN

<logical_expression> ::=
    <expression> <comparison_operator> <expression> |
    <logical_expression> { AND | OR | XOR } <logical_expression>
|
    NOT <logical_expression>

<comparison_operator> ::=
    = |
    <=> |
    <> |
    != |
    > |
    < |
    >= |
    <=
```

• <expression>: 비교할 수식을 선언한다.

- *bit_string*: 비트열에 대하여 불리언(boolean) 연산을 수행할 수 있으며, 모든 비교 연산자를 비트열을 비교하는데 사용할 수 있다. 길이가 같지 않은 두 수식을 비교할 때는 길이가 짧은 비트열의 오른쪽 끝에 0이 추가된다.
- *character_string*: 비교 연산자를 통해 비교할 두 문자열은 같은 문자셋을 가져야 한다. 문자 코드와 연관된 정렬 체계(collation)에 의해 비교 순서가 결정된다. 서로 다른 길이의 문자열을 비교할 때는, 비교 전에 길이가 긴 문자열의 길이와 같아지도록 길이가 짧은 문자열 뒤에 공백을 추가한다.

- *numeric_value*: 모든 숫자 값에 대해 불리언(Boolean)을 수행할 수 있으며, 모든 비교 연산자를 이용하여 비교 연산을 수행할 수 있다. 서로 다른 숫자 타입을 비교할 때에는 시스템이 묵시적으로 타입을 변환한다. 예를 들어, **INTEGER** 값을 **DECIMAL** 값과 비교할 때 시스템은 먼저 **INTEGER**를 **DECIMAL**로 변환한 후 비교한다. **FLOAT**에 대해서 비교할 때에는 **FLOAT**는 시스템 종속적으로 처리되므로 정확한 값이 아니라 범위를 지정해야 한다.
- *date-time_value*: 날짜/시간 값을 같은 타입 간에 비교할 때에 값의 순서는 연대기 순으로 결정된다. 즉, 두 개의 날짜와 시간 값을 비교할 때, 앞의 날짜가 나중 날짜보다 작은 것으로 간주된다. 서로 다른 타입의 날짜/시간 값에 대한 비교 연산은 허용되지 않으므로 명시적 타입 변환이 필요하지만, **DATE**, **TIMESTAMP**, **DATETIME** 타입 간에는 묵시적 타입 변환이 수행되어 비교 연산이 가능하다.
- *collection_value*: 두 **LIST** (= **SEQUENCE**)를 비교할 때에는 **LIST**가 생성되었을 때 사용자가 명시한 순서대로 원소 간의 비교가 이루어진다. **SET**과 **MULTISET**을 포함하는 비교는 적절한 집합 연산자로 오버로딩 된다. **SET**, **MULTISET**, **LIST** 또는 **SEQUENCE**에 대한 비교 연산은 이 장의 뒷부분에서 설명하는 포함 연산자를 이용하여 수행할 수 있다. 자세한 정보는 **포함 연산자**를 참조한다.
- **NULL**: **NULL** 값은 모든 데이터 타입의 값 범위 내에 포함되지 않는다. 따라서, **NULL** 값의 비교는 주어진 값이 **NULL** 값인지 아닌지에 대한 비교만 가능하다. **NULL** 값이 다른 데이터 타입으로 할당될 때 묵시적인 타입 변경은 일어나지 않는다. 예를 들어, **INTEGER** 타입의 칼럼이 **NULL** 값을 가지고 있고 부동 소수점 타입과 비교할 때, 비교하기 전에 **NULL** 값을 **FLOAT** 형으로 변환하지 않는다. **NULL** 값에 대한 비교 연산은 결과를 반환하지 않는다.
- *<logical_expression>*: 불리언(논리) 값으로 평가되는 표현식이다. 비교 연산(예: $a = 1$)이나 논리 연산(예: $(a = 1) \text{ AND } (b = 2)$)의 결과가 해당한다.

다음은 CUBRID에서 지원되는 비교 연산자의 설명 및 리턴 값을 나타낸 표이다.

비교 연산자

비교 연산자	설명	조건식	리턴 값
=	일반 등호이며, 두 피연산자의 값이 같은지 비교한다. 하나 이상의 피연산자가 NULL 이면 NULL 을 반환한다.	$1=2$ $1=NULL$	0 NULL
<=>	NULL safe 등호이며, NULL 을 포함하여 두 피연산자의 값이 같	$1<=>2$ $1<=>NULL$	0 0

비교 연산자	설명	조건식	리턴 값
	은지 비교한다. 피연산자가 모두 NULL 이면 1을 반환한다.		
<>, !=	두 피연산자의 값이 다른지 비교한다. 하나 이상의 피연산자가 NULL 이면 NULL 을 반환한다.	1<>2	1
>	왼쪽 피연산자가 오른쪽 피연산자보다 값이 큰지 비교한다. 하나 이상의 피연산자가 NULL 이면 NULL 을 반환한다.	1>2	0
<	왼쪽 피연산자가 오른쪽 피연산자보다 값이 작은지 비교한다. 하나 이상의 피연산자가 NULL 이면 NULL 을 반환한다.	1<2	1
>=	왼쪽 피연산자가 오른쪽 피연산자보다 값이 크거나 같은지 비교한다. 하나 이상의 피연산자가 NULL 이면 NULL 을 반환한다.	1>=2	0
<=	왼쪽 피연산자가 오른쪽 피연산자보다 값이 작거나 같은지 비교한다. 하나 이상의 피연산자가 NULL	1<=2	1

비교 연산자	설명	조건식	리턴 값
	이면 NULL 을 반환한다.		
IS <i>truth_value</i>	왼쪽 피연산자가 오른쪽 진리값과 같은지 비교한다. 진리값은 TRUE , FALSE , UNKNOWN 이 될 수 있다.	(1=1) IS FALSE	0
IS NOT <i>truth_value</i>	왼쪽 피연산자가 오른쪽 진리값과 다른지 비교한다. 진리값은 TRUE , FALSE , UNKNOWN 이 될 수 있다.	(1=1) IS NOT FALSE	1

참고

11.2 버전부터 **IS [NOT] { TRUE | FALSE | UNKNOWN }**의 왼쪽 피연산자가 논리식이 아니면 “operand must be logical expression.” 오류가 발생한다. 예를 들어 **1 IS FALSE**는 **(1=1) IS FALSE**와 같이 작성해야 한다. 논리식에서 UNKNOWN 진리값은 NULL로 표현되므로, 논리식에 대해 **IS [NOT] UNKNOWN**과 **IS [NOT] NULL**은 동일한 결과를 반환한다.

다음은 비교 연산자를 사용하는 예이다.

```
SELECT (1 <> 0); -- TRUE이므로 1을 출력한다.
SELECT (1 != 0); -- TRUE이므로 1을 출력한다.
SELECT (0.01 = '0.01'); -- 숫자 타입과 문자열 타입을 비교했으므로 에러가 발생한다.
SELECT (1 = NULL); -- NULL을 출력한다.
SELECT (1 <=> NULL); -- FALSE이므로 0을 출력한다.
SELECT (1.000 = 1); -- TRUE이므로 1을 출력한다.
SELECT ('cubrid' = 'CUBRID'); -- 대소문자를 구분하므로 0을 출력한다.
SELECT ('cubrid' = 'cubrid'); -- TRUE이므로 1을 출력한다.
SELECT (SYSTIMESTAMP = CAST(SYSDATETIME AS TIMESTAMP)); -- 명시적으로 타입을 변환하여 비교 연산을 수행한 결과, 1을 출력한다.
SELECT (SYSTIMESTAMP = SYSDATETIME); -- 묵시적으로 타입을 변환하여 비교 연
```

산을 수행한 결과, 0을 출력한다.

```
SELECT (SYSTIMESTAMP <> NULL); -- NULL의 비교 연산을 수행하지 않고 NULL을 반환한다.
```

```
SELECT (SYSTIMESTAMP IS NOT NULL); -- NULL이 아니므로 1을 반환한다.
```

```
SELECT ((1 > 0) IS TRUE); -- TRUE이므로 1을 출력한다.
```

```
SELECT ((1 > 0) IS FALSE); -- FALSE가 아니므로 0을 출력한다.
```

```
SELECT ((1 > 0) IS UNKNOWN); -- UNKNOWN이 아니므로 0을 출력한다.
```

```
SELECT ((NULL = 1) IS UNKNOWN); -- NULL 비교는 UNKNOWN이므로 1을 출력한다.
```

```
SELECT ((1 > 0) IS NOT FALSE); -- FALSE가 아니므로 1을 출력한다.
```

산술 연산자

목차

산술 연산자

7

- 수치형 데이터 타입의 산술 연산과 타입 변환 337
- 날짜/시간 데이터 타입의 산술 연산과 타입 변환 341
- 타임존 파라미터들과 관련된 동작 12

산술 연산자는 덧셈, 뺄셈, 곱셈, 나눗셈을 위한 이항(binary) 연산자와 양수, 음수를 나타내기 위한 단항(unary) 연산자가 있다. 양수/음수의 부호를 나타내는 단항 연산자의 연산 우선순위가 이항 연산자보다 높다.

```
<expression> <mathematical_operator> <expression>
```

```
<expression> ::=
    bit_string |
    character_string |
    numeric_value |
    date-time_value |
    collection_value |
    NULL
```

```
<mathematical_operator> ::=
    <set_arithmetic_operator> |
    <arithmetic_operator>
```

```
<arithmetic_operator> ::=
    + |
    - |
    * |
```

```

{ / | DIV } |
{ % | MOD }

<set_arithmetic_operator> ::=
    UNION |
    DIFFERENCE |
    { INTERSECT | INTERSECTION }

```

- *<expression>*: 연산을 수행할 수식을 선언한다.
- *<mathematical_operator>*: 수학적 연산을 지정하는 연산자로서, 산술 연산자와 집합 연산자가 있다.
 - *<set_arithmetic_operator>*: 컬렉션 타입의 피연산자에 대해 합집합, 차집합, 교집합을 수행하는 집합 산술 연산자이다.
 - *<arithmetic_operator>*: 사칙 연산을 수행하기 위한 연산자이다.

다음은 CUBRID가 지원하는 산술 연산자의 설명 및 리턴 값을 나타낸 표이다.

산술 연산자

산술 연산자	설명	연산식	리턴 값
+	더하기 연산	1+2	3
-	빼기 연산	1-2	-1
*	곱하기 연산	1*2	2
/	나누기 연산 후, 몫을 반환한다.	1/2.0	0.500000000
DIV	나누기 연산 후, 몫을 반환한다. 피연산자는 정수 타입이어야 하며, 정수를 반환한다.	1 DIV 2	0
%, MOD	나누기 연산 후, 나머지를 반환한다. 피연	1 % 2 1 MOD 2	1

산술 연산자	설명	연산식	리턴 값
	산자는 정수 타입이어야 하며, 정수를 반환한다. 피연산자가 실수이면 MOD 함수를 이용한다.		

수치형 데이터 타입의 산술 연산과 타입 변환

모든 수치형 데이터 타입을 산술 연산에 사용할 수 있으며, 연산 결과 타입은 피연산자의 데이터 타입과 연산의 종류에 따라 다르다. 아래는 피연산자 타입별 덧셈/뺄셈/곱셈 연산의 결과 데이터 타입을 정리한 표이다.

피연산자의 타입별 결과 데이터 타입

	INT	NUMERIC	FLOAT	DOUBLE
INT	INT 또는 BIGINT	NUMERIC	FLOAT	DOUBLE
NUMERIC	NUMERIC	NUMERIC (p와 s도 변환됨)	DOUBLE	DOUBLE
FLOAT	FLOAT	DOUBLE	FLOAT	DOUBLE
DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE

피연산자가 모두 동일한 데이터 타입이면 연산 결과의 타입이 변환되지 않으나, 나누기 연산의 경우 예외적으로 타입이 변환되므로 주의해야 한다. 분모, 즉 제수(divisor)가 0이면 에러가 발생한다.

아래는 피연산자가 모두 **NUMERIC** 타입인 경우, 연산 결과의 전체 자릿수(p)와 소수점 아래 자릿수(s)를 정리한 표이다.

NUMERIC 타입의 연산 결과

연산	결과의 최대 자릿수	결과의 소수점 이하 자릿수
$N(p1, s1) + N(p2, s2)$	$\max(p1 - s1, p2 - s2) + \max(s1, s2) + 1$	
$N(p1, s1) - N(p2, s2)$	$\max(p1 - s1, p2 - s2) + \max(s1, s2)$	
$N(p1, s1) * N(p2, s2)$	$p1 + p2 + 1$	$s1 + s2$
$N(p1, s1) / N(p2, s2)$	$Pt = (p1 - s1) + s2 + \max(9, \max(s1, s2))$ $Pt = (Pt > 38) ? 38 : Pt$	$St = \max(9, \max(s1, s2))$ $St = (Pt > 38) ? \min(9, St - (Pt - 38)) : St$

산술 연산자가 '/'인 경우 시스템 파라미터 oracle_compat_number_behavior의 설정에 따라 결과의 타입이 달라진다. 즉, 정수/정수의 경우 oracle_compat_number_behavior가 yes로 설정된 경우, 결과의 타입은 numeric이 되고 **no**로 설정된 경우는 **피연산자의 타입별 결과 데이터 타입 규칙**에 따른다.

예제

```
--int * int
SELECT 123*123;
```

```
      123*123
=====
      15129
```

```
-- int * int returns overflow error
SELECT (1234567890123*1234567890123);
```

```
ERROR: Data overflow on data type bigint.
```

```
-- int * numeric returns numeric type
SELECT (1234567890123*CAST(1234567890123 AS NUMERIC(15,2)));
```

```
(1234567890123* cast(1234567890123 as numeric(15,2)))
=====
1524157875322755800955129.00
```

```
-- int * float returns float type
SELECT (1234567890123*CAST(1234567890123 AS FLOAT));
```

```
(1234567890123* cast(1234567890123 as float))
=====
1.524158e+024
```

```
-- int * double returns double type
SELECT (1234567890123*CAST(1234567890123 AS DOUBLE));
```

```
(1234567890123* cast(1234567890123 as double))
=====
1.524157875322756e+024
```

```
-- numeric * numeric returns numeric type
SELECT (CAST(1234567890123 AS NUMERIC(15,2))*CAST(1234567890123 AS
NUMERIC(15,2)));
```

```
( cast(1234567890123 as numeric(15,2))* cast(1234567890123 as
numeric(15,2)))
=====
1524157875322755800955129.0000
```

```
-- numeric * float returns double type
SELECT (CAST(1234567890123 AS NUMERIC(15,2))*CAST(1234567890123 AS
FLOAT));
```

```
( cast(1234567890123 as numeric(15,2))* cast(1234567890123 as
float))
=====
==
1.524157954716582e+024
```

```
-- numeric * double returns double type
SELECT (CAST(1234567890123 AS NUMERIC(15,2))*CAST(1234567890123 AS
DOUBLE));
```

```
( cast(1234567890123 as numeric(15,2))* cast(1234567890123 as
double))
=====
===
1.524157875322756e+024
```

```
-- float * float returns float type
SELECT (CAST(1234567890123 AS FLOAT)*CAST(1234567890123 AS FLOAT));
```

```
( cast(1234567890123 as float)* cast(1234567890123 as float))
=====
1.524158e+024
```

```
-- float * double returns double type
SELECT (CAST(1234567890123 AS FLOAT)*CAST(1234567890123 AS DOUBLE));
```

```
( cast(1234567890123 as float)* cast(1234567890123 as double))
=====
1.524157954716582e+024
```

```
-- double * double returns double type
SELECT (CAST(1234567890123 AS DOUBLE)*CAST(1234567890123 AS DOUBLE));
```

```
( cast(1234567890123 as double)* cast(1234567890123 as double))
=====
1.524157875322756e+024
```

```
csql> ;get oracle_compat_number_behavior
oracle_compat_number_behavior=n
```

```
-- int / int returns int type without type conversion or rounding
SELECT 100100/100000;
```



```
100100/100000
=====
1
```

```
csql> ;get oracle_compat_number_behavior
oracle_compat_number_behavior=n

-- int / int returns int type without type conversion or rounding
SELECT 100100/200200;
```

```
100100/200200
=====
0
```

```
csql> ;get oracle_compat_number_behavior
oracle_compat_number_behavior=y

-- int / int returns numeric type with oracle_compat_number_behavior
SELECT 1/2;
```

```
1/2
=====
0.5
```

```
-- int / zero returns error
SELECT 100100/(100100-100100);
```

```
ERROR: Attempt to divide by zero.
```

날짜/시간 데이터 타입의 산술 연산과 타입 변환

피연산자가 모두 날짜/시간 데이터 타입이면 뺄셈 연산이 가능하며, 리턴 값의 타입은 **BIGINT** 이다. 이때 피연산자의 타입에 따라 연산 단위가 다르므로 주의한다. 날짜/시간 데이터 타입과 정수는 덧셈 및 뺄셈 연산이 가능하며, 이때 연산 단위와 리턴 값의 타입은 날짜/시간 데이터 타입을 따른다.

아래는 피연산자의 타입별로 허용하는 연산과 연산 결과의 데이터 타입을 정리한 표이다.

피연산자의 타입별 허용 연산과 결과 데이터 타입

	TIME (초 단위)	DATE (일 단위)	TIMESTAMP (초 단위)	DATETIME (밀리초 단위)	INT
TIME	뿔셈만 허용. BIGINT	X	X	X	덧셈, 뿔셈 허용. TIME
DATE	X	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	덧셈, 뿔셈 허용. DATE
TIMESTAMP	X	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	덧셈, 뿔셈 허용. TIMESTAMP
DATETIME	X	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	뿔셈만 허용. BIGINT	덧셈, 뿔셈 허용. DATETIME
INT	덧셈, 뿔셈 허용 TIME	덧셈, 뿔셈 허용. DATE	덧셈, 뿔셈 허용. TIMESTAMP	덧셈, 뿔셈 허용. DATETIME	모든 산술 연산 허용

참고

날짜/시간 산술 연산의 인자 중 하나라도 **NULL** 이 포함되어 있으면 수식의 결과로 **NULL** 이 반환된다.

예제

```
-- initial systimestamp value
SELECT SYSDATETIME;
```

```
SYSDATETIME
=====
07:09:52.115 PM 01/14/2010
```

```
-- time type + 10(seconds) returns time type
SELECT (CAST (SYSDATETIME AS TIME) + 10);
```

```
( cast( SYS_DATETIME  as time)+10)
=====
07:10:02 PM
```

```
-- date type + 10 (days) returns date type
SELECT (CAST (SYSDATETIME AS DATE) + 10);
```

```
( cast( SYS_DATETIME  as date)+10)
=====
01/24/2010
```

```
-- timestamp type + 10(seconds) returns timestamp type
SELECT (CAST (SYSDATETIME AS TIMESTAMP) + 10);
```

```
( cast( SYS_DATETIME  as timestamp)+10)
=====
07:10:02 PM 01/14/2010
```

```
-- systimestamp type + 10(milliseconds) returns systimestamp type
SELECT (SYSDATETIME + 10);
```

```
( SYS_DATETIME +10)
=====
07:09:52.125 PM 01/14/2010
```

```
SELECT DATETIME '09/01/2009 03:30:30.001 pm'- TIMESTAMP '08/31/2009
03:30:30 pm';
```

```
datetime '09/01/2009 03:30:30.001 pm'-timestamp '08/31/2009
03:30:30 pm'
=====
86400001
```

```
SELECT TIMESTAMP '09/01/2009 03:30:30 pm' - TIMESTAMP '08/31/2009
03:30:30 pm';
```

```
timestamp '09/01/2009 03:30:30 pm' - timestamp '08/31/2009 03:30:30
pm'
=====
86400
```

타임존 파라미터들과 관련된 동작

TIMESTAMP 및 TIMESTAMP WITH LOCAL TIME ZONE 데이터 타입은 내부적으로 UNIX epoch 값(1970년 이후 경과한 시간(초))으로 저장되며, 윤초를 사용(tz_leap_second_support가 yes로 설정, **타임존 파라미터** 참고)할 경우 가상 날짜-시간 값을 포함할 수 있다.

Virtual date-time	Unix timestamp
2008-12-31 23:59:58	-> 79399951
2008-12-31 23:59:59	-> 79399952
2008-12-31 23:59:60	-> 79399953
-> not real date (introduced by leap second)	
2009-01-01 00:00:00	-> 79399954
2009-01-01 00:00:01	-> 79399955

TIMESTAMP 및 TIMESTAMPTZ 값이 포함된 산술 연산은 Unix epoch 값에서 바로 수행되며, 존재하지 않는 날짜/시간 값에 해당하는 Unix epoch 값이 허용된다.

```
SELECT TIMESTAMPTZ'2008-12-31 23:59:59 UTC'=TIMESTAMPTZ'2008-12-31
23:59:59 UTC'+1;
```

```
timestamp tztz '2008-12-31 23:59:59 UTC'=timestamp tztz '2008-12-31
23:59:59 UTC'+1
=====
=====
0
```

따라서 위의 비교는 Unix 타임스탬프 79399952와 79399953을 비교하는 것과 같지만 동일한 값이 TIMESTAMPTZ로 사용되면 다음과 같이 동일하다.

```
SELECT TIMESTAMPTZ'2008-12-31 23:59:59 UTC'=TIMESTAMPTZ'2008-12-31
23:59:59 UTC'+1;
```

```
timestampz '2008-12-31 23:59:59 UTC'=timestampz '2008-12-31
23:59:59 UTC'+1
=====
=====
```

1

화면에는 다음과 같은 불일치가 나타난다.

```
SELECT TIMESTAMPTZ'2008-12-31 23:59:59 UTC'+1;
```

```
timestampz '2008-12-31 23:59:59 UTC'+1
=====
11:59:59 PM 12/31/2008 Etc/UTC UTC
```

Unix 타임스탬프 값 79399953과 관련된 '2008-12-31 23:59:60 UTC'는 실제 날짜가 아니기 때문에 바로 이전 값이 사용되지만 내부적으로는 '2008-12-31 23:59:60 UTC' 값과 동일하다.

TIMESTAMP WITH TIME ZONE 데이터 타입은 UNIX 타임스탬프와 타임존 식별자를 모두 포함한다. UNIX 타임스탬프 부분 값에서도 TIMESTAMPTZ에 대한 연산을 수행할 수 있으나, 이 경우 자동 조정 연산이 이어서 수행된다. 지역, 오프셋, 서머타임 등의 타임존 식별자를 포함하려면 TIMESTAMPTZ 객체의 날짜-시간이 유효해야 한다. timestampz'2008-12-31 23:59:59 UTC'+1의 경우 유효하지 않은 날짜-시간(79399953,UTC) 대신 '2008-12-31 23:59:59 UTC'에 해당하는 (79399952,UTC)로 자동 변환된다.

DATETIMETZ 및 TIMESTAMPTZ를 포함하는 산술 연산 후에는, CUBRID에서 다음과 관련된 결과 값의 자동 조정을 수행한다.

- 타임존 식별자 조정: 타임존이 포함된 날짜에 시간(초)을 더하면 내부적으로 저장된 오프셋 규칙과 서머타임 규칙이 변경될 수 있으므로, 이에 따라 타임존 식별자를 갱신 한다.
- Unix 타임스탬프 조정(TIMESTAMPTZ에만 해당): 가상의 날짜-시간 값(윤초가 활성화된 경우)이 항상 바로 이전 Unix 타임스탬프 값으로 변환된다.

집합 산술 연산자

SET, MULTiset, LIST

컬렉션 타입(**SET**, **MULTiset**, **LIST** (= **SEQUENCE**)) 데이터에 대해 합집합, 차집합, 교집합을 구하기 위해서 각각 +, -, * 연산자를 사용할 수 있다.

```
<value_expression> <set_arithmetic_operator> <value_expression>
<value_expression> ::=
```

```

collection_value |
NULL

<set_arithmetic_operator> ::=
+ (합집합) |
- (차집합) |
* (교집합)

```

다음은 컬렉션 타입이 피연산자인 경우, 연산별 결과 데이터 타입을 나타낸 표이다.

피연산자의 타입별 결과 데이터 타입

	SET	MULTISET	LIST
SET	+, -, *: SET	+, -, *: MULTISET	+, -, *: MULTISET
MULTISET	+, -, *: MULTISET	+, -, *: MULTISET	+, -, *: MULTISET
LIST (=SEQUENCE)	+, -, *: MULTISET	+, -, *: MULTISET	+ : LIST -, * : MULTISET

다음은 컬렉션 타입을 가지고 산술 연산을 수행하는 예이다.

```

SELECT ((CAST ({3,3,3,2,2,1} AS SET))+ (CAST ({4,3,3,2} AS
MULTISET)));

```

```

(( cast({3, 3, 3, 2, 2, 1} as set))+ ( cast({4, 3, 3, 2} as
multiset)))
=====
{1, 2, 2, 3, 3, 3, 4}

```

```

SELECT ((CAST ({3,3,3,2,2,1} AS MULTISET))+ (CAST ({4,3,3,2} AS
MULTISET)));

```

```

(( cast({3, 3, 3, 2, 2, 1} as multiset))+ ( cast({4, 3, 3, 2} as
multiset)))
=====
{1, 2, 2, 2, 3, 3, 3, 3, 3, 4}

```

```
SELECT ((CAST ({3,3,3,2,2,1} AS LIST))+(CAST ({4,3,3,2} AS
MULTISET)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as sequence))+( cast({4, 3, 3, 2} as
multiset)))
=====
{1, 2, 2, 2, 3, 3, 3, 3, 3, 4}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS SET))-(CAST ({4,3,3,2} AS
MULTISET)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as set))-( cast({4, 3, 3, 2} as
multiset)))
=====
{1}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS MULTISET))-(CAST ({4,3,3,2} AS
MULTISET)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as multiset))-( cast({4, 3, 3, 2} as
multiset)))
=====
{1, 2, 3}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS LIST))-(CAST ({4,3,3,2} AS
MULTISET)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as sequence))-( cast({4, 3, 3, 2} as
multiset)))
=====
{1, 2, 3}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS SET))*(CAST ({4,3,3,2} AS
MULTISET)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as set))*( cast({4, 3, 3, 2} as
multiset)))
```

```
=====
{2, 3}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS MULTiset))*(CAST ({4,3,3,2} AS
MULTiset)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as multiset))*( cast({4, 3, 3, 2} as
multiset)))
=====
{2, 3, 3}
```

```
SELECT ((CAST ({3,3,3,2,2,1} AS LIST))*(CAST ({4,3,3,2} AS
MULTiset)));
```

```
(( cast({3, 3, 3, 2, 2, 1} as sequence))*( cast({4, 3, 3, 2} as
multiset)))
=====
{2, 3, 3}
```

변수에 컬렉션 값 할당

컬렉션 값을 변수에 할당하기 위해서는 외부 질의가 하나의 행만을 반환해야 한다.

다음은 컬렉션 값을 변수에 할당하는 방법을 나타내는 예제이다. 다음과 같이 외부 질의는 하나의 행만을 반환해야 한다.

```
CREATE TABLE people (
    ssn VARCHAR(10),
    name VARCHAR(255)
);

INSERT INTO people
VALUES ('1234', 'Ken'), ('5678', 'Dan'), ('9123', 'Jones');

SELECT SET(SELECT name
FROM people
WHERE ssn in {'1234', '5678'})
TO :name_group;
```


문장 집합 연산자

UNION, DIFFERENCE, INTERSECTION

피연산자로 지정된 하나 이상의 질의문의 결과에 대해 합집합(**UNION**), 차집합(**DIFFERENCE**), 교집합(**INTERSECTION**)을 구하기 위하여 문장 집합 연산자(Statement Set Operator)를 이용한다. 단, 두 질의문의 대상 테이블에서 조회하고자 하는 데이터 타입이 동일하거나, 묵시적으로 변환 가능해야 한다.

```
query_term statement_set_operator [qualifier] <query_term>
[statement_set_operator [qualifier] <query_term>];

<query_term> ::=
    query_specification
    subquery
```

• *qualifier*

- DISTINCT, DISTINCTROW 또는 UNIQUE(결과로 반환되는 인스턴스가 서로 다르다는 것을 보장)
- ALL (모든 인스턴스가 반환, 중복 허용)

• *statement_set_operator*

- UNION (합집합)
- DIFFERENCE (차집합)
- INTERSECT | INTERSECTION (교집합)

다음은 CUBRID가 지원하는 문장 집합 연산자를 나타낸 표이다.

문장 집합 연산자

문장 집합 연산자	설명	비고
UNION	합집합 중복을 허용하지 않음	UNION ALL 이면 중복된 값을 포함한 모든 결과 인스턴스 출력
DIFFERENCE	차집합 중복을 허용하지 않음	EXCEPT 연산자와 동일. DIFFERENCE ALL 이면 중복된

문장 집합 연산자	설명	비고
		값을 포함한 모든 결과 인스턴스 출력
INTERSECTION	교집합 중복을 허용하지 않음	INTERSECT 연산자와 동일. INTERSECTION ALL 이면 중복된 값을 포함한 모든 결과 인스턴스 출력

다음은 문장 집합 연산자를 가지고 질의를 수행하는 예이다.

```
CREATE TABLE nojoin_tbl_1 (ID INT, Name VARCHAR(32));

INSERT INTO nojoin_tbl_1 VALUES (1, 'Kim');
INSERT INTO nojoin_tbl_1 VALUES (2, 'Moy');
INSERT INTO nojoin_tbl_1 VALUES (3, 'Jonas');
INSERT INTO nojoin_tbl_1 VALUES (4, 'Smith');
INSERT INTO nojoin_tbl_1 VALUES (5, 'Kim');
INSERT INTO nojoin_tbl_1 VALUES (6, 'Smith');
INSERT INTO nojoin_tbl_1 VALUES (7, 'Brown');

CREATE TABLE nojoin_tbl_2 (id INT, Name VARCHAR(32));

INSERT INTO nojoin_tbl_2 VALUES (5, 'Kim');
INSERT INTO nojoin_tbl_2 VALUES (6, 'Smith');
INSERT INTO nojoin_tbl_2 VALUES (7, 'Brown');
INSERT INTO nojoin_tbl_2 VALUES (8, 'Lin');
INSERT INTO nojoin_tbl_2 VALUES (9, 'Edwin');
INSERT INTO nojoin_tbl_2 VALUES (10, 'Edwin');

--Using UNION to get only distinct rows
SELECT id, name FROM nojoin_tbl_1
UNION
SELECT id, name FROM nojoin_tbl_2;
```

```

      id  name
=====
      1  'Kim'
      2  'Moy'
      3  'Jonas'
      4  'Smith'
      5  'Kim'
      6  'Smith'
      7  'Brown'
```

```

      8  'Lin'
      9  'Edwin'
     10  'Edwin'

```

```

--Using UNION ALL not eliminating duplicate selected rows
SELECT id, name FROM nojoin_tbl_1
UNION ALL
SELECT id, name FROM nojoin_tbl_2;

```

```

      id  name
=====
      1  'Kim'
      2  'Moy'
      3  'Jonas'
      4  'Smith'
      5  'Kim'
      6  'Smith'
      7  'Brown'
      5  'Kim'
      6  'Smith'
      7  'Brown'
      8  'Lin'
      9  'Edwin'
     10  'Edwin'

```

```

--Using DEFFERENCE to get only rows returned by the first query but
not by the second
SELECT id, name FROM nojoin_tbl_1
DIFFERENCE
SELECT id, name FROM nojoin_tbl_2;

```

```

      id  name
=====
      1  'Kim'
      2  'Moy'
      3  'Jonas'
      4  'Smith'

```

```

--Using INTERSECTION to get only those rows returned by both queries
SELECT id, name FROM nojoin_tbl_1
INTERSECT
SELECT id, name FROM nojoin_tbl_2;

```

```

      id  name
=====
      5  'Kim'
      6  'Smith'
      7  'Brown'

```

포함 연산자

목차

포함 연산자	7
● SETEQ	357
● SETNEQ	358
● SUPERSET	359
● SUPERSETEQ	360
● SUBSET	361
● SUBSETEQ	362

컬렉션 타입인 피연산자 간 비교 연산을 수행하여 포함(containment) 관계를 확인하기 위해 포함 연산자가 사용된다. 피연산자로 컬렉션 타입 또는 부질의(subquery)를 지정할 수 있으며, 두 피연산자의 포함 관계(동일하다/다르다/부분집합이다/진부분집합이다)에 따라 **TRUE** 또는 **FALSE** 를 반환한다.

```

<collection_operand> <containment_operator> <collection_operand>

<collection_operand> ::=
    <set> |
    <multiset> |
    <sequence> (또는 <list>) |
    <subquery> |
    NULL

<containment_operator> ::=
    SETEQ |
    SETNEQ |
    SUPERSET |

```

SUBSET | SUPERSETEQ | SUBSETEQ

• *<collection_operand>*: 피연산자로 지정될 수 있는 수식은 하나의 집합 값 속성(SET-valued attribute)이거나, 집합 연산자(SET operator)를 지닌 산술 수식(arithmetic expression)이거나, 중괄호로 둘러싸인 집합 값이다. 이때, 중괄호로 둘러싸인 집합 값은 타입을 명시하지 않을 경우 기본적으로 **LIST** 타입으로 처리한다.

피연산자로 부질의가 지정될 수 있으며, 컬렉션 타입이 아닌 칼럼을 조회하는 경우에는 **SET** (*subquery*)과 같이 해당 부질의에 컬렉션 타입 키워드를 붙여야 한다. 부질의에서 조회하는 칼럼은 하나의 집합만 결과로 반환해야 나머지 피연산자 집합과 비교할 수 있다.

컬렉션 원소의 타입이 오브젝트이면, 오브젝트의 내용이 아닌 객체 식별자(OID, object identifier)에 대해 비교한다. 예를 들어, 같은 속성 값을 갖고 OID가 다른 두 오브젝트는 서로 다른 것으로 간주한다.

◦ **NULL**: 비교 대상이 되는 피연산자 중 어느 하나가 **NULL** 인 경우, **NULL** 이 반환된다.

다음은 CUBRID가 지원하는 포함 연산자에 관한 설명 및 리턴 값을 나타낸 표이다.

CUBRID가 지원하는 포함 연산자

포함 연산자	설명	조건식	리턴 값
A SETEQ B	A = B: 집합 A와 집합 B의 원소가 서로 같다.	{1,2} SETEQ {1,2,2}	0
A SETNEQ B	A <> B: 집합 A와 집합 B의 원소가 같지 않다.	{1,2} SETNEQ {1,2,3}	1
A SUPERSET B	A > B: 집합 B는 집합 A의 진 부분집합이다.	{1,2} SUPERSET {1,2,3}	0
A SUBSET B	A < B: 집합 A는 집합 B의 진 부분집합이다.	{1,2} SUBSET {1,2,3}	1
A SUPERSETEQ B		{1,2} SUPERSETEQ {1,2,3}	0

포함 연산자	설명	조건식	리턴 값
	A >= B: 집합 B는 집합 A의 부분 집합이다.		
A SUBSETEQ B	A <= B: 집합 A는 집합 B의 부분 집합이다.	{1,2} {1,2,3}	SUBSETEQ 1

다음은 포함 연산자를 이용하는 경우, 피연산자의 타입별 연산 가능 여부 및 타입 변환 여부를 나타낸 표이다.

포함 연산자의 피연산자 타입별 연산 가능 여부

	SET	MULTISET	LIST(=SEQUENCE)
SET	연산 가능	연산 가능	연산 가능
MULTISET	연산 가능	연산 가능	연산 가능 (LIST 타입은 MULTISET 타입으로 변환됨)
LIST(=SEQUENCE)	연산 가능	연산 가능 (LIST 타입은 MULTISET 타입으로 변환됨)	일부 연산만 가능 (SETEQ, SETNEQ) 나머지 연산은 에러 발생

```
--empty set is a subset of any set
SELECT ({} SUBSETEQ (CAST ({3,1,2} AS SET)));
```

```
Result
=====
1
```

```
--operation between set type and null returns null
SELECT ((CAST ({3,1,2} AS SET)) SUBSETEQ NULL);
```

```
Result
=====
NULL
```

```
--{1,2,3} seteq {1,2,3} returns true
SELECT ((CAST ({3,1,2} AS SET)) SETEQ (CAST ({1,2,3,3} AS SET)));
```

```
Result
=====
1
```

```
--{1,2,3} seteq {1,2,3,3} returns false
SELECT ((CAST ({3,1,2} AS SET)) SETEQ (CAST ({1,2,3,3} AS
MULTISET)));
```

```
Result
=====
0
```

```
--{1,2,3} setneq {1,2,3,3} returns true
SELECT ((CAST ({3,1,2} AS SET)) SETNEQ (CAST ({1,2,3,3} AS
MULTISET)));
```

```
Result
=====
1
```

```
--{1,2,3} subseteq {1,2,3,4} returns true
SELECT ((CAST ({3,1,2} AS SET)) SUBSETEQ (CAST ({1,2,4,4,3} AS
SET)));
```

```
Result
=====
1
```

```
--{1,2,3} subseteq {1,2,3,4,4} returns true
SELECT ((CAST ({3,1,2} AS SET)) SUBSETEQ (CAST ({1,2,4,4,3} AS
MULTISET)));
```

```
Result
=====
1
```

```
--{1,2,3} subseteq {1,2,4,4,3} returns true
SELECT ((CAST ({3,1,2} AS SET)) SUBSETEQ (CAST ({1,2,4,4,3} AS
LIST)));
```

```
Result
=====
0
```

```
--{1,2,3} subseteq {1,2,3,4,4} returns true
SELECT ((CAST ({3,1,2} AS SET)) SUBSETEQ (CAST ({1,2,3,4,4} AS
LIST)));
```

```
Result
=====
1
```

```
--{3,1,2} seteq {3,1,2} returns true
SELECT ((CAST ({3,1,2} AS LIST)) SETEQ (CAST ({3,1,2} AS LIST)));
```

```
Result
=====
1
```

```
--error occurs because LIST subseteq LIST is not supported
SELECT ((CAST ({3,1,2} AS LIST)) SUBSETEQ (CAST ({3,1,2} AS LIST)));
```

```
ERROR: ' subseteq ' operator is not defined on types sequence and
sequence.
```


SETEQ

SETEQ 연산자는 첫 번째 피연산자와 두 번째 피연산자가 동일한 경우 **TRUE** (1)을 반환한다. 모든 컬렉션 타입에 대해 비교 연산을 수행할 수 있다.

```
collection_operand SETEQ collection_operand
```

```
--creating a table with SET type address column and LIST type
zip_code column

CREATE TABLE contain_tbl (id INT PRIMARY KEY, name CHAR(10), address
SET VARCHAR(20), zip_code LIST INT);
INSERT INTO contain_tbl VALUES(1, 'Kim', {'country', 'state'},{1, 2,
3});
INSERT INTO contain_tbl VALUES(2, 'Moy', {'country', 'state'},{3, 2,
1});
INSERT INTO contain_tbl VALUES(3, 'Jones', {'country', 'state',
'city'},{1,2,3,4});
INSERT INTO contain_tbl VALUES(4, 'Smith', {'country', 'state',
'city', 'street'},{1,2,3,4});
INSERT INTO contain_tbl VALUES(5, 'Kim', {'country', 'state',
'city', 'street'},{1,2,3,4});
INSERT INTO contain_tbl VALUES(6, 'Smith', {'country', 'state',
'city', 'street'},{1,2,3,5});
INSERT INTO contain_tbl VALUES(7, 'Brown', {'country', 'state',
'city', 'street'},{});

--selecting rows when two collection_operands are same in the WHERE
clause
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SETEQ {'country','state', 'city'};
```

```
      id  name      address      zip_code
=====
=====
      3  'Jones      '      {'city', 'country', 'state'}
{1, 2, 3, 4}

1 row selected.
```

```
--selecting rows when two collection_operands are same in the WHERE
clause
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SETEQ {1,2,3};
```

```

            id  name                address                zip_code
=====
=====
            1  'Kim                '                {'country', 'state'} {1, 2, 3}

1 rows selected.

```

SETNEQ

SETNEQ 연산자는 첫 번째 피연산자와 두 번째 피연산자가 동일하지 않은 경우에 **TRUE** (1)을 반환한다. 모든 컬렉션 타입에 대해 비교 연산을 수행할 수 있다.

```
collection_operand SETNEQ collection_operand
```

```

--selecting rows when two collection_operands are not same in the
WHERE clause
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SETNEQ {'country','state', 'city'};

```

```

            id  name                address                zip_code
=====
=====
            1  'Kim                '                {'country', 'state'} {1, 2, 3}
            2  'Moy                '                {'country', 'state'} {3, 2, 1}
            4  'Smith              '                {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
            5  'Kim                '                {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
            6  'Smith              '                {'city', 'country', 'state',
'street'} {1, 2, 3, 5}
            7  'Brown              '                {'city', 'country', 'state',
'street'} {}

6 rows selected.

```

```

--selecting rows when two collection_operands are not same in the
WHERE clause
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SETNEQ {1,2,3};

```

```

            id  name                address                zip_code
=====
=====

```

```

      2 'Moy      ' {'country', 'state'} {3, 2, 1}
      3 'Jones    ' {'city', 'country', 'state'}
{1, 2, 3, 4}
      4 'Smith    ' {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
      5 'Kim      ' {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
      6 'Smith    ' {'city', 'country', 'state',
'street'} {1, 2, 3, 5}
      7 'Brown    ' {'city', 'country', 'state',
'street'} {}

```

SUPERSET

SUPERSET 연산자는 첫 번째 피연산자가 두 번째 피연산자의 모든 원소를 포함하는 경우, 즉 두 번째 피연산자가 첫 번째 피연산자의 진부분집합인 경우 **TRUE** (1)을 반환한다. 피연산자 집합이 서로 동일한 경우에는 **FALSE** (0)을 반환한다. 단, 피연산자가 모두 **LIST** 타입인 경우에는 **SUPERSET** 연산을 지원하지 않는다.

```
collection_operand SUPERSET collection_operand
```

```

--selecting rows when the first operand is a superset of the second
operand and they are not same
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SUPERSET {'country','state','city'};

```

```

      id  name      address                                zip_code
=====
=====
      4  'Smith    ' {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
      5  'Kim      ' {'city', 'country', 'state',
'street'} {1, 2, 3, 4}
      6  'Smith    ' {'city', 'country', 'state',
'street'} {1, 2, 3, 5}
      7  'Brown    ' {'city', 'country', 'state',
'street'} {}

```

```

--SUPERSET operator cannot be used for comparison between LIST and
LIST type values
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUPERSET {1,2,3};

```

```
ERROR: ' superset ' operator is not defined on types sequence and
sequence.
```

```
--Comparing operands with a SUPERSET operator after casting LIST
type as SET type
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUPERSET (CAST ({1,2,3} AS SET));
```

id	name	address	zip_code
3	'Jones	'	{'city', 'country', 'state'}
{1, 2, 3, 4}	4	'Smith	'
	'street'}	{1, 2, 3, 4}	{'city', 'country', 'state',
5	'Kim	'	{'city', 'country', 'state',
'street'}	{1, 2, 3, 4}		{'city', 'country', 'state',
6	'Smith	'	{'city', 'country', 'state',
'street'}	{1, 2, 3, 5}		

SUPERSETEQ

SUPERSETEQ 연산자는 첫 번째 피연산자가 두 번째 피연산자의 모든 원소를 포함하거나 서로 동일한 경우, 즉 두 번째 피연산자가 첫 번째 피연산자의 부분집합인 경우 **TRUE** (1)를 반환한다. 단, 피연산자가 모두 **LIST** 타입인 경우에는 **SUPERSETEQ** 연산을 지원하지 않는다.

```
collection_operand SUPERSETEQ collection_operand
```

```
--selecting rows when the first operand is a superset of the second
operand
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SUPERSETEQ {'country','state','city'};
```

id	name	address	zip_code
3	'Jones	'	{'city', 'country', 'state'}
{1, 2, 3, 4}	4	'Smith	'
	'street'}	{1, 2, 3, 4}	{'city', 'country', 'state',
5	'Kim	'	{'city', 'country', 'state',
'street'}	{1, 2, 3, 4}		{'city', 'country', 'state',
6	'Smith	'	{'city', 'country', 'state',

```
'street'} {1, 2, 3, 5}
          7 'Brown'      {'city', 'country', 'state',
'street'} {}
```

```
--SUPERSETEQ operator cannot be used for comparison between LIST and
LIST type values
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUPERSETEQ {1,2,3};
```

```
ERROR: ' superseteq ' operator is not defined on types sequence and
sequence.
```

```
--Comparing operands with a SUPERSETEQ operator after casting LIST
type as SET type
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUPERSETEQ (CAST ({1,2,3} AS SET));
```

id	name	address	zip_code
1	'Kim	'	{'country', 'state'} {1, 2, 3}
3	'Jones	'	{'city', 'country', 'state'}
4	'Smith	'	{'city', 'country', 'state',
	'street'}	{1, 2, 3, 4}	
5	'Kim	'	{'city', 'country', 'state',
	'street'}	{1, 2, 3, 4}	
6	'Smith	'	{'city', 'country', 'state',
	'street'}	{1, 2, 3, 5}	

SUBSET

SUBSET 연산자는 두 번째 피연산자가 첫 번째 피연산자의 모든 원소를 포함하는 경우, 즉 첫 번째 피연산자가 두 번째 피연산자의 진부분집합인 경우 **TRUE** (1)을 반환한다. 피연산자 집합이 서로 동일한 경우에는 **FALSE** (0)을 반환한다. 단, 피연산자가 모두 **LIST** 타입인 경우에는 **SUBSET** 연산을 지원하지 않는다.

```
collection_operand SUBSET collection_operand
```

```
--selecting rows when the first operand is a subset of the second
operand and they are not same
```

```
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SUBSET {'country','state','city'};
```

```

      id  name                address                zip_code
=====
=====
      1  'Kim'                {'country', 'state'} {1, 2, 3}
      2  'Moy'                {'country', 'state'} {3, 2, 1}
```

```
--SUBSET operator cannot be used for comparison between LIST and
LIST type values
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUBSET {1,2,3};
```

```
ERROR: ' subset ' operator is not defined on types sequence and
sequence.
```

```
--Comparing operands with a SUBSET operator after casting LIST type
as SET type
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUBSET (CAST ({1,2,3} AS SET));
```

```

      id  name                address                zip_code
=====
=====
      7  'Brown'              {'city', 'country', 'state',
'street'} {}
```

SUBSETEQ

SUBSETEQ 연산자는 두 번째 피연산자가 첫 번째 피연산자의 모든 원소를 포함하거나 서로 동일한 경우, 즉 첫 번째 피연산자가 두 번째 피연산자의 부분집합인 경우 **TRUE** (1)을 반환한다. 단, 피연산자가 모두 **LIST** 타입인 경우에는 **SUBSETEQ** 연산을 지원하지 않는다.

```
collection_operand SUBSETEQ collection_operand
```

```
--selecting rows when the first operand is a subset of the second
operand
SELECT id, name, address, zip_code FROM contain_tbl WHERE address
SUBSETEQ {'country','state','city'};
```

```

      id  name                address                zip_code
=====
      1  'Kim      '          {'country', 'state'}  {1, 2, 3}
      2  'Moy      '          {'country', 'state'}  {3, 2, 1}
      3  'Jones    '          {'city', 'country', 'state'}
{1, 2, 3, 4}

```

```

--SUBSETEQ operator cannot be used for comparison between LIST and
LIST type values
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUBSETEQ {1,2,3};

```

```

ERROR: ' subseteq ' operator is not defined on types sequence and
sequence.

```

```

--Comparing operands with a SUBSETEQ operator after casting LIST
type as SET type
SELECT id, name, address, zip_code FROM contain_tbl WHERE zip_code
SUBSETEQ (CAST ({1,2,3} AS SET));

```

```

      id  name                address                zip_code
=====
      1  'Kim      '          {'country', 'state'}  {1, 2, 3}
      7  'Brown    '          {'city', 'country', 'state',
'street'} {}

```

비트 함수와 연산자

목차

비트 함수와 연산자

7

- 비트 연산자 10
- BIT_AND 365
- BIT_OR 366

● BIT_XOR	366
● BIT_COUNT	367

비트 연산자

비트 연산자(Bitwise operator)는 비트 단위로 연산을 수행하며 산술 연산식에서 이용될 수 있다. 피연산자로 정수 타입이 지정되며, **BIT** 타입은 지정될 수 없다. 연산 결과로 **BIGINT** 타입 정수(64비트 정수)를 반환한다. 이때, 하나 이상의 피연산자가 **NULL** 이면 **NULL** 을 반환한다.

아래는 CUBRID가 지원하는 비트 연산자의 종류에 관한 표이다.

비트 연산자

비트 연산자	설명	조건식	리턴 값
&	비트 단위로 AND 연산을 수행하고, BIGINT 정수를 반환한다.	17 & 3	1
	비트 단위로 OR 연산을 수행하고, BIGINT 정수를 반환한다.	17 3	19
^	비트 단위로 XOR 연산을 수행하고, BIGINT 정수를 반환한다.	17 ^ 3	18
~	단항 연산자이며, 피연산자의 비트를 역으로 전환(INVERT)하는 보수 연산을 수행하고, BIGINT 정수를 반환한다.	~17	-18
<<	왼쪽 피연산자의 비트를 오른쪽 피연산자만큼 왼쪽으로 이	17 << 3	136

비트 연산자	설명	조건식	리턴 값
	동시키는 연산을 수행하고, BIGINT 정수를 반환한다.		
>>	왼쪽 피연산자의 비트를 오른쪽 피연산자만큼 오른쪽으로 이동시키는 연산을 수행하고, BIGINT 정수를 반환한다.	17 >> 3	2

BIT_AND

BIT_AND(*expr*)

집계 함수로서, *expr* 의 모든 비트에 대해 비트 단위 **AND** 연산을 수행한다. 조건절을 만족하는 행이 없는 경우, NULL 을 반환한다.

매개변수:

expr – 정수 타입의 임의의 연산식이다.

반환 형식:

BIGINT

```
CREATE TABLE bit_tbl(id int);
INSERT INTO bit_tbl VALUES (1), (2), (3), (4), (5);
SELECT 18385, BIT_AND(id) FROM bit_tbl WHERE id in(1,3,5);
```

```
18385          bit_and(id)
=====
1              1
```

BIT_OR

BIT_OR(*expr*)

집계 함수로서, *expr* 의 모든 비트에 대해 비트 단위 **OR** 연산을 수행한다. 조건절을 만족하는 행이 없는 경우, **NULL** 을 반환한다.

매개변수:

expr – 정수 타입의 임의의 연산식이다.

반환 형식:

BIGINT

```
SELECT 1|3|5, BIT_OR(id) FROM bit_tbl WHERE id in(1,3,5);
```

1 3 5	bit_or(id)
=====	=====
7	7

BIT_XOR

BIT_XOR(*expr*)

집계 함수로서, *expr* 의 모든 비트에 대해 비트 단위 **XOR** 연산을 수행한다. 조건절을 만족하는 행이 없는 경우, **NULL** 을 반환한다.

매개변수:

expr – 정수 타입의 임의의 연산식이다.

반환 형식:

BIGINT

```
SELECT 1^3^5, BIT_XOR(id) FROM bit_tbl WHERE id in(1,3,5);
```

```

1^3^5          bit_xor(id)
=====
7              7

```

BIT_COUNT

BIT_COUNT(*expr*)

*expr*의 모든 비트 중 1로 설정된 비트의 개수를 반환하는 함수이며, 집계 함수는 아니다.

매개변수:

expr – 정수 타입의 임의의 연산식이다.

반환 형식:

BIGINT

```
SELECT BIT_COUNT(id) FROM bit_tbl WHERE id in(1,3,5);
```

```

bit_count(id)
=====
1
2
2

```

문자열 함수와 연산자

목차

문자열 함수와 연산자

7

- 병합 연산자 10
- ASCII 371
- BIN 371

● BIT_LENGTH	372
● CHAR_LENGTH, CHARACTER_LENGTH, LENGTHB, LENGTH	374
● CHR	376
● COLLECTION_TO_STRING	377
● CONCAT	378
● CONCAT_WS	379
● ELT	380
● FIELD	381
● FIND_IN_SET	382
● FROM_BASE64	383
● INSERT	384
● INSTR	385
● LCASE, LOWER	387
● LEFT	389
● LOCATE	390
● LPAD	391
● LTRIM	393
● MID	394
● OCTET_LENGTH	396
● POSITION	398
● REPEAT	399

● REPLACE	400
● REVERSE	402
● RIGHT	402
● RPAD	403
● RTRIM	405
● SPACE	406
● STRCMP	408
● SUBSTR	409
● SUBSTRING	411
● SUBSTRING_INDEX	412
● TO_BASE64	414
● TRANSLATE	415
● TRIM	416
● UCASE, UPPER	418

참고

문자열 함수에서 **oracle_style_empty_string** 파라미터의 설정 값이 yes이면, 빈 문자열('')과 NULL을 구분하지 않고 함수에 따라 모두 NULL로 취급하거나 모두 빈 문자열로 취급한다. 이와 관련한 자세한 설명은 [oracle_style_empty_string](#)을 참고한다.

병합 연산자

병합 연산자는 피연산자로 문자열 또는 비트열 데이터 타입이 지정되며, 병합(concatenation)된 문자열 또는 비트열을 반환한다. 문자열 데이터의 병합 연산자로 덧셈 기호(+)와 두 개의 파이프 기호(II)가 제공된다. 피연산자로 **NULL** 이 지정된 경우는 **NULL** 값이 반환된다.

SQL 구문 관련 파라미터인 **pipes_as_concat** 파라미터(기본값: yes)가 no이면 이중 파이프 기호(||)가 불리언(Boolean) OR 연산자로 해석되며 **plus_as_concat** 파라미터(기본값: yes)가 no이면 덧셈 기호가 + 연산자로 해석되므로, 이러한 경우 **CONCAT** 함수를 사용하여 문자열 또는 비트열을 병합하는 것이 좋다.

```
<concat_operand1> + <concat_operand2>
<concat_operand1> || <concat_operand2>

<concat_operand1> ::=
    bit string |
    character string
    NULL

<concat_operand2> ::=
    bit string |
    character string
    NULL
```

- <concat_operand1>: 병합 후 왼쪽에 위치할 문자열 또는 비트열이다.
- <concat_operand2>: 병합 후 오른쪽에 위치할 문자열 또는 비트열이다.

```
SELECT 'CUBRID' || ',' + '2008';
```

```
'CUBRID' || ',' + '2008'
=====
'CUBRID,2008'
```

```
SELECT 'cubrid' || ',' || B'0010' || B'0000' || B'0000' || B'1000';
```

```
'cubrid' || ',' || B'0010' || B'0000' || B'0000' || B'1000'
=====
'cubrid,2008'
```

```
SELECT ((EXTRACT(YEAR FROM SYS_TIMESTAMP)) || (EXTRACT(MONTH FROM
SYS_TIMESTAMP)));
```

```
(( extract(year from SYS_TIMESTAMP )) || ( extract(month from
SYS_TIMESTAMP )))
=====
'200812'
```

```
SELECT 'CUBRID' || ',' + NULL;
```

```
'CUBRID' || ',' + null
=====
NULL
```

ASCII

ASCII(*str*)

ASCII 함수는 인자로 지정된 문자열의 가장 좌측 문자에 대한 ASCII 코드 값을 숫자로 반환한다. 입력 문자열이 **NULL** 이면 **NULL** 을 반환한다. **ASCII** 함수는 1바이트 문자에 대해 동작한다. 숫자가 입력되면 문자열로 변환한 후 가장 왼쪽 문자의 ASCII 코드 값을 반환한다.

매개변수:

str – 입력 문자열

반환 형식:

STRING

```
SELECT ASCII('5');
```

```
53
```

```
SELECT ASCII('ab');
```

```
97
```

BIN

BIN(*n*)

BIN 함수는 **BIGINT** 타입의 숫자를 이진 문자열로 표현한다. 입력 인자가 **NULL** 이면 **NULL** 을 반환한다. **BIGINT**로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 **NULL**을 반환한다.

매개변수:

n – **BIGINT** 타입의 숫자

반환 형식:

STRING

```
SELECT BIN(12);
```

```
'1100'
```

BIT_LENGTH

BIT_LENGTH(*string*)

BIT_LENGTH 함수는 문자열 또는 비트열의 길이(bit)를 정수값으로 반환한다. 단, 문자열의 경우 데이터 입력 환경의 문자셋(character set)에 따라 한 문자가 차지하는 바이트 수가 다르므로, **BIT_LENGTH** 함수의 리턴 값 역시 문자셋에 따라 다를 수 있다(예: UTF-8 한글: 한 글자에 3*8비트). CUBRID가 지원하는 문자셋에 관한 상세한 설명은 [문자열 데이터 타입](#) 을 참고한다. 유효하지 않은 값을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

string – 비트 단위로 길이를 구할 문자열 또는 비트열을 지정한다. **NULL** 이 지정된 경우는 **NULL** 값이 반환된다.

반환 형식:

INT

```
SELECT BIT_LENGTH('');
```

```
bit_length('')
=====
0
```



```
SELECT BIT_LENGTH('CUBRID');
```

```
bit_length('CUBRID')
=====
48
```

```
-- UTF-8 Korean character
SELECT BIT_LENGTH('큐브리드');
```

```
bit_length('큐브리드')
=====
96
```

```
SELECT BIT_LENGTH(B'010101010');
```

```
bit_length(B'010101010')
=====
9
```

```
CREATE TABLE bit_length_tbl (char_1 CHAR, char_2 CHAR(5), varchar_1
VARCHAR, bit_var_1 BIT VARYING);
INSERT INTO bit_length_tbl VALUES(' ', ' ', ' ', B' '); --Length of
empty string
INSERT INTO bit_length_tbl VALUES('a', 'a', 'a', B'010101010'); --
English character
INSERT INTO bit_length_tbl VALUES(NULL, '큐', '큐', B'010101010'); --
UTF-8 Korean character and NULL
INSERT INTO bit_length_tbl VALUES(' ', ' 큐', ' 큐', B'010101010'); --
UTF-8 Korean character and space

SELECT BIT_LENGTH(char_1), BIT_LENGTH(char_2),
BIT_LENGTH(varchar_1), BIT_LENGTH(bit_var_1) FROM bit_length_tbl;
```

```
bit_length(char_1)  bit_length(char_2)      bit_length(varchar_1)
bit_length(bit_var_1)
=====
=====
8                  40                      0
0
8                  40                      8
9
```

NULL	56	24
9		
8	40	32
9		

CHAR_LENGTH, CHARACTER_LENGTH, LENGTHB, LENGTH

CHAR_LENGTH(*string*)

CHARACTER_LENGTH(*string*)

LENGTHB(*string*)

LENGTH(*string*)

문자의 개수를 정수 값으로 반환한다. CUBRID가 지원하는 문자셋에 관한 상세한 설명은 [다국어 개요](#)를 참고한다. **CHAR_LENGTH**, **CHARACTER_LENGTH**, **LENGTHB**, **LENGTH** 함수는 동일하다.

매개변수:

string – 문자 개수 단위로 길이를 구할 문자열을 지정한다.

NULL 이 지정된 경우는 **NULL** 값이 반환된다.

반환 형식:

INT

참고

- CUBRID 9.0 미만 버전에서 멀티바이트 문자열의 경우 문자열의 바이트 수를 반환한다. 즉, 문자셋에 따라 문자 한 개당 길이가 2바이트 또는 3바이트로 계산된다.
- 문자열 내에 포함된 공백 문자(space)의 길이는 1바이트이다.
- 공백 문자를 표현하기 위한 빈 따옴표("")의 길이는 0이다. 단, **CHAR** (*n*) 타입에서는 공백 문자의 길이가 *n* 이고, *n* 이 생략되는 경우 1로 처리되므로 주의한다.

```
--character set is UTF-8 for Korean characters
SELECT LENGTH('');
```

```
char length('')
=====
0
```

```
SELECT LENGTH('CUBRID');
```

```
char length('CUBRID')
=====
6
```

```
SELECT LENGTH('큐브리드');
```

```
char length('큐브리드')
=====
4
```

```
CREATE TABLE length_tbl (char_1 CHAR, char_2 CHAR(5), varchar_1
VARCHAR, varchar_2 VARCHAR);
INSERT INTO length_tbl VALUES(' ', ' ', ' ', ' '); --Length of empty
string
INSERT INTO length_tbl VALUES('a', 'a', 'a', 'a'); --English
character
INSERT INTO length_tbl VALUES(NULL, '큐', '큐', '큐'); --Korean
character and NULL
INSERT INTO length_tbl VALUES(' ', ' 큐', ' 큐', ' 큐'); --Korean
character and space

SELECT LENGTH(char_1), LENGTH(char_2), LENGTH(varchar_1),
LENGTH(varchar_2) FROM length_tbl;
```

```
char_length(char_1) char_length(char_2) char_length(varchar_1)
char_length(varchar_2)
=====
=====
1                5                0                0
1                5                1                1
NULL            5                1                1
1                5                2                2
```

CHR

CHR(*number_operand* [USING *charset_name*])

CHR 함수는 인자로 지정된 연산식의 리턴 값에 대응하는 문자를 반환하는 함수이다. 유효하지 않은 범위의 코드 값을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **number_operand** - 수치값을 반환하는 임의의 연산식을 지정한다.
- **charset_name** - 문자셋 이름. 지원하는 문자셋은 utf8과 iso88591이다.

반환 형식:

STRING

```
SELECT CHR(68) || CHR(68-2);
```

```
chr(68)|| chr(68-2)
=====
'DB'
```

CHR 함수를 사용해서 멀티바이트 문자를 반환하려면 해당 문자셋에 대해 유효한 범위의 숫자를 입력한다.

```
SELECT CHR(14909886 USING utf8);
-- Below query's result is the same as above.
SET NAMES utf8;
SELECT CHR(14909886);
```

```
chr(14909886 using utf8)
=====
'ま'
```

문자를 16진수 문자열로 반환하려면 **HEX** 함수를 사용한다.

```
SET NAMES utf8;
SELECT HEX('ま');
```

```
hex(_utf8'ま')
=====
'E381BE'
```

16진수 문자열을 10진수로 반환하려면 **CONV** 함수를 사용한다.

```
SET NAMES utf8;
SELECT CONV('E381BE',16,10);
```

```
conv(_utf8'E381BE', 16, 10)
=====
'14909886'
```

COLLECTION_TO_STRING

COLLECTION_TO_STRING(*collection*)

COLLECTION_TO_STRING 함수는 컬렉션 값을 문자열로 직렬화한다. 결과는 모든 요소를 쉼표로 구분하고 중괄호로 묶은 형태이며, 예를 들어 `{1, 2, 3}` 과 같다. 문자열 요소는 작은따옴표로 묶인다.

매개변수:

collection – SET, MULTiset, SEQUENCE 타입의 컬렉션 값.

반환 형식:

STRING

*collection*이 **NULL**이면 **NULL**을 반환한다. 빈 컬렉션은 빈 중괄호 쌍을 반환한다.

```
SELECT COLLECTION_TO_STRING({1, 2, 3});
```

```
collection_to_string({1, 2, 3})
=====
'{1, 2, 3}'
```

CONCAT

CONCAT(*string1*, *string2* [, *string3* [, ... [, *stringN*]...]])

CONCAT 함수는 두 개 이상의 인자가 지정되며, 모든 인자 값을 연결한 문자열을 결과로 반환한다. 지정 가능한 인자의 개수는 제한이 없으며, 문자열 타입이 아닌 인자가 지정되는 경우 자동으로 타입 변환이 수행된다. 인자 중에 **NULL** 이 포함되면 결과로 **NULL** 을 반환한다.

인자로 지정된 문자열 사이에 구분자(separator)를 삽입하여 연결하려면, **CONCAT_WS()** 함수를 사용한다.

매개변수:

strings – 연결할 문자열들

반환 형식:

STRING

```
SELECT CONCAT('CUBRID', '2008' , 'R3.0');
```

```
concat('CUBRID', '2008', 'R3.0')
=====
'CUBRID2008R3.0'
```

```
--it returns null when null is specified for one of parameters
SELECT CONCAT('CUBRID', '2008' , 'R3.0', NULL);
```

```
concat('CUBRID', '2008', 'R3.0', null)
=====
NULL
```

```
--it converts number types and then returns concatenated strings
SELECT CONCAT(2008, 3.0);
```

```
concat(2008, 3.0)
=====
'20083.0'
```

CONCAT_WS

CONCAT_WS(*string1*, *string2* [,*string3* [, ... [, *stringN*]...]])

CONCAT_WS 함수는 두 개 이상의 인자가 지정되며, 첫 번째 인자 값을 구분자로 이용하여 나머지 인자 값을 연결한 문자열을 결과로 반환한다. 지정 가능한 인자의 개수에는 제한이 없으며, 문자열 타입이 아닌 인자가 지정되는 경우 자동으로 타입 변환이 수행된다. 만약, 구분자로 **NULL** 이 지정되면 **NULL** 을 반환하고, 구분자 다음에 위치하는 나머지 인자에 **NULL** 이 지정되면 이를 무시하고 문자열을 반환한다.

매개변수:

strings – 연결할 문자열들

반환 형식:

STRING

```
SELECT CONCAT_WS(' ', 'CUBRID', '2008' , 'R3.0');
```

```
concat_ws(' ', 'CUBRID', '2008', 'R3.0')
=====
'CUBRID 2008 R3.0'
```

```
--it returns strings even if null is specified for one of parameters
SELECT CONCAT_WS(' ', 'CUBRID', '2008', NULL, 'R3.0');
```

```
concat_ws(' ', 'CUBRID', '2008', null, 'R3.0')
=====
'CUBRID 2008 R3.0'
```

```
--it converts number types and then returns concatenated strings
with separator
SELECT CONCAT_WS(' ',2008, 3.0);
```

```
concat_ws(' ', 2008, 3.0)
=====
'2008 3.0'
```

ELT

ELT(*N*, *string1*, *string2*, ...)

ELT 함수는 *N*이 1이면 *string1*을 반환하고, *N*이 2이면 *string2*를 반환한다. 리턴 값은 **VARCHAR** 타입이다. 조건식은 필요에 따라 늘릴 수 있다.

문자열의 최대 바이트 길이는 33,554,432이며 이를 초과하면 **NULL**을 반환한다.

*N*이 0 또는 음수이면 빈 문자열을 반환한다. *N*이 입력 문자열의 개수보다 크면 범위를 벗어나므로 **NULL**을 반환한다. *N*이 정수로 변환할 수 없는 타입이면 에러를 반환한다.

매개변수:

- **N** – 문자열 리스트 중 반환할 문자열의 위치
- **strings** – 문자열 리스트

반환 형식:

STRING

```
SELECT ELT(3,'string1','string2','string3');
```

```
elt(3, 'string1', 'string2', 'string3')
=====
'string3'
```

```
SELECT ELT('3','1/1/1','23:00:00','2001-03-04');
```

```
elt('3', '1/1/1', '23:00:00', '2001-03-04')
=====
'2001-03-04'
```

```
SELECT ELT(-1, 'string1','string2','string3');
```

```
elt(-1, 'string1','string2','string3')
=====
NULL
```



```
SELECT ELT(4,'string1','string2','string3');
```

```
elt(4, 'string1', 'string2', 'string3')
=====
NULL
```

```
SELECT ELT(3.2,'string1','string2','string3');
```

```
elt(3.2, 'string1', 'string2', 'string3')
=====
'string3'
```

```
SELECT ELT('a','string1','string2','string3');
```

```
ERROR: Cannot coerce 'a' to type bigint.
```

FIELD

FIELD(*search_string*, *string1* [,*string2* [, ... [, *stringN*...]]])

FIELD 함수는 *string1*, *string2* 등의 인자 중 *search_string*과 동일한 인자의 위치 인덱스 값(포지션)을 반환한다. *search_string*과 동일한 인자가 없으면 0을 반환한다. *search_string*이 **NULL**이면 다른 인자와 비교 연산을 수행할 수 없으므로 0을 반환한다.

FIELD 함수에서 지정된 모든 인자가 문자열 타입이면 문자열 비교 연산을 수행하고, 모두 수치 타입이면 수치 비교 연산을 수행한다. 어느 한 인자의 타입이 나머지와 다른 경우, 모든 인자를 첫 번째 인자의 타입으로 변환하여 비교 연산을 수행한다. 각 인자와의 비교 연산 도중 타입 변환에 실패하면 비교 연산의 결과를 **FALSE**로 간주하고, 나머지 연산을 계속 진행한다.

매개변수:

- **search_string** – 검색할 문자열 패턴
- **strings** – 검색되는 문자열들의 리스트

반환 형식:

INT

```
SELECT FIELD('abc', 'a', 'ab', 'abc', 'abcd', 'abcde');
```

```
field('abc', 'a', 'ab', 'abc', 'abcd', 'abcde')
=====
3
```

```
--it returns 0 when no same string is found in the list
SELECT FIELD('abc', 'a', 'ab', NULL);
```

```
field('abc', 'a', 'ab', null)
=====
0
```

```
--it returns 0 when null is specified in the first parameter
SELECT FIELD(NULL, 'a', 'ab', NULL);
```

```
field(null, 'a', 'ab', null)
=====
0
```

```
SELECT FIELD('123', 1, 12, 123.0, 1234, 12345);
```

```
field('123', 1, 12, 123.0, 1234, 12345)
=====
0
```

```
SELECT FIELD(123, 1, 12, '123.0', 1234, 12345);
```

```
field(123, 1, 12, '123.0', 1234, 12345)
=====
3
```

FIND_IN_SET

FIND_IN_SET(*str*, *strlist*)

FIND_IN_SET 함수는 여러 개의 문자열을 쉼표(,)로 연결하여 구성한 문자열 리스트 *strlist* 에서 특정 문자열 *str* 이 존재하면 *str* 의 위치를 반환한다. *strlist* 에 *str* 이 존재하지 않거나 *strlist* 가 빈 문자열이면 0을 반환한다. 둘 중 하나의 인자가 **NULL** 이면 **NULL** 을 반환한다. *str* 이 쉼표를 포함하면 제대로 동작하지 않는다.

매개변수:

- **str** – 검색 대상 문자열
- **strlist** – 쉼표로 구분한 문자열의 집합

반환 형식:

INT

```
SELECT FIND_IN_SET('b','a,b,c,d');
```

2

FROM_BASE64

FROM_BASE64(*str*)

FROM_BASE64 함수는 **TO_BASE64** 함수에서 사용되는 base-64 암호화 규칙으로 암호화된 문자열을 인자로 입력받아 복호화된 결과를 바이너리 문자열로 반환한다. 입력 인자가 **NULL** 이면 **NULL**을 반환한다. 유효하지 않은 base-64 문자열일 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 **NULL**을 반환한다. 암호화 규칙에 대한 상세 내용은 [TO_BASE64\(\)](#) 를 참고한다.

매개변수:

str – 입력 문자열

반환 형식:

STRING

```
SELECT TO_BASE64('abcd'), FROM_BASE64(TO_BASE64('abcd'));
```

```
to_base64('abcd') from_base64( to_base64('abcd'))
=====
'YWJjZA==' 'abcd'
```

! 더 보기

TO_BASE64()

INSERT

INSERT(*str*, *pos*, *len*, *string*)

INSERT 함수는 입력 문자열의 특정 위치부터 정해진 길이만큼 부분 문자열을 삽입한다. 리턴 값은 **VARCHAR** 타입이다. 문자열의 최대 길이는 33,554,432이며 이를 초과하면 **NULL** 을 반환한다.

매개변수:

- **str** – 입력 문자열
- **pos** – *str* 의 위치. 1부터 시작한다. *pos* 가 1보다 작거나 *string* 의 길이+1보다 크면, *string* 을 삽입하지 않고 *str* 을 리턴한다.
- **len** – *str* 의 *pos* 에 삽입할 *string* 의 길이. *len* 이 부분 문자열의 길이를 초과하면, *str* 의 *pos* 에서 *string* 만큼 삽입한다. *len* 이 음수이면 *str* 이 문자열의 끝이 된다.
- **string** – *str* 에 삽입할 부분 문자열

반환 형식:

STRING

```
SELECT INSERT('cubrid',2,2,'dbsql');
```

```
insert('cubrid', 2, 2, 'dbsql')
=====
'cdbsqlrid'
```

```
SELECT INSERT('cubrid',0,3,'db');
```

```
insert('cubrid', 0, 3, 'db')
=====
'cubrid'
```

```
SELECT INSERT('cubrid',-3,3,'db');
```

```
insert('cubrid', -3, 3, 'db')
=====
'cubrid'
```

```
SELECT INSERT('cubrid',3,100,'db');
```

```
insert('cubrid', 3, 100, 'db')
=====
'cudb'
```

```
SELECT INSERT('cubrid',7,100,'db');
```

```
insert('cubrid', 7, 100, 'db')
=====
'cubriddb'
```

```
SELECT INSERT('cubrid',3,-1,'db');
```

```
insert('cubrid', 3, -1, 'db')
=====
'cudb'
```

INSTR

INSTR(*string*, *substring*[, *position*])

INSTR 함수는 **POSITION** 함수와 유사하게 문자열 *string* 내에서 문자열 *substring* 의 위치를 반환한다. 단, **INSTR** 함수는 *substring* 의 검색을 시작할 위치를 지정할 수 있으므로 중복된 *substring* 을 검색할 수 있다.

매개변수:

- **string** – 입력 문자열을 지정한다.
- **substring** – 위치를 반환할 문자열을 지정한다.
- **position** – 선택 사항으로 탐색을 시작할 *string* 의 위치를 나타내며, 문자 개수 단위로 지정된다. 이 인자가 생략되면 기본값인 **1** 이 적용된다. *string* 의 첫 번째 위치는 1로 지정된다. 값이 음수이면 *string* 의 끝에서부터 지정된 값만큼 떨어진 위치에서 역방향으로 *string* 을 탐색한다.

반환 형식:

INT

참고

CUBRID 9.0 미만 버전에서는 문자 단위가 아닌 바이트 단위로 위치를 반환한다는 점을 주의한다. CUBRID 9.0 미만 버전에서 멀티바이트 문자셋이면 한 문자를 표현하는 바이트 수가 다르므로 반환되는 결과 값이 다를 수 있다.

```
--character set is UTF-8 for Korean characters
--it returns position of the first 'b'
SELECT INSTR ('12345abcdeabcde','b');
```

```
instr('12345abcdeabcde', 'b', 1)
=====
7
```

```
-- it returns position of the first '나' on UTF-8 Korean charset
SELECT INSTR ('12345가나다라마가나다라마', '나' );
```

```
instr('12345가나다라마가나다라마', '나', 1)
=====
7
```

```
-- it returns position of the second '나' on UTF-8 Korean charset
SELECT INSTR ('12345가나다라마가나다라마', '나', 11 );
```

```
instr('12345가나다라마가나다라마', '나', 11)
=====
12
```

```
--it returns position of the 'b' searching from the 8th position
SELECT INSTR ('12345abcdeabcde','b', 8);
```

```
instr('12345abcdeabcde', 'b', 8)
=====
12
```

```
--it returns position of the 'b' searching backwardly from the end
SELECT INSTR ('12345abcdeabcde','b', -1);
```

```
instr('12345abcdeabcde', 'b', -1)
=====
12
```

```
--it returns position of the 'b' searching backwardly from a
specified position
SELECT INSTR ('12345abcdeabcde','b', -8);
```

```
instr('12345abcdeabcde', 'b', -8)
=====
7
```

LCASE, LOWER

LCASE(*string*)

LOWER(*string*)

LCASE 함수와 **LOWER** 함수는 동일하며, 문자열에 포함된 대문자를 소문자로 변환한다.

매개변수:

string – 소문자로 변환할 문자열을 지정한다. 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.

반환 형식:

STRING

```
SELECT LOWER('');
```

```
lower('')
=====
''
```

```
SELECT LOWER(NULL);
```

```
lower(null)
=====
NULL
```

```
SELECT LOWER('Cubrid');
```

```
lower('Cubrid')
=====
'cubrid'
```

단, 콜레이션의 지정에 따라 정상 동작하지 않을 수 있으므로 주의한다. 예를 들어, 루마니아어에서 사용되는 문자 Ă를 소문자로 변환하고자 할 때 콜레이션에 따라 다음과 같이 동작한다.

콜레이션이 utf8_bin이면 이 문자는 변환되지 않는다.

```
SET NAMES utf8 COLLATE utf8_bin;
SELECT LOWER('Ă');

lower(_utf8'Ă')
```



```
=====
'Ä'
```

콜레이션이 utf8_ro_cs이면 'Ä'는 소문자로 변환이 가능하다.

```
SET NAMES utf8 COLLATE utf8_ro_cs;
SELECT LOWER('Ä');

      lower(_utf8'Ä' COLLATE utf8_ro_cs)
=====
      'ä'
```

CUBRID가 지원하는 콜레이션에 관한 상세한 설명은 [CUBRID 콜레이션](#)을 참고한다.

LEFT

LEFT(*string*, *length*)

LEFT 함수는 *string* 의 가장 왼쪽에서부터 *length* 개의 문자를 반환한다. 어느 하나의 인자가 **NULL** 인 경우 **NULL** 이 반환되고, *string* 길이보다 큰 값이나 음수가 *length* 로 지정되면 문자열 전체를 반환한다. 문자열의 가장 오른쪽에서부터 *length* 길이의 문자열을 추출하려면 **RIGHT()** 를 사용한다.

매개변수:

- **string** – 입력 문자열
- **length** – 반환할 문자열의 길이

반환 형식:

STRING

```
SELECT LEFT('CUBRID', 3);
```

```
left('CUBRID', 3)
=====
'CUB'
```

```
SELECT LEFT('CUBRID', 10);
```

```
left('CUBRID', 10)
=====
'CUBRID'
```

LOCATE

LOCATE(*substring*, *string* [, *position*])

LOCATE 함수는 문자열 *string* 내에서 문자열 *substring* 의 위치 인덱스 값을 반환한다. 세 번째 인자 *position* 은 생략할 수 있으며, 이 인자가 지정되면 해당 위치에서부터 *substring* 을 검색하여 처음 검색한 위치 인덱스 값을 반환한다. *substring* 이 *string* 내에서 검색되지 않으면 0을 반환한다. **LOCATE** 함수는 **POSITION()** 와 유사하게 동작하지만, 비트열에 대해서는 **LOCATE** 함수를 적용할 수 없다.

매개변수:

- **substring** – 검색 대상 문자열의 패턴
- **string** – 전체 문자열
- **position** – 검색 시작 위치

반환 형식:

INT

```
--it returns 1 when substring is empty space
SELECT LOCATE ('', '12345abcdeabcde');
```

```
locate('', '12345abcdeabcde')
=====
1
```

```
--it returns position of the first 'abc'
SELECT LOCATE ('abc', '12345abcdeabcde');
```

```
locate('abc', '12345abcdeabcde')
=====
6
```

```
--it returns position of the second 'abc'
SELECT LOCATE ('abc', '12345abcdeabcde', 8);
```

```
locate('abc', '12345abcdeabcde', 8)
=====
11
```

```
--it returns 0 when no substring found in the string
SELECT LOCATE ('ABC', '12345abcdeabcde');
```

```
locate('ABC', '12345abcdeabcde')
=====
0
```

LPAD

LPAD(*char1*, *n*[, *char2*])

LPAD 함수는 문자열이 일정 길이가 될 때까지 왼쪽에 특정 문자를 덧붙인다.

매개변수:

- **char1** – 덧붙이는 대상 문자열을 지정한다. *char1*의 길이보다 작은 *n*이 지정되면, 패딩을 수행하지 않고 *char1*을 길이 *n*으로 잘라내어 반환한다. 값이 **NULL**이면 결과는 **NULL**이 반환된다.
- **n** – *char1*의 전체 문자 개수를 지정한다. 값이 **NULL**이면 결과는 **NULL**이 반환된다.
- **char2** – *char1*의 길이가 *n*이 될 때까지 왼쪽에 덧붙일 문자열을 지정한다. 이를 지정하지 않으면 공백 문자(' ')가 *char2*의 기본값으로 사용된다. 값이 **NULL**이면 결과는 **NULL**이 반환된다.

반환 형식:

STRING

참고

CUBRID 9.0 미만 버전에서 멀티바이트 문자셋이면 한 문자를 2바이트 또는 3바이트로 처리하는데, n 값에 의해 한 문자를 표현하는 첫 번째 바이트까지 char1을 잘라내는 경우, 마지막 문자를 정상적으로 표현할 수 없으므로 마지막 바이트를 제거하고 왼쪽에 공백 문자 하나(1바이트)를 덧붙인다. 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.

```
--character set is UTF-8 for Korean characters
--it returns only 3 characters if not enough length is specified
SELECT LPAD ('CUBRID', 3, '?');
```

```
lpad('CUBRID', 3, '?')
=====
'CUB'

SELECT LPAD ('큐브리드', 3, '?');
```

```
lpad('큐브리드', 3, '?')
=====
'큐브리'
```

```
--padding spaces on the left till char_length is 10
SELECT LPAD ('CUBRID', 10);
```

```
lpad('CUBRID', 10)
=====
'      CUBRID'
```

```
--padding specific characters on the left till char_length is 10
SELECT LPAD ('CUBRID', 10, '?');
```

```
lpad('CUBRID', 10, '?')
=====
'????CUBRID'
```

```
--padding specific characters on the left till char_length is 10
SELECT LPAD ('큐브리드', 10, '?');
```

```
lpad('큐브리드', 10, '?')
=====
'?????큐브리드'
```

```
--padding 4 characters on the left
SELECT LPAD ('큐브리드', LENGTH('큐브리드')+4, '?');
```

```
lpad('큐브리드', char_length('큐브리드')+4, '?')
=====
'????큐브리드'
```

LTRIM

LTRIM(*string* [, *trim_string*])

LTRIM 함수는 문자열의 왼쪽(앞 부분)에 위치한 특정 문자를 제거한다.

매개변수:

- **string** – 트리밍할 문자열 또는 문자열 타입의 칼럼을 입력하며, 이 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.
- **trim_string** – *string* 의 왼쪽에서 제거하고자 하는 특정 문자열을 지정할 수 있으며, 이를 지정하지 않으면 공백 문자 (' ')가 자동으로 지정되어 대상 문자열의 왼쪽에 위치한 공백이 제거된다.

반환 형식:

STRING

```
--trimming spaces on the left
SELECT LTRIM ('    Olympic    ');
```

```
ltrim('    Olympic    ')
=====
'Olympic    '
```

```
--If NULL is specified, it returns NULL
SELECT LTRIM ('iiiiOlympiciiii', NULL);
```

```
ltrim('iiiiOlympiciiii', null)
=====
NULL
```

```
-- trimming specific strings on the left
SELECT LTRIM ('iiiiOlympiciiii', 'i');
```

```
ltrim('iiiiOlympiciiii', 'i')
=====
'Olympiciiii'
```

MID

MID(*string*, *position*, *substring_length*)

MID 함수는 문자열 *string* 내의 *position* 위치로부터 *substring_length* 길이의 문자열을 추출하여 반환한다. 만약, *position* 값으로 음수가 지정되면, 문자열의 끝에서부터 역방향으로 위치를 산정한다. *substring_length* 는 생략할 수 없으며, 음수가 지정되는 경우 이를 0으로 간주하여 공백 문자열을 반환한다.

MID 함수는 `SUBSTR()` 와 유사하게 동작하나, 비트열에 대해서는 적용할 수 없고, *substring_length* 인자를 생략할 수 없으며, *substring_length* 에 음수가 지정되면 공백 문자열을 반환한다는 차이점이 있다.

매개변수:

- **string** - 입력 문자열을 지정한다. 입력 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **position** - 문자열을 추출할 시작 위치를 지정한다. 첫 번째 문자의 위치는 1이며, 0으로 지정되더라도 1로 간주된다. 입력 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.

- **substring_length** - 추출할 문자열의 길이를 지정한다. 0 또는 음수가 지정되는 경우 공백 문자열이 반환되고, 입력 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.

반환 형식:

STRING

```
CREATE TABLE mid_tbl(a VARCHAR);
INSERT INTO mid_tbl VALUES('12345abcdeabcde');

--it returns empty string when substring_length is 0
SELECT MID(a, 6, 0), SUBSTR(a, 6, 0), SUBSTRING(a, 6, 0) FROM
mid_tbl;
```

mid(a, 6, 0)	substr(a, 6, 0)	substring(a from 6 for 0)
' '	' '	' '

```
--it returns 4-length substrings counting from the 6th position
SELECT MID(a, 6, 4), SUBSTR(a, 6, 4), SUBSTRING(a, 6, 4) FROM
mid_tbl;
```

mid(a, 6, 4)	substr(a, 6, 4)	substring(a from 6 for 4)
'abcd'	'abcd'	'abcd'

```
--it returns an empty string when substring_length < 0
SELECT MID(a, 6, -4), SUBSTR(a, 6, -4), SUBSTRING(a, 6, -4) FROM
mid_tbl;
```

mid(a, 6, -4)	substr(a, 6, -4)	substring(a from 6 for -4)
' '	NULL	'abcdeabcde'

```
--it returns 4-length substrings at 6th position counting backward
from the end
```

```
SELECT MID(a, -6, 4), SUBSTR(a, -6, 4), SUBSTRING(a, -6, 4) FROM
mid_tbl;
```

```
mid(a, -6, 4)      substr(a, -6, 4)      substring(a from -6
for 4)
=====
'eabc'            'eabc'                '1234'
```

OCTET_LENGTH

OCTET_LENGTH(string)

OCTET_LENGTH 함수는 문자열 또는 비트열의 바이트(byte) 길이를 정수로 반환한다. 따라서, 비트열의 길이가 8비트인 경우에는 1(byte)을 반환하지만, 9비트인 경우에는 2(byte)를 반환한다.

매개변수:

string - 바이트 단위로 길이를 구할 문자열 또는 비트열을 지정한다. **NULL** 이 지정된 경우는 **NULL** 값이 반환된다.

반환 형식:

INT

```
--character set is UTF-8 for Korean characters
```

```
SELECT OCTET_LENGTH('');
```

```
octet_length('')
=====
0
```

```
SELECT OCTET_LENGTH('CUBRID');
```

```
octet_length('CUBRID')
=====
6
```



```
SELECT OCTET_LENGTH('큐브리드');
```

```
octet_length('큐브리드')
=====
12
```

```
SELECT OCTET_LENGTH(B'010101010');
```

```
octet_length(B'010101010')
=====
2
```

```
CREATE TABLE octet_length_tbl (char_1 CHAR, char_2 CHAR(5),
varchar_1 VARCHAR, bit_var_1 BIT VARYING);
INSERT INTO octet_length_tbl VALUES(' ', ' ', ' ', B''); --Length of
empty string
INSERT INTO octet_length_tbl VALUES('a', 'a', 'a', B'010101010'); --
English character
INSERT INTO octet_length_tbl VALUES(NULL, '큐', '큐', B'010101010');
--Korean character and NULL
INSERT INTO octet_length_tbl VALUES(' ', ' 큐', ' 큐', B'010101010');
--Korean character and space

SELECT OCTET_LENGTH(char_1), OCTET_LENGTH(char_2),
OCTET_LENGTH(varchar_1), OCTET_LENGTH(bit_var_1) FROM
octet_length_tbl;
```

```
octet_length(char_1) octet_length(char_2) octet_length(varchar_1)
octet_length(bit_var_1)
=====
=====
1                5
0                0
1                5
1                2
NULL            7
3                2
1                7
4                2
```

POSITION

POSITION(*substring* IN *string*)

POSITION 함수는 문자열 *string* 내에서 문자열 *substring* 의 위치를 반환한다.

이 함수의 인자로 문자열 또는 비트열을 반환하는 임의의 연산식을 지정할 수 있으며, 리턴 값은 0 이상의 정수이다. 문자열에 대해서는 문자 개수 단위로 위치 값을 반환하고, 비트열에 대해서는 비트 단위로 위치 값을 반환한다.

POSITION 함수는 가끔 다른 함수와 연결되어서 사용된다. 예를 들어, 특정 문자열에서 일부 문자열을 추출하고 싶은 경우에 **POSITION** 함수의 결과를 **SUBSTRING** 함수의 입력으로 사용할 수 있다.

참고

CUBRID 9.0 미만 버전에서는 문자 단위가 아닌 바이트 단위로 위치를 반환한다는 점을 주의한다. 멀티바이트 문자셋에서는 한 문자를 표현하는 바이트 수가 다르므로 반환되는 결과 값이 다를 수 있다.

매개변수:

substring – 위치를 반환할 문자열을 지정한다. 값이 공백 문자열이면 1이 반환된다. **NULL** 이면 **NULL** 이 반환된다.

반환 형식:

INT

```
--character set is UTF-8 for Korean characters
--it returns 1 when substring is empty space
SELECT POSITION ('' IN '12345abcdeabcde');
```

```
position('' in '12345abcdeabcde')
=====
1
```

```
--it returns position of the first 'b'
SELECT POSITION ('b' IN '12345abcdeabcde');
```

```
position('b' in '12345abcdeabcde')
=====
7
```

```
-- it returns position of the first '나'
SELECT POSITION ('나' IN '12345가나다라마가나다라마');
```

```
position('나' in '12345가나다라마가나다라마')
=====
7
```

```
--it returns 0 when no substring found in the string
SELECT POSITION ('f' IN '12345abcdeabcde');
```

```
position('f' in '12345abcdeabcde')
=====
0
```

```
SELECT POSITION (B'1' IN B'000011110000');
```

```
position(B'1' in B'000011110000')
=====
5
```

REPEAT

REPEAT(*string*, *count*)

REPEAT 함수는 입력 문자열에 대해 반복 횟수만큼의 문자열을 반환한다. 리턴 값은 **VARCHAR** 타입이다. 문자열의 최대 길이는 33,554,432이며, 이를 초과하면 **NULL** 을 반환한다. 입력 인자 중 하나가 **NULL** 이면 **NULL** 을 반환한다.

매개변수:

- **substring** – 문자열
- **count** – 반복 횟수. 0 또는 음수를 입력하면 빈 문자열을 반환하고, 숫자가 아닌 다른 데이터 타입을 입력하면 에러를 반환한다.

반환 형식:

STRING

```
SELECT REPEAT('cubrid',3);
```

```
repeat('cubrid', 3)
=====
'cubridcubridcubrid'
```

```
SELECT REPEAT('cubrid',32000000);
```

```
repeat('cubrid', 32000000)
=====
NULL
```

```
SELECT REPEAT('cubrid',-1);
```

```
repeat('cubrid', -1)
=====
''
```

```
SELECT REPEAT('cubrid','a');
```

```
ERROR: Cannot coerce 'a' to type integer.
```

REPLACE

`REPLACE(string, search_string[, replacement_string])`

REPLACE 함수는 주어진 문자열 *string* 내에서 문자열 *search_string* 을 검색하여 이를 문자열 *replacement_string* 으로 대체한다. 이때, 대체할 문자열 *replacement_string* 이 생략되면 *string* 내에서 검색된 *search_string* 이 모두 제거된다. 만약, 인자에 **NULL** 이 지정되면, **NULL** 이 반환된다.

매개변수:

- **string** – 원본 문자열을 지정한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **search_string** – 검색할 문자열을 지정한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **replacement_string** – *search_string* 을 대체할 문자열을 지정한다. 값이 생략되면 *string* 에서 *search_string* 을 제거하여 반환한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.

반환 형식:

STRING

```
--it returns NULL when an argument is specified with NULL value
SELECT REPLACE('12345abcdeabcde','abcde',NULL);
```

```
replace('12345abcdeabcde', 'abcde', null)
=====
NULL
```

```
--not only the first substring but all substrings into 'ABCDE' are
replaced
SELECT REPLACE('12345abcdeabcde','abcde','ABCDE');
```

```
replace('12345abcdeabcde', 'abcde', 'ABCDE')
=====
'12345ABCDEABCDE'
```

```
--it removes all of substrings when replace_string is omitted
SELECT REPLACE('12345abcdeabcde','abcde');
```

```
replace('12345abcdeabcde', 'abcde')
=====
'12345'
```

다음은 개행 문자(newline)를 “\n”으로 출력하도록 하는 예이다.

```
-- no_backslash_escapes=yes (default)

CREATE TABLE tbl (cmt_no INT PRIMARY KEY, cmt VARCHAR(1024));
INSERT INTO tbl VALUES (1234,
'This is a test for
new line.');
```

```
SELECT REPLACE(cmt, CHR(10), '\n')
FROM tbl
WHERE cmt_no=1234;
```

```
This is a test for\n\n new line.
```

REVERSE

REVERSE(*string*)

REVERSE 함수는 문자열 *string*을 역순으로 변환한 후 반환한다.

매개변수:

string - 입력 문자열을 지정한다. 입력 값이 공백 문자열이면 공백 문자열을 반환하고, **NULL** 이면 **NULL** 을 반환한다.

반환 형식:

STRING

```
SELECT REVERSE('CUBRID');
```

```
reverse('CUBRID')
=====
'DIRBUC'
```

RIGHT

RIGHT(*string*, *length*)

RIGHT 함수는 *string* 의 가장 오른쪽에서부터 *length* 개의 문자를 반환한다. 어느 하나의 인자가 **NULL** 인 경우 **NULL** 이 반환되고, *string* 길이보다 큰 값이나 음수가 *length* 로 지정되면 문

자열 전체를 반환한다. 문자열의 가장 왼쪽에서부터 *length* 길이의 문자열을 추출하려면 `LEFT()` 를 사용한다.

매개변수:

- **string** – 입력 문자열
- **length** – 반환할 문자열의 길이

반환 형식:

STRING

```
SELECT RIGHT('CUBRID', 3);
```

```
right('CUBRID', 3)
=====
'RID'
```

```
SELECT RIGHT ('CUBRID', 10);
```

```
right('CUBRID', 10)
=====
'CUBRID'
```

RPAD

RPAD(*char1*, *n*[, *char2*])

RPAD 함수는 문자열이 일정 길이가 될 때까지 오른쪽에 특정 문자를 덧붙인다.

매개변수:

- **char1** – 덧붙이는 대상 문자열을 지정한다. *char1* 의 길이보다 작은 *n* 이 지정되면, 패딩을 수행하지 않고 *char1* 을 길이 *n* 으로 잘라내어 반환한다. 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.
- **n** – *char1* 의 전체 길이를 지정한다. 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.

- **char2** – *char1*의 길이가 *n*이 될 때까지 오른쪽에 덧붙일 문자열을 지정한다. 이를 지정하지 않으면 공백 문자(' ')가 *char2*의 기본값으로 사용된다. 값이 **NULL**이면 결과는 **NULL**이 반환된다.

반환 형식:

STRING

참고

CUBRID 9.0 미만 버전에서 멀티바이트 문자셋이면 한 문자를 2바이트 또는 3바이트로 처리하는데, *n* 값에 의해 한 문자를 표현하는 첫 번째 바이트까지 *char1*을 잘라내는 경우, 마지막 문자를 정상적으로 표현할 수 없으므로 마지막 바이트를 제거하고 오른쪽에 공백 문자 하나(1바이트)를 덧붙인다. 값이 **NULL**이면 결과는 **NULL**이 반환된다.

```
--character set is UTF-8 for Korean characters

--it returns only 3 characters if not enough length is specified
SELECT RPAD ('CUBRID', 3, '?');
```

```
rpad('CUBRID', 3, '?')
=====
'CUB'
```

```
--on multi-byte charset, it returns the first character only with a
right-padded space
SELECT RPAD ('큐브리드', 3, '?');
```

```
rpad('큐브리드', 3, '?')
=====
'큐브리'
```

```
--padding spaces on the right till char_length is 10
SELECT RPAD ('CUBRID', 10);
```



```

rpad('CUBRID', 10)
=====
'CUBRID      '

```

```

--padding specific characters on the right till char_length is 10
SELECT RPAD ('CUBRID', 10, '?');

```

```

rpad('CUBRID', 10, '?')
=====
'CUBRID????'

```

```

--padding specific characters on the right till char_length is 10
SELECT RPAD ('큐브리드', 10, '?');

```

```

rpad('큐브리드', 10, '?')
=====
'큐브리드?????'

```

```

--padding 4 characters on the right
SELECT RPAD ('큐브리드', LENGTH('큐브리드')+4, '?');

```

```

rpad('', char_length('')+4, '?')
=====
'큐브리드????'

```

RTRIM

RTRIM(*string* [, *trim_string*])

RTRIM 함수는 문자열의 오른쪽(뒷 부분)에 위치한 특정 문자를 제거한다.

매개변수:

- **string** – 트리밍할 문자열 또는 문자열 타입의 칼럼을 입력하며, 이 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.
- **trim_string** – *string* 의 오른쪽에서 제거하고자 하는 특정 문자열을 지정할 수 있으며, 이를 지정하지 않으면 공백 문자(' ')가 자동으로 지정되어 대상 문자열의 오른쪽에 위치한 공백이 제거된다.

반환 형식:

STRING

```
SELECT RTRIM ('      Olympic     ');
```

```
rtrim('      Olympic      ')
=====
'      Olympic'
```

```
--If NULL is specified, it returns NULL
SELECT RTRIM ('iiiiOlympiciiii', NULL);
```

```
rtrim('iiiiOlympiciiii', null)
=====
NULL
```

```
-- trimming specific strings on the right
SELECT RTRIM ('iiiiOlympiciiii', 'i');
```

```
rtrim('iiiiOlympiciiii', 'i')
=====
'iiiiOlympic'
```

SPACE

SPACE(*N*)

SPACE 함수는 지정한 숫자만큼의 공백 문자열을 반환한다. 리턴 값은 **VARCHAR** 타입이다.

매개변수:

N – 공백 개수. 시스템 파라미터 **string_max_size_bytes** 에 지정된 값보다 클 수 없으며(기본값 1048576), 이를 초과하면 **NULL** 을 반환한다. 최대값은 33,554,432이며 이를 초과하면 **NULL** 을 반환한다. 0 또는 음수를 입력하면 빈 문자열을 반환하고, 숫자로 변환할 수 없는 타입을 입력하면 에러를 반환한다.

반환 형식:

STRING

```
SELECT SPACE(8);
```

```
space(8)
=====
      '      '
```

```
SELECT LENGTH(space(1048576));
```

```
char_length( space(1048576))
=====
                        1048576
```

```
SELECT LENGTH(space(1048577));
```

```
char_length( space(1048577))
=====
                        NULL
```

```
-- string_max_size_bytes=33554432
SELECT LENGTH(space('33554432'));
```

```
char_length( space('33554432'))
=====
                        33554432
```

```
SELECT SPACE('aaa');
```

```
ERROR: Cannot coerce 'aaa' to type bigint.
```

STRCMP

STRCMP(*string1*, *string2*)

STRCMP 함수는 두 개의 문자열 *string1*, *string2* 을 비교하여 동일하면 0을 반환하고, *string1* 이 더 크면 1을 반환하고, *string1* 이 더 작은 경우에는 -1을 반환한다. 어느 하나의 인자가 **NULL** 이면 **NULL** 을 반환한다.

매개변수:

- **string1** – 비교 대상 문자열
- **string2** – 비교 대상 문자열

반환 형식:

INT

```
SELECT STRCMP('abc', 'abc');
```

```
0
```

```
SELECT STRCMP ('acc', 'abc');
```

```
1
```

참고

9.0 미만 버전까지는 STRCMP가 대소문자를 구분하지 않고 문자열을 비교했으나, 9.0 버전부터는 대소문자를 구분하여 문자열을 비교한다. 대소문자를 구분하지 않게 동작하려면 문자열에 대소문자를 구분하지 않는 콜레이션(예: utf8_en_ci)을 지정한다.

```
-- In previous version of 9.0 STRCMP works case-insensitively
SELECT STRCMP ('ABC', 'abc');
```

```
0
```

```
-- From 9.0 version, STRCMP distinguish the uppercase and the
lowercase when the collation is case-sensitive.
-- charset is en_US.iso88591
```

```
SELECT STRCMP ('ABC', 'abc');
```

```
-1
```

```
-- If the collation is case-insensitive, it does not distinguish
the uppercase and the lowercase.
-- charset is en_US.iso88591
```

```
SELECT STRCMP ('ABC' COLLATE utf8_en_ci , 'abc' COLLATE utf8_en_ci);
```

```
0
```

SUBSTR

SUBSTR(*string*, *position*[, *substring_length*])

SUBSTR 함수는 문자열 *string* 내의 *position* 위치로부터 *substring_length* 길이의 문자열을 추출하여 반환한다. 만약, *position* 값으로 음수가 지정되면, 문자열의 끝에서부터 역방향으로 위치를 산정한다. 또한, *substring_length* 가 생략되는 경우, 주어진 *position* 위치로부터 마지막 까지 문자열을 추출하여 반환한다.

참고

CUBRID 9.0 미만 버전에서는 문자 단위가 아닌 바이트 단위로 시작 위치와 문자열의 길이를 산정한다는 점에 주의한다. 따라서, 멀티바이트 문자셋에서는 한 문자를 표현하는 바이트 수를 고려하여 인자를 지정해야 한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 입력 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **position** – 문자열을 추출할 시작 위치를 지정한다. 첫 번째 문자의 위치는 1이며, 0으로 지정되더라도 1로 간주된다. string 길이보다 큰 값을 지정하거나 **NULL** 을 지정하면 결과로 **NULL** 이 반환된다.
- **substring_length** – 추출할 문자열의 길이를 지정한다. 이 인자가 생략되면 *position* 위치로부터 마지막까지 문자열을 추출한다. 이 인자의 값으로 **NULL** 이 지정될 수 없으며, 0 이 지정되는 경우 공백 문자열이 반환되고, 음수가 지정되는 경우 **NULL** 이 반환된다.

반환 형식:

STRING

```
--character set is UTF-8 for Korean characters
--it returns empty string when substring_length is 0
SELECT SUBSTR('12345abcdeabcde',6, 0);
```

```
substr('12345abcdeabcde', 6, 0)
=====
''
```

```
--it returns 4-length substrings counting from the position
SELECT SUBSTR('12345abcdeabcde', 6, 4), SUBSTR('12345abcdeabcde',
-6, 4);
```

```
substr('12345abcdeabcde', 6, 4)    substr('12345abcdeabcde', -6, 4)
=====
'abcd'                            'eabc'
```

```
--it returns substrings counting from the position to the end
SELECT SUBSTR('12345abcdeabcde', 6), SUBSTR('12345abcdeabcde', -6);
```

```
substr('12345abcdeabcde', 6)    substr('12345abcdeabcde', -6)
=====
'abcdeabcde'                  'eabcde'
```

```
-- it returns 4-length substrings counting from 11th position
SELECT SUBSTR ('12345가나다라마가나다라마', 11 , 4);
```

```
substr('12345가나다라마가나다라마', 11 , 4)
=====
'가나다라'
```

SUBSTRING

SUBSTRING (string, position [, substring_length]),

SUBSTRING(*string FROM position [FOR substring_length]*)

SUBSTRING 함수는 **SUBSTR** 함수와 유사하며, 문자열 *string* 내의 *position* 위치로부터 *substring_length* 길이의 문자열을 추출하여 반환한다. *position* 값에 음수가 지정되면, **SUBSTRING** 함수는 문자열의 처음으로 검색 위치를 산정하고, **SUBSTR** 함수는 문자열의 끝에서부터 역방향으로 위치를 산정한다. *substring_length* 값에 음수가 지정되면, **SUBSTRING** 함수는 해당 인자가 생략된 것으로 처리하지만, **SUBSTR** 함수는 **NULL** 을 반환한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 입력 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **position** – 문자열을 추출할 시작 위치를 지정한다. 0이나 음수가 지정되면, 첫 번째 문자의 위치인 1로 간주된다. *string* 길이보다 큰 값을 지정하면 공백 문자열이 반환되고, **NULL** 을 지정하면 **NULL** 이 반환된다.
- **substring_length** – 추출할 문자열의 길이를 지정한다. 이 인자가 생략되면 *position* 위치로부터 마지막까지 문자열을 추출한다. 이 인자의 값으로 **NULL** 이 지정될 수 없으며, 0 이 지정되는 경우 공백 문자열이 반환되고, 음수를 지정하면 무시한다.

반환 형식:

STRING

```
SELECT SUBSTRING('12345abcdeabcde', -6 ,4),
SUBSTR('12345abcdeabcde', -6 ,4);
```

```
substring('12345abcdeabcde' from -6 for 4)
substr('12345abcdeabcde', -6, 4)
=====
'1234'                'eabc'
```

```
SELECT SUBSTRING('12345abcdeabcde', 16), SUBSTR('12345abcdeabcde',
16);
```

```
substring('12345abcdeabcde' from 16)    substr('12345abcdeabcde',
16)
=====
' '                NULL
```

```
SELECT SUBSTRING('12345abcdeabcde', 6, -4),
SUBSTR('12345abcdeabcde', 6, -4);
```

```
substring('12345abcdeabcde' from 6 for -4)
substr('12345abcdeabcde', 6, -4)
=====
'abcdeabcde'      NULL
```

SUBSTRING_INDEX

SUBSTRING_INDEX(*string*, *delim*, *count*)

SUBSTRING_INDEX 함수는 문자열에 포함된 구분자를 세어 *count* 번째 구분자 앞까지의 부분 문자열을 반환한다. 리턴 값은 **VARCHAR** 타입이다.

매개변수:

- **string** – 입력 문자열. 최대 길이는 33,554,432이며, 이를 초과하면 **NULL** 을 반환한다.
- **delim** – 구분자. 대소문자를 구분한다.
- **count** – 구분자가 나타나는 횟수. 양수를 입력하면 문자열의 왼쪽부터 세고, 음수를 입력하면 오른쪽부터 센다. 0이

면 빈 문자열을 반환한다. 정수로 변환할 수 없는 타입을 입력하면 에러를 반환한다.

반환 형식:

STRING

```
SELECT SUBSTRING_INDEX('www.cubrid.org','.', '2');
```

```
substring_index('www.cubrid.org', '.', '2')
=====
'www.cubrid'
```

```
SELECT SUBSTRING_INDEX('www.cubrid.org','.', '2.3');
```

```
substring_index('www.cubrid.org', '.', '2.3')
=====
'www.cubrid'
```

```
SELECT SUBSTRING_INDEX('www.cubrid.org', ':', '2.3');
```

```
substring_index('www.cubrid.org', ':', '2.3')
=====
'www.cubrid.org'
```

```
SELECT SUBSTRING_INDEX('www.cubrid.org', 'cubrid', 1);
```

```
substring_index('www.cubrid.org', 'cubrid', 1)
=====
'www.'
```

```
SELECT SUBSTRING_INDEX('www.cubrid.org', '.', 100);
```

```
substring_index('www.cubrid.org', '.', 100)
=====
'www.cubrid.org'
```

TO_BASE64

TO_BASE64(str)

문자열을 base-64 암호화 형식으로 변환하여 결과를 반환한다. 입력 인자가 문자열이 아니면 변환이 발생하기 전에 문자열로 변환된다. 입력 인자가 **NULL**이면 **NULL**을 반환한다. Base-64로 암호화된 문자열은 `FROM_BASE64()` 함수로 복호화될 수 있다.

매개변수:

str – 입력 문자열

반환 형식:

STRING

```
SELECT TO_BASE64('abcd'), FROM_BASE64(TO_BASE64('abcd'));
```

```
to_base64('abcd') from_base64( to_base64('abcd'))
=====
'YWJjZA==' 'abcd'
```

다음은 `TO_BASE64()` 함수와 `FROM_BASE64()` 함수에서 사용되는 암호화 및 복호화 규칙이다.

- 알파벳 값 62에 대한 암호화는 '+'이다.
- 알파벳 값 63에 대한 암호화는 '/'이다.
- 암호화된 결과는 4개의 출력 가능한 문자 그룹으로 구성되어 있다. 입력 데이터의 세 바이트는 네 개의 문자로 암호화된다. 마지막 그룹이 네 개의 문자로 채워지지 않으면 '=' 문자를 덧붙여 (padding) 네 개 문자의 길이를 만든다.
- 긴 출력을 여러 개의 라인으로 나누기 위해 76개의 암호화된 출력 문자마다 뉴라인(newline)이 추가된다.
- 복호화는 뉴 라인(newline), 캐리지 리턴(carriage return), 탭, 공백 문자를 인식하고 이들을 무시한다.

! 더 보기

FROM_BASE64()

TRANSLATE

TRANSLATE(*string*, *from_substring*, *to_substring*)

TRANSLATE 함수는 지정된 문자열 *string* 내에 문자열 *from_substring* 에 지정된 문자가 존재한다면, 이를 *to_substring* 에 지정된 문자로 대체한다. 이때, *from_substring* 과 *to_substring* 에 지정되는 문자의 순서에 따라 대응 관계를 가지며, *to_substring* 과 1:1 대응되지 않는 나머지 *from_substring* 문자는 문자열 *string* 내에서 모두 제거된다. **REPLACE()** 함수와 유사하게 동작하나, **TRANSLATE** 함수에서는 *to_substring* 인자를 생략할 수 없다.

매개변수:

- **string** – 입력 문자열. 최대 길이는 33,554,432이며, 이를 초과하면 **NULL** 을 반환한다
- **from_substring** – 검색할 문자열을 지정한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **to_substring** – *from_substring* 에 지정된 문자열을 대체할 문자열을 지정하며, 생략할 수 없다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.

반환 형식:

STRING

```
--it returns NULL when an argument is specified with NULL value
SELECT TRANSLATE('12345abcdeabcde','abcde', NULL);
```

```
translate('12345abcdeabcde', 'abcde', null)
=====
NULL
```

```
--it translates 'a','b','c','d','e' into '1', '2', '3', '4', '5'
respectively
SELECT TRANSLATE('12345abcdeabcde', 'abcde', '12345');
```

```
translate('12345abcdeabcde', 'abcde', '12345')
=====
'123451234512345'
```

```
--it translates 'a','b','c' into '1', '2', '3' respectively and
removes 'd's and 'e's
SELECT TRANSLATE('12345abcdeabcde','abcde', '123');
```

```
translate('12345abcdeabcde', 'abcde', '123')
=====
'12345123123'
```

```
--it removes 'a's,'b's,'c's,'d's, and 'e's in the string
SELECT TRANSLATE('12345abcdeabcde','abcde', '');
```

```
translate('12345abcdeabcde', 'abcde', '')
=====
'12345'
```

```
--it only translates 'a','b','c' into '3', '4', '5' respectively
SELECT TRANSLATE('12345abcdeabcde','ABabc', '12345');
```

```
translate('12345abcdeabcde', 'ABabc', '12345')
=====
'12345345de345de'
```

TRIM

TRIM([[LEADING | TRAILING | BOTH] [trim_string] FROM] string)

TRIM 함수는 문자열의 앞, 뒤 또는 앞뒤에 위치한 특정 문자들을 제거한다.

매개변수:

- **trim_string** – 대상 문자열의 앞, 뒤 또는 앞뒤에서 제거하고자 하는 특정 문자열을 지정할 수 있으며, 이를 지정하지 않으면 공백 문자(' ')가 자동으로 지정되어 대상 문자열의 앞, 뒤 또는 앞뒤에 위치한 공백이 제거된다.

- **string** – 트리밍할 문자열 또는 문자열 타입의 칼럼을 입력하며, 이 값이 **NULL** 이면 **NULL** 이 반환된다.

반환 형식:

STRING

- **[LEADING|TRAILING|BOTH]** : 대상 문자열의 어느 위치에서 지정된 문자열을 트리밍할 것인지를 옵션으로 명시할 수 있다. **LEADING** 은 문자열의 앞 부분에서 트리밍을 수행하고, **TRAILING** 은 문자열의 뒷 부분에서 트리밍을 수행하며, **BOTH** 는 앞뒤에서 지정된 문자열을 트리밍한다. 옵션을 명시하지 않으면 기본값은 **BOTH** 이다.
- *trim_string* 과 *string* 의 문자열은 같은 문자셋을 가져야 한다.

```
--trimming NULL returns NULL
SELECT TRIM (NULL);
```

```
trim(both from null)
=====
NULL
```

```
--trimming spaces on both leading and trailing parts
SELECT TRIM ('    Olympic   ');
```

```
trim(both from '    Olympic    ')
=====
'Olympic'
```

```
--trimming specific strings on both leading and trailing parts
SELECT TRIM ('i' FROM 'iiiiOlympiciiii');
```

```
trim(both 'i' from 'iiiiOlympiciiii')
=====
'Olympic'
```

```
--trimming specific strings on the leading part
SELECT TRIM (LEADING 'i' FROM 'iiiiOlympiciiii');
```

```
trim(leading 'i' from 'iiiiOlympiciiii')
=====
'Olympiciiii'
```

```
--trimming specific strings on the trailing part
SELECT TRIM (TRAILING 'i' FROM 'iiiiOlympiciiii');
```

```
trim(trailing 'i' from 'iiiiOlympiciiii')
=====
'iiiiOlympic'
```

UCASE, UPPER

UCASE(*string*)

UPPER(*string*)

UCASE 함수와 **UPPER** 함수는 동일하며, 문자열에 포함된 소문자를 대문자로 변환한다.

매개변수:

string - 대문자로 변환할 문자열을 지정한다. 값이 **NULL** 이면 결과는 **NULL** 이 반환된다.

반환 형식:

STRING

```
SELECT UPPER('');
```

```
upper('')
=====
''
```

```
SELECT UPPER(NULL);
```

```
upper(null)
=====
NULL
```

```
SELECT UPPER('Cubrid');
```

```
upper('Cubrid')
=====
'CUBRID'
```

단, 콜레이션의 지정에 따라 정상 동작하지 않을 수 있으므로 주의한다. 예를 들어, 루마니아어에서 사용되는 문자 ă을 대문자로 변환하고자 할 때 콜레이션에 따라 다음과 같이 동작한다.

콜레이션이 utf8_bin이면 변환이 되지 않는다.

```
SET NAMES utf8 COLLATE utf8_bin;
SELECT UPPER('ă');

upper(_utf8'ă')
=====
'ă'
```

콜레이션이 utf8_ro_cs이면 변환이 가능하다.

```
SET NAMES utf8 COLLATE utf8_ro_cs;
SELECT UPPER('ă');

upper(_utf8'ă' COLLATE utf8_ro_cs)
=====
'Ă'
```

CUBRID가 지원하는 콜레이션에 관한 상세한 설명은 [CUBRID 콜레이션](#)을 참고한다.

정규 표현식 함수와 연산자

정규 표현식은 복잡한 검색 패턴을 표현하는 강력한 방법이다. 이 섹션에서는 정규 표현식을 이용한 패턴을 매칭하는 함수 및 연산자를 설명한다.

목차

정규 표현식 함수와 연산자 7

● 정규 표현식 문법	420
● 특수 패턴 문자	10
● 수량자	12
● 그룹	19
● 어서션	21
● 구분자	21
● 문자 클래스	64
● 개별 문자	159
● 범위	161
● POSIX 기반 문자 클래스	431
● 유니코드 문자 클래스	433
● REGEXP, RLIKE	445
● REGEXP_COUNT	446
● REGEXP_INSTR	448
● REGEXP_LIKE	449
● REGEXP_REPLACE	451
● REGEXP_SUBSTR	453

정규 표현식 문법

정규 표현식을 구현하기 위해 CUBRID는 Google RE2 라이브러리와 C++ <regex> 표준 라이브러리를 사용한다. **regexp_engine** 시스템 파라미터를 설정하여 정규 표현식 라이브러리를 선택할 수 있고 Google RE2 라이브러리를 기본으로 사용한다.

다음 하위 섹션에서 지원되는 정규식 문법을 예시와 함께 설명한다.

참고

호환성 고려사항

- CUBRID 11 이전 버전에서 CUBRID는 Henry Spencer의 정규식 구현을 사용하였다.
 - CUBRID 11.0 부터 CUBRID는 Henry Spencer 라이브러리를 제거하고 C++ <regex> 표준 라이브러리를 사용하여 정규식 함수와 연산자를 지원한다.
 - CUBRID 11.2 부터 CUBRID는 Google RE2 라이브러리를 추가로 지원한다. 시스템 파라미터를 설정하여 C++ <regex> 또는 Google RE2를 선택하여 사용할 수 있다.
1. Henry Spencer의 정규식 구현은 바이트 방식으로 작동한다. 따라서 REGEXP 및 RLIKE는 멀티바이트를 지원하지 않았다. 따라서 인자의 콜레이션을 고려하지 않고 ASCII 인코딩으로만 작동했다.
 2. Henry Spencer 라이브러리는 POSIX의 *collating sequence* (`[.character.]`) 표현식을 지원했지만 더 이상 지원하지 않는다. 또한 *character equivalents* (`[=word=]`) 문법도 지원하지 않는다. 이러한 문법을 가진 표현식이 주어지면 CUBRID는 에러를 반환한다.
 3. Henry Spencer 라이브러리는 점 연산자 (`.`)로 행 종결자를 매치한다. 그러나 C++ <regex>는 매치되지 않는다.
 4. 단어-시작 경계 와 단어-끝 경계 (각각 `[[:<:]]` 와 `[[:>:]]`) 문법을 지원하지 않는다. 대신, 단어 경계 표기 (`\b`) 를 사용할 수 있다.

경고

C++ <regex> 라이브러리 주의사항

입력 문자열이 길거나 정규 표현식 패턴이 복잡한 경우 C++ <regex> 라이브러리에서 과도한 재귀 호출이 발생할 수 있는 문제가 있다. 따라서 C++ <regex> 대신 Google RE2를 사용하는 것을 권장한다. C++ <regex>는 하위 호환성을 지원하기 위해 남겨두었으며 이후 버전에서 제거될 예정이다. (deprecated)

참고

멀티바이트 문자열 비교 고려사항

C++ <regex>는 시스템 로케일에 따라 C++ <locale>에 의해 멀티바이트 비교를 수행한다. 따라서 로케일에 민감한 기능을 사용하려면 해당 시스템 로케일이 설치되어 있어야 합니다.

특수 패턴 문자

특수 패턴 문자는 문자열로 표현하기 어려운 문자를 나타내거나 문자 범위 (예: [a-z] 범위) 를 나타내기 위해 정규식 패턴에 나타날 때 특별한 의미를 갖는 문자(또는 문자 시퀀스) 이다. 이러한 각 특수 패턴 문자는 단일 문자에 대해 매칭을 수행한다 (수량자 기호 제외).

문자	설명
.	행 종결자를 제외한 모든 문자 (LF, CR, LS, PS).
\t	수평 탭 문자 (\u0009 와 동일).
\n	개행(줄 바꿈) 문자 (\u000A 와 동일).
\v	세로 탭 문자 (\u000B 와 동일).
\f	폼 피드 문자 (\u000C 와 동일).
\r	캐리지 리턴 문자 (\u000D 와 동일)
\cletter	코드 단위 값이 <i>letter</i> 의 코드 단위 값을 32로 나눈 나머지와 같은 제어 코드 문자.
\xhh	코드 단위 값이 2개의 16진수 <i>hh</i> 에 해당하는 16진수 값을 갖는 문자
\uhhhh	코드 단위 값이 4자리 16진수 <i>hhhh</i> 에 해당하는 16진수 값을 갖는 문자 C++ <regex>만 지원

문자	설명
<code>\x{digits}</code>	<i>digits</i> 값에 해당하는 16진수 값을 갖는 문자 <i>Google RE2</i> 만 지원
<code>\0</code>	널 문자 (<code>\u0000</code> 와 동일). <i>C++ <regex></i> 만 지원
<code>\num</code>	여는 괄호가 <i>num</i> 번째인 부분 일치의 결과. 자세한 내용은 아래 그룹을 참조.
<code>\d</code>	10진수 문자 (<code>[[:digit:]]</code> 와 동일).
<code>\D</code>	10진수 문자가 아닌 모든 문자 (<code>[^[:digit:]]</code> 와 동일).
<code>\s</code>	공백 문자 (<code>[[:space:]]</code> 와 동일).
<code>\S</code>	공백 문자가 아닌 모든 문자 (<code>[^[:space:]]</code> 와 동일).
<code>\w</code>	영숫자 또는 밑줄 문자 (<code>[[:alnum:]]</code> 와 동일).
<code>\W</code>	영숫자 또는 밑줄 문자가 아닌 모든 문자 (<code>[^[:alnum:]]</code> 와 동일).
<code>\character</code>	정규 표현식 내에서 특별한 의미를 해석하지 않고 있는 그대로 <i>character</i> 문자 위의 특수 문자 시퀀스를 형성하는 문자를 제외한 모든 문자를 이스케이프 한다. 필요한 대상: <code>^\$ \ . * + ? () [] { } </code>
<code>[class]</code>	<i>class</i> 의 일부인 문자열. 아래의 POSIX 기반 문자 클래스 를 참조.

문자

설명

[*^class*]

*class*의 일부가 아닌 문자열. 아래의 **POSIX 기반 문자 클래스**를 참조.

\p{*class*}

*class*의 유니코드 범위를 가지는 문자열. 아래의 **유니코드 문자 클래스**를 참조.

```
-- .: 모든 문자와 일치
SELECT ('cubrid dbms' REGEXP '^c.*$');
```

```
('cubrid dbms' regexp '^c.*$')
=====
1
```

“\n”, “\t”, “\r” 및 “\\”와 같은 특수 문자를 매칭하려면 일부는 **no_backslash_escapes**(기본값: yes) 값을 **no**로 지정하여 백슬래시(\)로 이스케이프해야 한다. **no_backslash_escapes**에 대한 자세한 내용은 **특수 문자 이스케이프**를 참조한다.

```
-- \n : no_backslash_escapes=yes(기본값)인 경우 \n 특수 문자와 일치
SELECT ('new\nline' REGEXP 'new\\nline');
```

```
('new\nline' REGEXP 'new\\nline');
=====
1
```

```
-- \n : no_backslash_escapes=no일 때 \n 특수 문자와 일치
SELECT ('new\nline' REGEXP 'new
line');
```

```
('new
line' regexp 'new
line')
=====
0
```

수량자

수량자 문자 또는 특수 패턴 문자 뒤에 지정합니다. 정규표현식 매칭에서 캐릭터가 반복되는 횟수를 지정할 수 있다:

Characters	Description
*	선행하는 문자 또는 패턴이 0회 이상 일치
+	선행하는 문자 또는 패턴이 1회 이상 일치
?	선행하는 문자 또는 패턴이 선택사항 (0회 또는 1회 일치)
{num}	선행하는 문자 또는 패턴이 정확히 num회 일치
{num,}	선행하는 문자 또는 패턴이 정확히 num회 또는 그 이상 일치
{min,max}	선행하는 문자 또는 패턴이 최소 min회 일치하지만 max를 넘지 않음

```
-- a+ : 하나 이상의 a 문자 시퀀스와 일치. 대소문자 구분하지 않음.
SELECT ('Aaaapricot' REGEXP '^A+pricot');
```

```
('Aaaapricot' regexp '^A+pricot')
=====
1
```

```
-- a? : 0 또는 1개의 문자와 일치
SELECT ('Apricot' REGEXP '^Aa?pricot');
```

```
('Apricot' regexp '^Aa?pricot')
=====
1
```

```
SELECT ('Apricot' REGEXP '^Aa?pricot');
```

```
('Apricot' regexp '^Aa?pricot')
=====
1
```

```
SELECT ('Aapricot' REGEXP '^Aa?pricot');
```

```
('Aapricot' regexp '^Aa?pricot')
=====
0
```

```
-- (cub)* : 시퀀스 abc의 0개 이상의 인스턴스와 일치
SELECT ('cubcub' REGEXP '^ (cub)*$');
```

```
('cubcub' regexp '^ (cub)*$')
=====
1
```

기본적으로 이러한 모든 수량자는 가능한 한 조건을 충족하는 많은 문자를 사용하는 *greedy* 방식으로 수행된다. 그리고 이 동작은 한정자 뒤에 물음표(?)를 추가하여 *non-greedy*로 재정의할 수 있다.

```
-- (a+), (a+?) : greedy 또는 non-greedy로 수량자를 사용해 각각 매칭 수행.
SELECT REGEXP_SUBSTR ('aardvark', '(a+)'), REGEXP_SUBSTR
('aardvark', '(a+?');
```

```
regexp_substr('aardvark', '(a+)') regexp_substr('aardvark', '(a+?')
=====
'aa'                               'a'
```

그룹

그룹을 사용하면 단일 문자 대신 일련의 문자에 레이블 (예 \$1)을 적용할 수 있다. 두 종류의 그룹이 있다.

⚠ 경고

\$int를 사용한 역참조 문법은 C++ <regex> 에서만 지원한다.

문자	설명
(subpattern)	역참조를 생성하는 그룹
(?:subpattern)	역참조를 생성하지 않는 패시브 그룹

```
-- 캡처된 그룹은 $int로 참조할 수 있다
SELECT REGEXP_REPLACE ('hello cubrid', '([[:alnum:]]+)', '$1!');
```

```
regexp_replace('hello cubrid', '([[:alnum:]]+)', '$1!')
=====
'hello! cubrid!'
```

그룹이 역참조를 생성하면, 문자열에서 하위 패턴을 나타내는 문자가 하위 일치로 저장된다. 각 하위 일치 항목은 여는 괄호가 나타나는 순서에 따라 번호가 매겨진다 (첫 번째 하위 일치 항목은 1번, 두 번째 하위 일치 항목은 2번 등...). 이러한 하위 일치는 전체 하위 패턴이 다른 곳에 다시 나타나도록 지정하기 위해 정규식 자체에서 사용할 수 있는데 (특수 문자 목록의 int 참조), REGEXP_REPLACE 함수의 대체 문자열에서 사용할 수 있다.

```
-- 그룹 없이 regexp_substr를 수행한다. 다음은 완전히 일치하는 경우이다.
SELECT REGEXP_SUBSTR ('abckabcjabc', '[a-c]{3}k[a-c]{3}j[a-c]{3}');

-- ([a-c]{3}) 는 \1 역참조를 생성한다
SELECT REGEXP_SUBSTR ('abckabcjabc', '([a-c]{3})k\1j\1');
```

```
regexp_substr('abckabcjabc', '[a-c]{3}k[a-c]{3}j[a-c]{3}')
=====
'abckabcjabc'

regexp_substr('abckabcjabc', '([a-c]{3})k\1j\1')
=====
'abckabcjabc'
```

어서션

어서션은 문자열의 문자를 사용하지 않는 조건이다. 즉, 문자를 나타내지는 않지만 문자의 앞이나 뒤에 충족되어야 하는 조건이다.

문자	설명
^	문자열의 시작이나 행 종결자 다음
\$	문자열의 끝이나 행 종결자 이전

참고

다음의 문법은 C++ <regex> 에서만 제공한다.

문자	설명
\b	이전 문자는 단어 문자이고 다음 문자는 단어가 아닌 문자(또는 그 반대)
\B	이전 및 다음 문자는 모두 단어 문자이거나 둘 다 단어가 아닌 문자
(?=subpattern)	Positive lookahead. 이 문자 다음의 문자열은 하위 패턴과 일치해야 하지만 어떤 문자도 소비되지 않음
(?!subpattern)	Negative lookahead. 어서션 다음의 문자열은 하위 패턴과 일치하지 않아야 하지만 어떤 문자도 소비되지 않음

```
-- ^ : 문자열의 시작과 매칭
SELECT ('cubrid dbms' REGEXP '^cub');
```



```
('cubrid dbms' regexp '^cub')
=====
1
```

```
-- $ : 문자열의 끝과 매칭
SELECT ('this is cubrid dbms' REGEXP 'dbms$');
```

```
('this is cubrid dbms' regexp 'dbms$')
=====
1
```

```
-- (?=subpattern): positive lookahead
SELECT REGEXP_REPLACE ('cubrid dbms cubrid sql cubrid rdbms',
'cubrid(?= sql)', 'CUBRID');

-- (?!subpattern): negative lookahead
SELECT REGEXP_REPLACE ('cubrid dbms cubrid sql cubrid rdbms',
'cubrid(?! sql)', 'CUBRID');
```

```
regexp_replace('cubrid dbms cubrid sql cubrid rdbms', 'cubrid(?=
sql)', 'CUBRID')
=====
'cubrid dbms CUBRID sql cubrid rdbms'

regexp_replace('cubrid dbms cubrid sql cubrid rdbms', 'cubrid(?!
sql)', 'CUBRID')
=====
'CUBRID dbms cubrid sql CUBRID rdbms'
```

구분자

패턴에 여러 구분자를 포함할 수 있다:

문자	설명
	두 개의 대체 패턴 또는 하위 패턴을 분리

```
-- a|b : a 또는 b인 모든 문자열 매칭
SELECT ('a' REGEXP 'a|b');
SELECT ('d' REGEXP 'a|b');
```

```
('a' regexp 'a|b')
=====
1

('d' regexp 'a|b')
=====
0
```

정규식은 구분자 연산자(alternatives, |)로 간단히 구분하여 여러 대체 패턴을 포함할 수 있다. 정규식은 대체 패턴이 일치하는 경우 일치하는 즉시 매칭한다. 하위 패턴(그룹 또는 어서션)은 구분자 연산자를 사용하여 여러 다른 패턴을 구분할 수도 있다.

```
-- a|b|c : matches any character that is either a, b or c.
SELECT ('a' REGEXP 'a|b|c');
SELECT ('d' REGEXP 'a|b|c');
```

```
('a' regexp 'a|b|c')
=====
1

('d' regexp 'a|b|c')
=====
0
```

문자 클래스

문자 클래스 문법은 문자 하나 또는 대괄호 내의 문자 범주와 매칭한다.

개별 문자

지정된 모든 문자는 클래스의 일부로 간주된다 (\, [,] 제외).

```
-- [abc] : a, b 또는 c인 모든 문자와 일치.
SELECT ('a' REGEXP '[abc]');
SELECT ('d' REGEXP '[abc]');
```

```
('a' regexp '[abc]')
=====
```

```
1
('d' regexp '[abc]')
=====
0
```

범위

문자 범위를 나타내기 위해 유효한 두 문자 사이에 대시 문자(-)를 사용한다. 예를 들어 “[a-z]”는 모든 알파벳 문자와 일치하지만 “[0-9]”는 단일 숫자와 일치한다.

```
SELECT ('adf' REGEXP '[a-f]');
SELECT ('adf' REGEXP '[g-z]');
```

```
('adf' regexp '[a-f]')
=====
1

('adf' regexp '[g-z]')
=====
0
```

```
-- [0-9]+: 문자열의 숫자 시퀀스와 일치
SELECT REGEXP_SUBSTR ('aas200gjb', '[0-9]+');
```

```
regexp_substr('aas200gjb', '[0-9]+')
=====
'200'
```

```
SELECT ('strike' REGEXP '^[^a-dXYZ]+$');
```

```
('strike' regexp '^[^a-dXYZ]+$')
=====
1
```

POSIX 기반 문자 클래스

POSIX 기반 문자 클래스([:classname:])는 아래와 같이 문자의 범주를 정의한다.

참고

C++ <regex>에서는 유니코드 영역까지 매칭하지만 Google RE2에서는 ASCII 영역만 매칭한다.

클래스	설명
[[:alnum:]]	영숫자 문자
[[:alpha:]]	알파벳 문자
[[:blank:]]	빈 문자
[[:cntrl:]]	제어 문자
[[:digit:]]	십진수 문자
[[:graph:]]	표시할 수 있는 문자
[[:lower:]]	소문자
[[:print:]]	인쇄 가능한 문자
[[:punct:]]	구두점 문자
[[:space:]]	공백 문자
[[:upper:]]	대문자
[[:xdigit:]]	16진수 문자

[[:d:]], [[:w:]] 및 [[:s:]]는 ECMAScript 문법의 확장이다. C++ <regex>에서만 사용할 수 있다.

클래스	설명
[d:]	숫자 (0-9)
[w:]	단어
[s:]	공백

```
SELECT REGEXP_SUBSTR ('Samseong-ro 86-gil, Gangnam-gu, Seoul 06178',
'[[[:digit:]]{5}');
```

```
regexp_substr('Samseong-ro 86-gil, Gangnam-gu, Seoul 06178',
'[[[:digit:]]{5}')
=====
'06178'
```

```
-- ;set regexp_engine=cppstd
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_REPLACE ('가나다 가나 가나다라', '\b([[:alpha:]]{2})\b',
'#');
```

```
regexp_count('가나123abc가다abc가가' collate utf8_ko_cs, '[가-나]+'
collate utf8_ko_cs)
```

```
=====
=====
```

유니코드 문자 클래스

유니코드 문자 클래스(`\p{classname}`)는 Google RE2 에서만 제공한다. 아래의 표와 같이 클래스 이름을 지정할 수 있다. 예를 들어 한글 문자를 매칭하기 위해서는 `\p{Hangul}`을 사용한다.

유니코드 문자 클래스

Adlam

Ahom

Anatolian_Hieroglyphs

Arabic

Armenian

Avestan

Balinese

Bamum

Bassa_Vah

Batak

Bengali

Bhaiksuki

Bopomofo

Brahmi

Braille

유니코드 문자 클래스

Buginese

Buhid

Canadian_Aboriginal

Carian

Caucasian_Albanian

Chakma

Cham

Cherokee

Chorasmian

Common

Coptic

Cuneiform

Cypriot

Cypro_Minoan

Cyrillic

유니코드 문자 클래스

Deseret

Devanagari

Dives_Akuru

Dogra

Duployan

Egyptian_Hieroglyphs

Elbasan

Elymaic

Ethiopic

Georgian

Glagolitic

Gothic

Grantha

Greek

Gujarati

유니코드 문자 클래스

Gunjala_Gondi

Gurmukhi

Han

Hangul

Hanifi_Rohingya

Hanunoo

Hatran

Hebrew

Hiragana

Imperial_Aramaic

Inherited

Inscriptional_Pahlavi

Inscriptional_Parthian

Javanese

Kaithi

유니코드 문자 클래스

Kannada

Katakana

Kawi

Kayah_Li

Kharoshthi

Khitan_Small_Script

Khmer

Khojki

Khudawadi

Lao

Latin

Lepcha

Limbu

Linear_A

Linear_B

유니코드 문자 클래스

Lisu

Lycian

Lydian

Mahajani

Makasar

Malayalam

Mandaic

Manichaeen

Marchen

Masaram_Gondi

Medefaidrin

Meetei_Mayek

Mende_Kikakui

Meroitic_Cursive

Meroitic_Hieroglyphs

유니코드 문자 클래스

Miao

Modi

Mongolian

Mro

Multani

Myanmar

Nabataean

Nag_Mundari

Nandinagari

New_Tai_Lue

Newa

Nko

Nushu

Nyiakeng_Puachue_Hmong

Ogham

유니코드 문자 클래스

Ol_Chiki

Old_Hungarian

Old_Italic

Old_North_Arabian

Old_Permic

Old_Persian

Old_Sogdian

Old_South_Arabian

Old_Turkic

Old_Uyghur

Oriya

Osage

Osmanya

Pahawh_Hmong

Palmyrene

유니코드 문자 클래스

Pau_Cin_Hau

Phags_Pa

Phoenician

Psalter_Pahlavi

Rejang

Runic

Samaritan

Saurashtra

Sharada

Shavian

Siddham

SignWriting

Sinhala

Sogdian

Sora_Sompeng

유니코드 문자 클래스

Soyombo

Sundanese

Syloti_Nagri

Syriac

Tagalog

Tagbanwa

Tai_Le

Tai_Tham

Tai_Viet

Takri

Tamil

Tangsa

Tangut

Telugu

Thaana

유니코드 문자 클래스

Thai

Tibetan

Tifinagh

Tirhuta

Toto

Ugaritic

Vai

Vithkuqi

Wancho

Warang_Citi

Yezidi

Yi

Zanabazar_Square

```
-- ;set regexp_engine=re2
SELECT REGEXP_COUNT('가나 가나다라 마바사아 자차카타 파하', '\p{Hangul}+');
```



```
regexp_count(_utf8'가나 가나다라 마바사아 자차카타 파하' collate
utf8_ko_cs, _utf8'\p{Hangul}+' collate utf8_ko_cs)
=====
5
```

REGEXP, RLIKE

REGEXP, **RLIKE**는 동일한 의미로 사용되며, 정규 표현식을 이용한 패턴을 매칭한다. 아래의 문법에서, *expression*이 *pattern*과 일치하면 1을 반환한다. 그렇지 않으면 0을 반환한다. *expression* 또는 *pattern*이 **NULL**이면 **NULL**을 반환한다. 두 번째 구문은 두 구문 모두 **NOT**을 사용하는 세 번째 구문과 동일한 의미를 갖는다.

```
expression REGEXP | RLIKE [BINARY] pattern
expression NOT REGEXP | RLIKE pattern
NOT (expression REGEXP | RLIKE pattern)
```

- *expression* : 칼럼 또는 입력 표현식
- *pattern* : 정규 표현식에 사용될 패턴. 대소문자 구분 없음

REGEXP와 **LIKE**의 차이는 다음과 같다.

- **LIKE** 절은 입력값 전체가 패턴과 매칭되어야 성공한다.
- **REGEXP**는 입력값의 일부가 패턴과 매칭되면 성공한다. **REGEXP**에서 전체 값에 대한 패턴 매칭을 하려면, 패턴의 시작에는 “^”을, 끝에는 “\$”을 사용해야 한다.
- **LIKE** 절의 패턴은 대소문자를 구분하지만 **REGEXP**에서 정규 표현식의 패턴은 대소문자를 구분하지 않는다. 대소문자를 구분하려면 **REGEXP BINARY** 구문을 사용해야 한다.

```
-- [a-dX], [^a-dX] : a, b, c, d 또는 X인 모든 문자와 매칭됨.
SELECT ('aXbc' REGEXP '[a-dX]');
```

```
('aXbc' regexp '[a-dX]')
=====
1
```

```
-- SELECT 리스트에서 REGEXP를 사용하는 경우 괄호로 묶어야 한다.
-- 단, WHERE 절에서는 괄호가 필요하지 않다.
-- BINARY와 함께 사용하는 경우를 제외하고 대소문자를 구분하지 않는다.
SELECT name FROM public.athlete where name REGEXP '^[a-d]';
```

```
name
=====
'Dziouba Irina'
'Dzieciol Iwona'
'Dzamalutdinov Kamil'
'Crucq Maurits'
'Crosta Daniele'
'Bukovec Brigita'
'Bukic Perica'
'AbduLLayev Namik'
```

REGEXP_COUNT

REGEXP_COUNT(*string*, *pattern_string*[, *position*[, *match_type*]])

REGEXP_COUNT 함수는 주어진 문자열 *string* 내에서 정규식 패턴 *pattern_string*의 매칭 횟수를 반환한다. **NULL**이 인수로 지정된 경우 **NULL**을 반환한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다
- **pattern_string** – 매칭을 수행할 정규식 패턴 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **position** – 매칭을 수행할 *string*의 위치를 지정한다. 값을 생략하면 기본값 1이 적용된다. 값이 음수이거나 0이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다
- **match_type** – 함수의 매칭 옵션을 설정할 문자열을 지정한다. 값을 생략하면 기본값 'i'가 적용된다. 값이 'c' 또는 'I'가 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

INT

```
-- 인수가 NULL 값으로 지정되면 NULL을 반환한다
SELECT REGEXP_COUNT('ab123ab111a', '[a-d]+', NULL);
```

```
regexp_count('ab123ab111a', '[a-d]+', NULL)
=====
NULL
```

```
-- 빈 문자열 패턴은 어떤 문자열과도 일치하지 않는다
SELECT REGEXP_COUNT('ab123ab111a', '');
```

```
regexp_count('ab123ab111a', '')
=====
0
```

```
SELECT REGEXP_COUNT('ab123Ab111aAA', '[a-d]', 3);
```

```
regexp_count('ab123Ab111aAA', '[a-d]', 3)
=====
5
```

```
-- 대소문자 구분 안함 옵션('i')이 기본값이다
SELECT REGEXP_COUNT('ab123Ab111aAA', '[a-d]', 3, 'i');

-- 대소문자 구분 옵션('c')이 match_type으로 지정된 경우 A는 매칭되지 않는다.
SELECT REGEXP_COUNT('ab123Ab111aAA', '[a-d]', 3, 'c');
```

```
regexp_count('ab123Ab111aAA', '[a-d]', 3, 'i')
=====
5

regexp_count('ab123Ab111aAA', '[a-d]', 3, 'c')
=====
2
```

```
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_COUNT('가나123abc가다abc가가', '[가-나]+');
```

```
regexp_count('가나123abc가다abc가가' collate utf8_ko_cs, '[가-나]+'
collate utf8_ko_cs)
=====
=====
```

REGEXP_INSTR

REGEXP_INSTR(*string*, *pattern_string*[, *position*[, *occurrence*[, *return_option*[, *match_type*]]]])

REGEXP_INSTR 함수는 주어진 문자열 *string* 내에서 정규식 패턴 *pattern_string*을 검색하여 시작 위치 또는 끝 위치를 반환한다. **NULL**이 인수로 지정된 경우 **NULL**을 반환한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다
- **pattern_string** – 매칭을 수행할 정규식 패턴 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **position** – 매칭을 수행할 *string*의 위치를 지정한다. 값을 생략하면 기본값 1이 적용된다. 값이 음수이거나 0이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다
- **occurrence** – 몇 번째 매칭을 사용할 지 지정한다. 값을 생략하면 기본값 1을 적용한다. 값을 생략하면 기본값 1이 적용된다. 값이 음수이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **return_option** – 일치하는 문자열의 시작 또는 끝 지점 중 어떤 위치를 반환할지 여부를 지정한다. 만약 값이 0이면 일치하는 문자의 시작 위치를 반환한다. 값이 1이면 일치하는 문자의 끝 위치를 반환한다. 값을 생략하면 기본값 0이 적용된다. 값이 0 또는 1이 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **match_type** – 함수의 매칭 옵션을 설정할 문자열을 지정한다. 값을 생략하면 기본값 'i'가 적용된다. 값이 'c' 또는 'I'가 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

INT

```
-- 인수가 NULL 값으로 지정되면 NULL을 반환한다
SELECT REGEXP_INSTR('12345abcdeabcde', '[abc]', NULL);
```

```
regexp_instr('12345abcdeabcde', '[abc]', null)
=====
NULL
```

```
-- 빈 문자열 패턴은 어떤 문자열과도 일치하지 않는다
SELECT REGEXP_INSTR('12345abcdeabcde', '');
```

```
regexp_instr('12345abcdeabcde', '')
=====
0
```

```
-- 매칭되는 첫 번째 문자의 위치를 반환한다
SELECT REGEXP_INSTR('12354abc5', '[:alpha:]+', 1, 1, 0);
```

```
regexp_instr('12354abc5', '[:alpha:]+', 1, 1, 0);
=====
6
```

```
-- 매칭 후 일치하는 문자열의 위치를 반환한다
SELECT REGEXP_INSTR('12354abc5', '[:alpha:]+', 1, 1, 1);
```

```
regexp_instr('12354abc5', '[:alpha:]+', 1, 1, 1);
=====
7
```

```
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_INSTR('12345가나다라마가나다라마바', '[가-다]+');
```

```
regexp_instr('12345가나다라마가나다라마바' collate utf8_ko_cs, '[가-다]+'
collate utf8_ko_cs)
=====
=====
```

REGEXP_LIKE

REGEXP_LIKE(*string*, *pattern_string*[, *match_type*])

REGEXP_LIKE 함수는 주어진 문자열 *string* 내에서 정규식 패턴 *pattern_string*을 검색한다. 정규식 패턴이 *pattern_string*의 어느 곳에서나 일치하면 1이 반환된다. 그렇지 않으면 0이 반환된다. **NULL**이 인수로 지정된 경우 **NULL**을 반환한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다
- **pattern_string** – 매칭을 수행할 정규식 패턴 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **match_type** – 함수의 매칭 옵션을 설정할 문자열을 지정한다. 값을 생략하면 기본값 'i'가 적용된다. 값이 'c' 또는 'I'가 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

INT

```
SELECT REGEXP_LIKE('abbbbc', 'ab+c');
```

```
regexp_like('abbbbc', 'ab+c');
=====
1
```

```
-- 빈 문자열 패턴은 어떤 문자열과도 일치하지 않는다
SELECT REGEXP_LIKE('abbbbc', '');
```

```
regexp_like('abbbbc', '');
=====
0
```

```
SELECT REGEXP_LIKE('abbbbc', 'AB+C', 'c');
```

```
regexp_like('abbbbc', 'AB+C');
=====
0
```

```
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_LIKE('가나다', '가나?다');
SELECT REGEXP_LIKE('가나라다', '가나?다');
```

```
regexp_like('가나다' collate utf8_ko_cs, '가나?다' collate utf8_ko_cs)
=====
=====
1

regexp_like('가나라다' collate utf8_ko_cs, '가나?다' collate utf8_ko_cs)
=====
=====
0
```

REGEXP_REPLACE

REGEXP_REPLACE(*string*, *pattern_string*, *replacement_string*[, *position*[, *occurrence*[, *match_type*]]])

REGEXP_REPLACE 함수는 주어진 문자열 *string* 내에서 정규식 패턴 *pattern_string*을 검색하여 문자열 *replacement_string*으로 바꾼다. **NULL**이 인수로 지정된 경우 **NULL**을 반환한다.

매개변수:

- **string** – 입력 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다
- **pattern_string** – 매칭을 수행할 정규식 패턴 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **replacement_string** – *pattern_string**과 일치하는 문자열을 대체할 문자열을 지정한다. 값이 ****NULL***이면 **NULL**을 반환한다.
- **position** – 매칭을 수행할 *string*의 위치를 지정한다. 값을 생략하면 기본값 1이 적용된다. 값이 음수이거나 0이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다

- **occurrence** – 몇 번째 매칭을 사용할 지 지정한다. 값을 생략하면 기본값 1을 적용한다. 만약 값을 생략하면 기본값 0이 적용된다. 값이 음수이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **match_type** – 함수의 매칭 옵션을 설정할 문자열을 지정한다. 값을 생략하면 기본값 'i'가 적용된다. 값이 'c' 또는 'i'가 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

STRING

```
-- 인수가 NULL 값으로 지정되면 NULL을 반환한다
SELECT REGEXP_REPLACE('12345abcdeabcde','[a-d]',NULL);
```

```
regexp_replace('12345abcdeabcde','[a-d]', null)
=====
NULL
```

```
-- 빈 문자열 패턴은 어떤 문자열과도 일치하지 않는다
SELECT REGEXP_REPLACE('12345abcdeabcde','','','#');
```

```
regexp_replace('12345abcdeabcde',' ','#')
=====
'12345abcdeabcde'
```

```
SELECT REGEXP_REPLACE('12345abDEKBcde','[a-d]','#');
```

```
regexp_replace('12345abDEKBcde','[a-d]','#')
=====
'12345###EK###e'
```

```
-- 대소문자 구분 안함 옵션('i')이 기본값이다
SELECT REGEXP_REPLACE('12345abDEKBcde','[a-d]','#', 1, 0, 'i');
```

```
-- 대소문자 구분 옵션('c')이 match_type으로 지정된 경우 'B' 와 'D'는 매칭되
```


지 않는다.

```
SELECT REGEXP_REPLACE('12345abDEKBcde','[a-d]','#', 1, 0, 'c');
```

```
regexp_replace('12345abDEKBcde', '[a-d]', '#', 1, 0, 'i')
=====
'12345###EK###e'
```

```
regexp_replace('12345abDEKBcde', '[a-d]', '#', 1, 0, 'c')
=====
'12345##DEKB##e'
```

```
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_REPLACE('a1가b2나라라','[가-다]','#',6);
```

```
regexp_replace('a1가b2나라라' collate utf8_ko_cs, '[가-다]' collate
utf8_ko_cs, '#' collate utf8_ko_cs, 6)
=====
'a1가b2##라'
```

REGEXP_SUBSTR

REGEXP_SUBSTR(*string*, *pattern_string*[, *position*[, *occurrence*[, *match_type*]]])

REGEXP_SUBSTR 함수는 주어진 문자열 *string* 내에서 정규식 패턴 *pattern_string*에 일치하는 문자열을 추출한다. **NULL**이 인수로 지정된 경우 **NULL**을 반환한다.

매개변수:

- **string** - 입력 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다
- **pattern_string** - 매칭을 수행할 정규식 패턴 문자열을 지정한다. 값이 **NULL**이면 **NULL**을 반환한다.
- **position** - 매칭을 수행할 *string*의 위치를 지정한다. 값을 생략하면 기본값 1이 적용된다. 값이 음수이거나 0이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다
- **occurrence** - 몇 번째 매칭을 사용할 지 지정한다. 값을 생략하면 기본값 0을 적용한다. 이 의미는 발생하는 모든 매칭을 사용한다는 의미이다. 값이 음수이면 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

- **match_type** – 함수의 매칭 옵션을 설정할 문자열을 지정한다. 값을 생략하면 기본값 'i'가 적용된다. 값이 'c' 또는 'I'가 아닌 경우 에러를 반환한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

STRING

```
-- 만약 패턴이 매칭되지 않으면, NULL을 반환한다
SELECT REGEXP_SUBSTR('12345abcdeabcde','[k-z]+');
```

```
regexp_substr('12345abcdeabcde','[k-z]+');
=====
NULL
```

```
-- 빈 문자열 패턴은 어떤 문자열과도 일치하지 않는다
SELECT REGEXP_SUBSTR('12345abcdeabcde','');
```

```
regexp_substr('12345abcdeabcde','')
=====
NULL
```

```
SELECT REGEXP_SUBSTR('Samseong-ro, Gangnam-gu, Seoul','',[^,]+,');
```

```
regexp_substr('Samseong-ro, Gangnam-gu, Seoul','',[^,]+,')
=====
', Gangnam-gu,'
```

```
SET NAMES utf8 COLLATE utf8_ko_cs;
SELECT REGEXP_SUBSTR('삼성로, 강남구, 서울특별시','\p{Hangul}+',1,2);
```

```
regexp_substr('삼성로, 강남구, 서울특별시' collate utf8_ko_cs,
'\p{Hangul}+' collate utf8_ko_cs, 1, 2)
=====
'강남구'
```

수치 연산 함수

목차

수치 연산 함수	7
● ABS	456
● ACOS	457
● ASIN	457
● ATAN	458
● ATAN2	459
● CEIL	459
● CONV	460
● COS	461
● COT	462
● CRC32	462
● DEGREES	463
● DRANDOM, DRAND	463
● EXP	465
● FLOOR	466
● HEX	467
● LN	467
● LOG	468
● LOG2	469

● LOG10	469
● MOD	470
● PI	471
● POW, POWER	471
● RADIANS	472
● RANDOM, RAND	473
● ROUND	475
● SIGN	476
● SIN	477
● SQRT	477
● TAN	478
● TRUNC, TRUNCATE	479
● WIDTH_BUCKET	480

ABS

ABS(*number_expr*)

함수는 지정된 인자 값의 절대값을 반환하며, 리턴 값의 타입은 주어진 인자의 타입과 같다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

number_expr – 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

number_expr의 타입

```
--it returns the absolute value of the argument
SELECT ABS(12.3), ABS(-12.3), ABS(-12.3000), ABS(0.0);
```

```
abs(12.3)          abs(-12.3)          abs(-12.3000)
abs(0.0)
=====
=====
12.3              12.3              12.3000      .
0
```

ACOS

ACOS(x)

ACOS 함수는 인자의 아크 코사인(arc cosine) 값을 반환한다. 즉, 코사인이 x 인 값을 라디안 단위로 반환하며, 리턴 값은 **DOUBLE** 타입이다. x 는 -1 이상 1 이하의 값이어야 하며, 그 외의 경우 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT ACOS(1), ACOS(0), ACOS(-1);
```

```
acos(1)          acos(0)          acos(-1)
=====
=====
0.000000000000000e+00  1.570796326794897e+00
3.141592653589793e+00
```

ASIN

ASIN(x)

ASIN 함수는 인자의 아크 사인(arc sine) 값을 반환한다. 즉, 사인이 x 인 값을 라디안 단위로 반환하며, 리턴 값은 **DOUBLE** 타입이다. x 는 -1 이상 1 이하의 값이어야 하며, 그 외의 경우 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT ASIN(1), ASIN(0), ASIN(-1);
```

asin(1)	asin(0)	asin(-1)
=====	=====	=====
=====	=====	=====
1.570796326794897e+00	0.000000000000000e+00	
-1.570796326794897e+00		

ATAN

ATAN([y], x)

ATAN 함수는 탄젠트가 x 인 값을 라디안 단위로 반환한다. 인자 y 는 생략될 수 있으며, y 가 지정되는 경우 함수는 y / x 의 아크 탄젠트 값을 계산한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x,y - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT ATAN(1), ATAN(-1), ATAN(1,-1);
```

```

              atan(1)              atan(-1)
atan2(1, -1)
=====
=====
       7.853981633974483e-01      -7.853981633974483e-01
       2.356194490192345e+00

```

ATAN2

ATAN2(y, x)

ATAN2 함수는 y / x 의 아크 탄젠트 값을 라디안 단위로 반환하며, **ATAN()** 와 유사하게 동작한다. 인자 x, y 가 모두 지정되어야 한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x,y - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT ATAN2(1,1), ATAN2(-1,-1), ATAN2(Pi(),0);
```

```

atan2(1, 1)              atan2(-1, -1)              atan2( pi(), 0)
=====
=====
       7.853981633974483e-01      -2.356194490192345e+00
       1.570796326794897e+00

```

CEIL

CEIL(number_operand)

CEIL 함수는 인자보다 크거나 같은 최소 정수 값을 인자의 타입으로 반환한다. 리턴 값은 **number_operand** 인자로 지정된 값의 유효 자릿수를 따른다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

number_operand - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

INT

```
SELECT CEIL(34567.34567), CEIL(-34567.34567);
```

```

ceil(34567.34567)      ceil(-34567.34567)
=====
34568.00000           -34567.00000

SELECT CEIL(34567.1), CEIL(-34567.1);
```

```

ceil(34567.1)          ceil(-34567.1)
=====
34568.0                -34567.0
```

CONV

CONV(*number*, *from_base*, *to_base*)

CONV 함수는 숫자의 진수를 변환하는 함수이며, 진수가 변환된 숫자를 문자열로 반환한다. 진수의 최소값은 2, 최대값은 36이다. 반환할 숫자의 진수를 나타내는 *to_base* 가 음수이면 입력 숫자인 *number* 가 부호 있는(signed) 숫자로 간주되고, 그 외의 경우에는 부호 없는(unsigned) 숫자로 간주된다. *from_base* 또는 *to_base*에 숫자로 변환되지 않는 문자열을 입력할 때 **cuprid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **number** - 입력 숫자
- **from_base** - 입력 숫자의 진수
- **to_base** - 반환할 숫자의 진수

반환 형식:

STRING

```
SELECT CONV('f',16,2);
```

```
'1111'
```

```
SELECT CONV('6H',20,8);
```

```
'211'
```

```
SELECT CONV(-30,10,-20);
```

```
'-1A'
```

COS

COS(x)

COS 함수는 인자의 코사인(cosine) 값을 반환하며, 인자 x 는 라디안 값이어야 한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT COS(pi()/6), COS(pi()/3), COS(pi());
```

```
cos( pi()/6)          cos( pi()/3)          cos( pi())
=====
=====
```

```
8.660254037844387e-01    5.000000000000001e-01
-1.000000000000000e+00
```

COT

COT(*x*)

COT 함수는 인자 *x* 의 코탄젠트(cotangent) 값을 반환한다. 즉, 탄젠트가 *x* 인 값을 라디안 단위로 반환하며, 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT COT(1), COT(-1), COT(0);
```

```

cot(1)                cot(-1)    cot(0)
=====
=====
6.420926159343306e-01  -6.420926159343306e-01  NULL
```

CRC32

CRC32(*string*)

CRC32 함수는 32비트 정수로 순환 중복 검사 값을 반환한다. NULL을 입력하면 NULL을 반환한다. :param string: 문자열 값을 반환하는 표현식 :rtype: INTEGER

```
SELECT CRC32('cubrid');
```

```

crc32('cubrid')
=====
908740081
```

DEGREES

DEGREES(*x*)

DEGREES 함수는 라디안 단위로 지정된 인자 *x* 를 각도로 환산하여 반환한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT DEGREES(pi()/6), DEGREES(pi()/3), DEGREES (pi());
```

```
degrees( pi()/6)      degrees( pi()/3)      degrees(
pi())
=====
=====
3.000000000000000e+01  5.999999999999999e+01
1.800000000000000e+02
```

DRANDOM, DRAND

DRANDOM(*[seed]*)

DRAND(*[seed]*)

DRANDOM / **DRAND** 함수는 구간 0.0 이상 1.0 미만의 구간에서 임의의 이중 정밀도(double-precision) 부동 소수점 값을 반환하며, *seed* 인자를 지정할 수 있다. *seed* 인자의 타입은 **INTEGER** 이며, 실수가 지정되면 반올림하고, **INTEGER** 범위를 초과하면 에러를 반환한다.

seed 값이 주어지지 않은 경우 **DRAND()**는 연산을 출력하는 행(row)의 개수와 관계없이 한 문장 내에서 1회만 연산을 수행하여 오직 한 개의 임의값만 생성하는 반면, **DRANDOM()**는 함수가 호출될 때마다 매번 연산을 수행하므로 한 문장 내에서 여러 개의 다른 임의 값을 생성한다. 따라서, 무작위 순서로 행을 출력하기 위해서는 **ORDER BY** 절에 **DRANDOM()**을 이용해야 한다. 무작위 정수값을 구하기 위해서는 **RANDOM()** 를 사용한다.

매개변수:

seed – seed 값

반환 형식:

DOUBLE

```
SELECT DRAND(), DRAND(1), DRAND(1.4);
```

drand(1.4)	drand()	drand(1)
=====		
=====		
2.849646518006921e-001	4.163034446537495e-002	
4.163034446537495e-002		

```
CREATE TABLE rand_tbl (
  id INT,
  name VARCHAR(255)
);

INSERT INTO rand_tbl VALUES
(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e'),
(6, 'f'), (7, 'g'), (8, 'h'), (9, 'i'), (10, 'j');

SELECT * FROM rand_tbl;
```

id	name
1	'a'
2	'b'
3	'c'
4	'd'
5	'e'
6	'f'
7	'g'
8	'h'
9	'i'
10	'j'

```
--drandom() returns random values on every row
SELECT DRAND(), DRANDOM() FROM rand_tbl;
```

drand()

drandom()

```
=====
=====
7.638782921842098e-001    1.018707846308786e-001
7.638782921842098e-001    3.191320535905026e-001
7.638782921842098e-001    3.461714529862361e-001
7.638782921842098e-001    6.791894283883175e-001
7.638782921842098e-001    4.533829767754143e-001
7.638782921842098e-001    1.714224677266762e-001
7.638782921842098e-001    1.698049867244484e-001
7.638782921842098e-001    4.507583849604786e-002
7.638782921842098e-001    5.279091769157994e-001
7.638782921842098e-001    7.021088290047914e-001
```

```
--selecting rows in random order
SELECT * FROM rand_tbl ORDER BY DRANDOM();
```

```
      id  name
=====
        6  'f'
        2  'b'
        7  'g'
        8  'h'
        1  'a'
        4  'd'
       10  'j'
        9  'i'
        5  'e'
        3  'c'
```

EXP

EXP(x)

EXP 함수는 자연로그의 밑수인 e를 x 제공한 값을 **DOUBLE** 타입으로 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT EXP(1), EXP(0);
```

```
exp(1)                exp(0)
=====
2.718281828459045e+000 1.0000000000000000e+000
```

```
SELECT EXP(-1), EXP(2.00);
```

```
exp(-1)                exp(2.00)
=====
3.678794411714423e-001 7.389056098930650e+000
```

FLOOR

FLOOR(*number_operand*)

FLOOR 함수는 인자보다 작거나 같은 최대 정수 값을 반환하며, 리턴 값의 타입은 인자의 타입과 같다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

number_operand – 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

number_operand의 타입

```
--it returns the largest integer less than or equal to the arguments
SELECT FLOOR(34567.34567), FLOOR(-34567.34567);
```

```
floor(34567.34567)    floor(-34567.34567)
=====
34567.00000           -34568.00000
```

```
SELECT FLOOR(34567), FLOOR(-34567);
```

```

floor(34567)    floor(-34567)
=====
          34567          -34567

```

HEX

HEX(*n*)

HEX 함수는 문자열을 인자로 지정하면 해당 문자열에 대한 16진수 문자열을 반환하고, 숫자를 인자로 지정하면 해당 숫자에 대한 16진수 문자열을 반환한다. 숫자를 인자로 지정하면 CONV(num, 10, 16)과 같은 값을 반환한다.

매개변수:

n – 문자열 또는 숫자

반환 형식:

STRING

```
SELECT HEX('ab'), HEX(128), CONV(HEX(128), 16, 10);
```

```

hex('ab')           hex(128)           conv(hex(128), 16, 10)
=====
    '6162'           '80'              '128'

```

LN

LN(*x*)

LN 함수는 진수 *x*의 자연 로그(밑수가 *e*인 로그) 값을 반환하며, 리턴 값은 **DOUBLE** 타입이다. 진수 *x*가 0이거나 음수인 경우, 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 양수 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT ln(1), ln(2.72);
```

```

ln(1)                ln(2.72)
=====
0.000000000000000e+00  1.000631880307906e+00

```

LOG

LOG(x, y)

LOG 함수는 진수가 x 이고, 밑수가 y 인 로그 값을 반환하며, 리턴 값은 **DOUBLE** 타입이다. x, y 가 0이거나 음수인 경우, 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cnubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **x** - 로그의 밑수로, 양수이면서 1이 아닌 값을 반환하는 임의의 연산식이다.
- **y** - 로그의 진수로, 양수 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT log(2,1), log(2,8), log(10,1), log(10,1000);
```

```

log(2,1)            log(2,8)                log(10,
1)                log(10, 1000)
=====
=====
0.000000000000000e+00  3.000000000000000e+00
0.000000000000000e+00  3.000000000000000e+00

```


LOG2

LOG2(x)

LOG2 함수는 진수가 x 이고, 밑수가 2인 로그 값을 반환하며, 리턴 값은 **DOUBLE** 타입이다. 진수 x 가 0이거나 음수인 경우, 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 양수 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT log2(1), log2(8);
```

```

log2(1)                log2(8)
=====
0.0000000000000000e+00 3.0000000000000000e+00

```

LOG10

LOG10(x)

LOG10 함수는 진수 x 의 상용 로그 값을 반환하며, 리턴 값은 **DOUBLE** 타입이다. 진수 x 가 0이거나 음수인 경우, 에러를 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 양수 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT log10(1), log10(1000);
```

```

log10(1)                                log10(1000)
=====
0.0000000000000000e+00                3.0000000000000000e+00

```

MOD

MOD(*m*, *n*)

MOD 함수는 첫 번째 인자 *m*을 두 번째 인자 *n*으로 나눈 나머지 값을 반환하며, 만약 *n*이 0이거나 *m*보다 큰 경우, 나누기 연산을 수행하지 않고 *m* 값을 그대로 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

주의할 점은 피제수, 즉 **MOD** 함수의 인자 *m*이 음수인 경우, 전형적인 연산(classical modulus) 방식과 다르게 동작한다는 점이다. 아래의 표를 참고한다.

MOD 함수의 결과

m	n	MOD(m, n)	Classical Modulus m-n*FLOOR(m/n)
11	4	3	3
11	-4	3	-1
-11	4	-3	1
-11	-4	-3	-3
11	0	11	0으로 나누기 에러

매개변수:

- **m** – 피제수를 나타내며, 수치 값을 반환하는 연산식이다.
- **n** – 제수를 나타내며, 수치 값을 반환하는 연산식이다.

반환 형식:

m/n 연산 결과의 타입

```
--it returns the remainder of m divided by n
SELECT MOD(11, 4), MOD(11, -4), MOD(-11, 4), MOD(-11, -4), MOD(11,0);
```

```
      mod(11, 4)   mod(11, -4)   mod(-11, 4)   mod(-11, -4)   mod(11,
0)
=====
              3             3             -3             -3             11
```

```
SELECT MOD(11.0, 4), MOD(11.000, 4), MOD(11, 4.0), MOD(11, 4.000);
```

```
      mod(11.0, 4)           mod(11.000, 4)           mod(11, 4.0)
mod(11, 4.000)
=====
=====
      3.0              3.000              3.0
      3.000
```

PI

PI()

PI 함수는 π 값을 반환하며, 리턴 값은 DOUBLE 타입이다.

반환 형식:

DOUBLE

```
SELECT PI(), PI()/2;
```

```
      pi()              pi()/2
=====
      3.141592653589793e+00      1.570796326794897e+00
```

POW, POWER

POW(x, y)

POWER(x, y)

POW 함수와 **POWER** 함수는 동일하며, 지정된 밑수 x 를 지수 y 만큼 거듭제곱한 값을 반환한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **x** - 밑수를 나타내며, 수치 값을 반환하는 연산식이다.
- **y** - 지수를 나타내며, 수치 값을 반환하는 연산식이다. 밑수가 음수인 경우, 지수는 반드시 정수가 지정되어야 한다.

반환 형식:

DOUBLE

```
SELECT POWER(2, 5), POWER(-2, 5), POWER(0, 0), POWER(1,0);
```

```
power(2, 5)           power(-2, 5)           power(0,
0)
power(1, 0)
=====
=====
3.200000000000000e+01  -3.200000000000000e+01
1.000000000000000e+00  1.000000000000000e+00
```

```
--it returns an error when the negative base is powered by a non-int
exponent
SELECT POWER(-2, -5.1), POWER(-2, -5.1);
```

```
ERROR: Argument of power() is out of range.
```

RADIANS**RADIANS(x)**

RADIANS 함수는 각도 단위로 지정된 인자 x 를 라디안 단위로 환산하여 리턴한다. 리턴 값은 **DOUBLE** 타입이다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT RADIANS(90), RADIANS(180), RADIANS(360);
```

radians(90)	radians(180)	radians(360)
=====	=====	=====
=====		
1.570796326794897e+00	3.141592653589793e+00	
6.283185307179586e+00		

RANDOM, RAND

RANDOM([*seed*])

RAND([*seed*])

RANDOM / **RAND** 함수는 0 이상 2^{31} 미만 구간에서 임의의 정수 값을 반환하며, *seed* 인자를 지정할 수 있다. *seed* 인자의 타입은 **INTEGER** 이며, 실수가 지정되면 반올림하고 **INTEGER** 범위를 초과하면 에러를 반환한다.

seed 값이 주어지지 않은 경우 **RAND()**는 연산을 출력하는 행(row)의 개수와 관계없이 한 문장 내에서 1회만 연산을 수행하여 오직 한 개의 임의값만 생성하는 반면, **RANDOM()**은 함수가 호출될 때마다 매번 연산을 수행하므로 한 문장 내에서 여러 개의 다른 임의 값을 생성한다. 따라서, 무작위 순서로 행을 출력하기 위해서는 **RANDOM()**을 이용해야 한다.

무작위 실수 값을 구하기 위해서는 **DRANDOM()** 를 사용한다.

매개변수:

seed

반환 형식:

INT

```
SELECT RAND(), RAND(1), RAND(1.4);
```

rand()	rand(1)	rand(1.4)
1526981144	89400484	89400484

```
--creating a new table
SELECT * FROM rand_tbl;
```

id	name
1	'a'
2	'b'
3	'c'
4	'd'
5	'e'
6	'f'
7	'g'
8	'h'
9	'i'
10	'j'

```
--random() returns random values on every row
SELECT RAND(),RANDOM() FROM rand_tbl;
```

rand()	random()
2078876566	1753698891
2078876566	1508854032
2078876566	625052132
2078876566	279624236
2078876566	1449981446
2078876566	1360529082
2078876566	1563510619
2078876566	1598680194
2078876566	1160177096
2078876566	2075234419

```
--selecting rows in random order
SELECT * FROM rand_tbl ORDER BY RANDOM();
```

```

      id  name
=====
      6  'f'
      1  'a'
      5  'e'
      4  'd'
      2  'b'
      7  'g'
     10  'j'
      9  'i'
      3  'c'
      8  'h'

```

ROUND

ROUND(*number_operand*, *integer*)

ROUND 함수는 지정된 인자 *number_operand* 를 소수점 아래 *integer* 자리까지 반올림한 값을 반환한다. 반올림할 자릿수를 지정하는 *integer* 인자가 생략되거나 0인 경우에는 소수점 아래 첫째 자리에서 반올림한다. 그리고 *integer* 인자가 음수이면, 소수점 위 자리, 즉 정수부에서 반올림한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **number_operand** - 수치 값을 반환하는 임의의 연산식이다.
- **integer** - 반올림 처리할 위치를 지정한다. 양의 정수 *n* 이 지정되면 소수점 아래 *n* 자리까지 표현되고, 음의 정수 *n* 이 지정되면 소수점 위 *n* 자리에서 반올림한다.

반환 형식:

*number_operand*의 타입

```

--it rounds a number to one decimal point when the second argument
is omitted
SELECT ROUND(34567.34567), ROUND(-34567.34567);

```

```
round(34567.34567, 0)    round(-34567.34567, 0)
=====
34567.00000            -34567.00000
```

```
--it rounds a number to three decimal point
SELECT ROUND(34567.34567, 3), ROUND(-34567.34567, 3) FROM db_root;
```

```
round(34567.34567, 3)    round(-34567.34567, 3)
=====
34567.34600            -34567.34600
```

```
--it rounds a number three digit to the left of the decimal point
SELECT ROUND(34567.34567, -3), ROUND(-34567.34567, -3);
```

```
round(34567.34567, -3)    round(-34567.34567, -3)
=====
35000.00000            -35000.00000
```

SIGN

SIGN(*number_operand*)

SIGN 함수는 지정된 인자 값의 부호를 반환한다. 양수이면 1을, 음수이면 -1을, 0이면 0을 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

number_operand - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

INT

```
--it returns the sign of the argument
SELECT SIGN(12.3), SIGN(-12.3), SIGN(0);
```



```

      sign(12.3)    sign(-12.3)    sign(0)
=====
              1              -1              0

```

SIN

SIN(x)

SIN 함수는 인자의 사인(sine) 값을 반환하며, 인자 x 는 라디안 값이어야 한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT SIN(pi()/6), SIN(pi()/3), SIN(pi());
```

```

      sin( pi()/6)          sin( pi()/3)          sin( pi())
=====
=====
      4.999999999999999e-01      8.660254037844386e-01
      1.224646799147353e-16

```

SQRT

SQRT(x)

SQRT 함수는 x 의 제곱근(square root) 값을 **DOUBLE** 타입으로 반환한다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다. 만약, 음수이면 에러를 반환한다.

반환 형식:

DOUBLE

```
SELECT SQRT(4), SQRT(16.0);
```

sqrt(4)	sqrt(16.0)
=====	=====
2.0000000000000000e+00	4.0000000000000000e+00

TAN

TAN(**x**)

TAN 함수는 인자의 탄젠트(tangent) 값을 반환하며, 인자 **x** 는 라디안 값이어야 한다. 리턴 값은 **DOUBLE** 타입이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

x - 수치 값을 반환하는 임의의 연산식이다.

반환 형식:

DOUBLE

```
SELECT TAN(pi()/6), TAN(pi()/3), TAN(pi()/4);
```

tan(pi()/6)	tan(pi()/3)	tan(pi()/4)
=====	=====	=====
5.773502691896257e-01	1.732050807568877e+00	
9.999999999999999e-01		

TRUNC, TRUNCATE

TRUNC(*x*[, *dec*])TRUNCATE(*x*, *dec*)

TRUNC 함수와 **TRUNCATE** 함수는 지정된 인자 *x* 의 소수점 아래 숫자가 *dec* 자리까지 표현되도록 버림(truncation)한 값을 반환한다. 단, **TRUNC** 함수의 *dec* 인자는 생략할 수 있지만, **TRUNCATE** 함수의 *dec* 인자는 생략할 수 없다. 버림할 위치를 지정하는 *dec* 인자가 음수이면 정수부의 소수점 위 *dec* 번째 자리까지 0으로 표시한다. 리턴 값의 표현 자릿수는 인자 *x* 를 따른다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

매개변수:

- **x** – 수치 값을 반환하는 임의의 연산식이다.
- **dec** – 버림할 위치를 지정한다. 양의 정수 *n* 이 지정되면 소수점 아래 *n* 자리까지 표현되고, 음의 정수 *n* 이 지정되면 소수점 위 *n* 자리까지 0으로 표시한다. *dec* 인자가 0이거나 생략되면 소수부를 버림한다. 단, **TRUNCATE** 함수에서는 *dec* 인자를 생략할 수 없다.

반환 형식:

*x*의 타입

```
--it returns a number truncated to 0 places
SELECT TRUNC(34567.34567), TRUNCATE(34567.34567, 0);
```

```
trunc(34567.34567, 0)    trunc(34567.34567, 0)
=====
34567.00000             34567.00000
```

```
--it returns a number truncated to three decimal places
SELECT TRUNC(34567.34567, 3), TRUNC(-34567.34567, 3);
```

```
trunc(34567.34567, 3)    trunc(-34567.34567, 3)
=====
34567.34500             -34567.34500
```

```
--it returns a number truncated to three digits left of the decimal
point
SELECT TRUNC(34567.34567, -3), TRUNC(-34567.34567, -3);
```

```
trunc(34567.34567, -3)    trunc(-34567.34567, -3)
=====
34000.00000             -34000.00000
```

WIDTH_BUCKET

WIDTH_BUCKET(*expression, from, to, num_buckets*)

WIDTH_BUCKET 함수는 순차적인 데이터 집합을 균등한 범위로 부여된 일련의 버킷으로 나누며, 각 행에 적당한 버킷 번호를 1부터 할당한다. 즉, WIDTH_BUCKET 함수는 equi-width histogram을 생성한다. 반환되는 값은 정수이다. 숫자로 변환되지 않는 문자열을 입력할 때 **cubrid.conf**의 **return_null_on_function_errors** 파라미터의 값이 no(기본값)면 에러, yes면 NULL을 반환한다.

이 함수는 주어진 버킷 개수로 범위를 균등하게 나누어 버킷 번호를 부여한다. 즉, 버킷마다 각 범위의 넓이는 균등하다.

참고로 **NTILE()** 분석 함수는 이에 비해 주어진 버킷 개수로 전체 행의 개수를 균등하게 나누어 버킷 번호를 부여한다. 즉, 버킷마다 각 행의 개수는 균등하다.

매개변수:

- **expression** – 버킷 번호를 부여받기 위한 입력 값. 수치 값을 반환하는 임의의 연산식을 지정한다.
- **from** – *expression*이 취할 수 있는 범위의 시작값으로, 이 값은 전체 범위 안에 포함된다.
- **to** – *expression*이 취할 수 있는 범위의 마지막 값으로, 이 값은 전체 범위 안에 포함되지 않는다.
- **num_buckets** – 버킷의 개수. 추가로 범위 밖의 내용을 담기 위한 0번 버킷과 (*num_buckets* + 1)번 버킷이 생성된다.

반환 형식:

INT

*expression*은 버킷 번호를 부여받기 위한 입력 데이터이다. *from*과 *to* 값으로 숫자형 타입과 날짜/시간 타입의 값 또는 날짜/시간 타입으로 변환 가능한 문자열이 입력될 수 있다. 전체 범위에서 *from*은 범위에 포함되지만 *to*는 범위 밖에 존재한다.

예를 들어 `WIDTH_BUCKET (score, 80, 50, 3)`이 반환하는 값은 `score`가

- 80보다 크면 0,
- [80, 70)이면 1,
- [70, 60)이면 2,
- [60, 50)이면 3,
- 50보다 작거나 같으면 4가 된다.

다음 예제는 80점보다 작거나 같고 50점보다 큰 범위를 1부터 3까지 균등한 점수 범위로 나누어 등급을 부여한다. 해당 범위를 벗어나는 경우 80점보다 크면 0, 50점이거나 50점보다 작으면 4등급을 부여한다.

```
CREATE TABLE t_score (name VARCHAR(10), score INT);
INSERT INTO t_score VALUES
  ('Amie', 60),
  ('Jane', 80),
  ('Lora', 60),
  ('James', 75),
  ('Peter', 70),
  ('Tom', 50),
  ('Ralph', 99),
  ('David', 55);

SELECT name, score, WIDTH_BUCKET (score, 80, 50, 3) grade
FROM t_score
ORDER BY grade ASC, score DESC;
```

name	score	grade
'Ralph'	99	0
'Jane'	80	1
'James'	75	1
'Peter'	70	2
'Amie'	60	3
'Lora'	60	3
'David'	55	3
'Tom'	50	4

다음의 예에서 **WIDTH_BUCKET** 함수는 birthdate의 지정 범위를 균등하게 나누고 이를 기준으로 버킷 번호를 부여한다. 8 명의 고객을 생년월일을 기준으로 '1950-01-01'부터 '1999-12-31'까지의 범위를 5개로 균등 분할하며, birthdate 값이 범위를 벗어나면 0 또는 버킷 개수 + 1인 6을 반환한다.

```
CREATE TABLE t_customer (name VARCHAR(10), birthdate DATE);
INSERT INTO t_customer VALUES
  ('Amie', date'1978-03-18'),
  ('Jane', date'1983-05-12'),
  ('Lora', date'1987-03-26'),
  ('James', date'1948-12-28'),
  ('Peter', date'1988-10-25'),
  ('Tom', date'1980-07-28'),
  ('Ralph', date'1995-03-17'),
  ('David', date'1986-07-28');

SELECT name, birthdate, WIDTH_BUCKET (birthdate, date'1950-01-01',
date'2000-1-1', 5) age_group
FROM t_customer
ORDER BY birthdate;
```

name	birthdate	age_group
'James'	12/28/1948	0
'Amie'	03/18/1978	4
'Tom'	07/28/1980	4
'Jane'	05/12/1983	5
'David'	07/28/1986	5
'Lora'	03/26/1987	5
'Peter'	10/25/1988	5
'Ralph'	03/17/1995	6

날짜/시간 함수와 연산자

목차

날짜/시간 함수와 연산자

7

- ADDDATE, DATE_ADD 485
- ADDTIME 489
- ADD_MONTHS 490

● CURDATE, CURRENT_DATE	491
● CURRENT_DATETIME, NOW	493
● CURTIME, CURRENT_TIME	495
● CURRENT_TIMESTAMP, LOCALTIME, LOCALTIMESTAMP	496
● DATE	143
● DATEDIFF	498
● DATE_SUB, SUBDATE	499
● DAY, DAYOFMONTH	501
● DAYOFWEEK	502
● DAYOFYEAR	502
● EXTRACT	503
● FROM_DAYS	505
● FROM_TZ	506
● FROM_UNIXTIME	507
● HOUR	508
● LAST_DAY	509
● MAKEDATE	510
● MAKETIME	511
● MINUTE	512
● MONTH	513
● MONTHS_BETWEEN	514

● NEW_TIME	515
● QUARTER	516
● ROUND	475
● SEC_TO_TIME	520
● SECOND	521
● SYS_DATE, SYSDATE	522
● SYS_DATETIME, SYSDATETIME	523
● SYS_TIME, SYSTIME	525
● SYS_TIMESTAMP, SYSTIMESTAMP	526
● TIME	143
● TIME_TO_SEC	529
● TIMEDIFF	530
● TIMESTAMP	144
● TO_DAYS	532
● TRUNC	533
● TZ_OFFSET	535
● UNIX_TIMESTAMP	536
● UTC_DATE	537
● UTC_TIME	537
● WEEK	538
● WEEKDAY	540

● YEAR

541

ADDDATE, DATE_ADD

ADDDATE(*date*, *INTERVAL expr unit*)ADDDATE(*date*, *days*)DATE_ADD(*date*, *INTERVAL expr unit*)

ADDDATE 함수와 **DATE_ADD** 함수는 동일하며, 특정 **DATE** 값에 대해 덧셈 또는 뺄셈을 실행한다. 리턴 값은 **DATE** 타입 또는 **DATETIME** 타입이다. **DATETIME** 타입을 반환하는 경우는 다음과 같다.

- 첫 번째 인자가 **DATETIME** 타입 또는 **TIMESTAMP** 타입인 경우
- 첫 번째 인자가 **DATE** 타입이고 **INTERVAL** 값의 단위가 날짜 단위 미만으로 지정된 경우

위의 경우 외에 **DATETIME** 타입의 결과 값을 반환하려면 **CAST()** 함수를 이용하여 첫 번째 인자 값의 타입을 변환해야 한다. 연산 결과의 날짜가 해당 월의 마지막 날짜를 초과하면, 해당 월의 말일을 적용하여 유효한 **DATE** 값을 반환한다.

입력 인자의 날짜와 시간 값이 모두 0이면 시스템 파라미터 **return_null_on_function_errors**의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors**가 yes이면 **NULL**을 반환하고 no이면 에러를 반환하며, 기본값은 **no**이다.

계산 결과가 '0000-00-00 00:00:00'과 '0001-01-01 00:00:00' 사이이면, 날짜와 시간 값이 모두 0인 **DATE** 또는 **DATETIME** 타입의 값을 반환한다. 그러나 JDBC 프로그램에서는 연결 URL 속성인 **zeroDateTimeBehavior**의 설정에 따라 동작이 달라진다. JDBC의 연결 URL 속성은 **연결 설정**을 참고하면 된다.

매개변수:

- **date** - **DATE**, **DATETIME** 또는 **TIMESTAMP** 타입의 연산식이며, 시작 날짜를 의미한다. 만약, '2006-07-00'와 같이 유효하지 않은 **DATE** 값이 지정되면, 에러를 반환한다.
- **expr** - 시작 날짜로부터 더할 시간 간격 값(interval value)을 의미하며, **INTERVAL** 키워드 뒤에 음수가 명시되면 시작 날짜로부터 시간 간격 값을 뺀다.
- **unit** - *expr* 수식에 명시된 시간 간격 값의 단위를 의미하며, 아래의 테이블을 참고하여 시간 간격 값 해석을 위한 형식을 지정할 수 있다. *expr*의 단위 값이 *unit*에서 요구하는 단위 값의 개수보다 적을 경우 가장 작은 단위부터 채운다. 예를 들어, **HOURL_SECONDS**의 경우

‘HOURS:MINUTES:SECONDS’와 같이 3개의 값이 요구되는데,
“1:1” 처럼 2개의 값만 주어지면 ‘MINUTES:SECONDS’로 간주
한다.

반환 형식:

DATE 또는 DATETIME

unit에 대한 expr 형식

unit 값	expr 형식	예
MILLISECOND	MILLISECONDS	ADDDATE(SYSDATE, INTERVAL 123 MILLISECOND)
SECOND	SECONDS	ADDDATE(SYSDATE, INTERVAL 123 SECOND)
MINUTE	MINUTES	ADDDATE(SYSDATE, INTERVAL 123 MINUTE)
HOUR	HOURS	ADDDATE(SYSDATE, INTERVAL 123 HOUR)
DAY	DAYS	ADDDATE(SYSDATE, INTERVAL 123 DAY)
WEEK	WEEKS	ADDDATE(SYSDATE, INTERVAL 123 WEEK)
MONTH	MONTHS	ADDDATE(SYSDATE, INTERVAL 12 MONTH)
QUARTER	QUARTERS	ADDDATE(SYSDATE, INTERVAL 12 QUARTER)

unit 값	expr 형식	예
YEAR	YEARS	ADDDATE(SYSDATE, INTERVAL 12 YEAR)
SECOND_MILLISECOND	'SECONDS.MILLISECONDS'	ADDDATE(SYSDATE, INTERVAL '12.123' SECOND_MILLISECOND)
MINUTE_MILLISECOND	'MINUTES:SECONDS.MILLISEC ONDS'	ADDDATE(SYSDATE, INTERVAL '12:12.123' MINUTE_MILLISECOND)
MINUTE_SECOND	'MINUTES:SECONDS'	ADDDATE(SYSDATE, INTERVAL '12:12' MINUTE_SECOND)
HOUR_MILLISECOND	'HOURS:MINUTES:SECONDS.M ILLISECONDS'	ADDDATE(SYSDATE, INTERVAL '12:12:12.123' HOUR_MILLISECOND)
HOUR_SECOND	'HOURS:MINUTES:SECONDS'	ADDDATE(SYSDATE, INTERVAL '12:12:12' HOUR_SECOND)
HOUR_MINUTE	'HOURS:MINUTES'	ADDDATE(SYSDATE, INTERVAL '12:12' HOUR_MINUTE)
DAY_MILLISECOND	'DAYS HOURS:MINUTES:SECONDS.M ILLISECONDS'	ADDDATE(SYSDATE, INTERVAL '12 12:12:12.123' DAY_MILLISECOND)

unit 값	expr 형식	예
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'	ADDDATE(SYSDATE, INTERVAL '12 12:12:12' DAY_SECOND)
DAY_MINUTE	'DAYS HOURS:MINUTES'	ADDDATE(SYSDATE, INTERVAL '12 12:12' DAY_MINUTE)
DAY_HOUR	'DAYS HOURS'	ADDDATE(SYSDATE, INTERVAL '12 12' DAY_HOUR)
YEAR_MONTH	'YEARS-MONTHS'	ADDDATE(SYSDATE, INTERVAL '12-13' YEAR_MONTH)

```
SELECT SYSDATE, ADDDATE(SYSDATE,INTERVAL 24 HOUR), ADDDATE(SYSDATE,
1);
```

```
03/30/2010 12:00:00.000 AM 03/31/2010 03/31/2010
```

```
--it subtracts days when argument < 0
SELECT SYSDATE, ADDDATE(SYSDATE,INTERVAL -24 HOUR), ADDDATE(SYSDATE,
-1);
```

```
03/30/2010 12:00:00.000 AM 03/29/2010 03/29/2010
```

```
--when expr is not fully specified for unit
SELECT SYS_DATETIME, ADDDATE(SYS_DATETIME, INTERVAL '1:20'
HOUR_SECOND);
```

```
06:18:24.149 PM 06/28/2010 06:19:44.149 PM 06/28/2010
```

```
SELECT ADDDATE('0000-00-00', 1 );
```

```
ERROR: Conversion error in date format.
```

```
SELECT ADDDATE('0001-01-01 00:00:00', -1);
```

```
'12:00:00.000 AM 00/00/0000'
```

ADDTIME

ADDTIME(*expr1*, *expr2*)

ADDTIME 함수는 특정 시간 값에 대해 덧셈 또는 뺄셈을 실행한다. 첫 번째 인자는 **DATE**, **DATETIME**, **TIMESTAMP** 또는 **TIME** 타입이며, 두 번째 인자는 **TIME**, **DATETIME** 또는 **TIMESTAMP** 타입이다. 두 번째 인자는 반드시 시간을 포함해야 하며, 두 번째 인자의 날짜는 무시된다. 각 인자의 타입에 따른 반환 타입은 다음과 같다.

첫 번째 인자 타입	두 번째 인자 타입	반환 타입	참고
TIME	TIME, DATETIME, TIMESTAMP	TIME	결과 값은 24시를 넘어서는 안 된다.
DATE	TIME, DATETIME, TIMESTAMP	DATETIME	
DATETIME	TIME, DATETIME, TIMESTAMP	DATETIME	
날짜/시간 문자열	TIME, DATETIME, TIMESTAMP 또는 시간 문자열	VARCHAR	결과 문자열은 시간을 포함한 문자열이다.

매개변수:

- **expr1** – **DATE**, **DATETIME**, **TIMESTAMP**, **TIME** 타입 또는 날짜/시간 문자열

- **expr2** – DATETIME, TIMESTAMP, TIME 타입 또는 시간 문자열

```
SELECT ADDTIME(datetime'2007-12-31 23:59:59', time'1:1:2');
```

```
01:01:01.000 AM 01/01/2008
```

```
SELECT ADDTIME(time'01:00:00', time'02:00:01');
```

```
03:00:01 AM
```

ADD_MONTHS

ADD_MONTHS(*date_argument*, *month*)

ADD_MONTHS 함수는 **DATE** 타입의 *date_argument* 표현식에 *month* 값을 더하고 **DATE** 타입 값을 반환한다. 인자로 지정된 값의 일(*dd*)이 연산 결과 값의 월에 존재하면 해당 일(*dd*)을 반환한다. 존재하지 않으면 해당 월의 마지막 날(*dd*)을 반환한다. 연산 결과 값이 **DATE** 타입의 표현식 범위를 초과하는 경우 오류를 반환한다.

매개변수:

- **date_argument** – **DATE** 타입의 표현식을 지정한다.
TIMESTAMP 또는 **DATETIME** 값을 지정하려면 **DATE** 타입으로 명시적 변환을 해야 한다. 값이 **NULL** 이면 **NULL** 을 반환한다.
- **month** – *date_argument* 에 더할 개월 수를 지정한다. 양수와 음수 모두 지정할 수 있다. 지정한 값이 정수 타입이 아닐 경우 자동으로 정수 타입으로 변환(소수점 아래 첫째 자리를 반올림 처리)된다. 값이 **NULL** 이면 **NULL** 을 반환한다.

```
--it returns DATE type value by adding month to the first argument
SELECT ADD_MONTHS(DATE '2008-12-25', 5), ADD_MONTHS(DATE '2008-12-25', -5);
```

```
05/25/2009
```

```
07/25/2008
```

```
SELECT ADD_MONTHS(DATE '2008-12-31', 5.5), ADD_MONTHS(DATE
'2008-12-31', -5.5);
```

```
06/30/2009
```

```
06/30/2008
```

```
SELECT ADD_MONTHS(CAST (SYS_DATETIME AS DATE), 5), ADD_MONTHS(CAST
(SYS_TIMESTAMP AS DATE), 5);
```

```
07/03/2010
```

```
07/03/2010
```

다음은 타임존 타입 값 사용 예이다. 타임존 관련 설명은 [타임존이 있는 날짜/시간 데이터 타입](#) 을 참고한다.

```
SELECT ADD_MONTHS (datetimeeltz'2001-10-11 10:11:12', 1);
```

```
11/11/2001
```

```
SELECT ADD_MONTHS (datetimeetz'2001-10-11 10:11:12 Europe/Paris', 1);
```

```
11/11/2001
```

```
SELECT ADD_MONTHS (timestampltz'2001-10-11 10:11:12', 1);
```

```
11/11/2001
```

```
SELECT ADD_MONTHS (timestamptz'2001-10-11 10:11:12 Europe/Paris', 1);
```

```
11/11/2001
```

CURDATE, CURRENT_DATE

CURDATE()

CURRENT_DATE()

CURRENT_DATE

CURDATE (), **CURRENT_DATE** 및 **CURRENT_DATE** () 는 서로 바꿔 사용할 수 있으며 세션의 현재 날짜를 **DATE** 타입(*MM/DD/YYYY* 또는 *YYYY-MM-DD*)으로 반환한다. 산술 연산의 단위는 일이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **SYS_DATE**, **SYSDATE** 와 동일하다. 차이점은 **SYS_DATE**, **SYSDATE** 및 다음 예를 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

입력 인자의 연, 월, 일이 모두 0이면 반환되는 값은 **return_null_on_function_errors** 시스템 변수에 의해서 결정된다. 그 변수가 **yes**로 설정되었으면 **NULL** 이 반환된다. 그 변수가 **no**로 설정되었으면 오류가 반환된다. 기본값은 **no** 이다.

반환 형식:

DATE

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

dbtimezone           sessiontimezone
=====
'Asia/Seoul'         'Asia/Seoul'
```

```
-- it returns the current date in DATE type
```

```
SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE, SYS_DATE, SYSDATE;
```

```

CURRENT_DATE  CURRENT_DATE  CURRENT_DATE  SYS_DATE
SYS_DATE
=====
===
02/05/2016    02/05/2016    02/05/2016    02/05/2016  02/05/
2016
```

```
-- it returns the date 60 days added to the current date
```

```
SELECT CURDATE()+60;
```

```

CURRENT_DATE +60
=====
04/05/2016
```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'
```

```
SET TIME ZONE 'America/Los_Angeles';
```



```

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'America/Los_Angeles'

-- Note that CURDATE() and SYS_DATE returns different results

SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE, SYS_DATE, SYSDATE;

    CURRENT_DATE    CURRENT_DATE    CURRENT_DATE    SYS_DATE
SYS_DATE
=====
===
    02/04/2016      02/04/2016      02/04/2016      02/05/2016    02/05/
2016

```

⚠ 경고

10.0 이후로 **CURDATE ()**, **CURRENT_DATE**, **CURRENT_DATE ()**가 **SYS_DATE** 및 **SYSDATE** 와 다르다.
9.x 및 그 이전 버전에서는 동일하다.

CURRENT_DATETIME, NOW

CURRENT_DATETIME()

CURRENT_DATETIME

NOW()

CURRENT_DATETIME, **CURRENT_DATETIME ()** 및 **NOW ()**는 서로 바꿔 사용할 수 있으며 세션의 현재 날짜와 시간을 **DATETIME** 타입으로 반환한다. 산술 연산의 단위는 밀리초(msec)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **SYS_DATETIME**, **SYSDATETIME** 과 동일하다. 차이점은 **SYS_DATETIME**, **SYSDATETIME** 을 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

반환 형식:

DATETIME

```

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone          sessiontimezone

```

```
=====
'Asia/Seoul'          'Asia/Seoul'

-- it returns the current date and time in DATETIME type

SELECT NOW(), SYS_DATETIME;

CURRENT_DATETIME          SYS_DATETIME
=====
04:05:09.292 PM 02/05/2016    04:05:09.292 PM 02/05/2016
```

```
-- it returns the timestamp value 1 hour added to the current
sys_datetime value

SELECT TO_CHAR(SYSDATETIME+3600*1000, 'YYYY-MM-DD HH24:MI');

to_char( SYS_DATETIME +3600*1000, 'YYYY-MM-DD HH24:MI')
=====
'2016-02-05 17:05'
```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'

set time zone 'America/Los_Angeles';

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone          sessiontimezone
=====
'Asia/Seoul'          'America/Los_Angeles'

-- NOW()와 SYS_DATETIME은 서로 다른 결과를 반환하므로 주의한다.

SELECT NOW(), SYS_DATETIME;

CURRENT_DATETIME          SYS_DATETIME
=====
11:08:57.041 PM 02/04/2016    04:08:57.041 PM 02/05/2016
```

⚠ 경고

10.0 이후로 **CURRENT_DATETIME ()**, **NOW ()**가 **SYS_DATEIME**, **SYSDATETIME** 과 다르다. 9.x 및 그 이전 버전에서는 동일하다.

CURTIME, CURRENT_TIME

CURTIME()

CURRENT_TIME

CURRENT_TIME()

CURTIME (), **CURRENT_TIME** 및 **CURRENT_TIME** ()은 서로 바꿔 사용할 수 있으며 세션의 현재 시간을 **TIME** 타입(*HH:MI:SS*)으로 반환한다. 산술 연산의 단위는 초(sec)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **SYS_TIME**, **SYSTIME** 과 동일하다. 차이점은 **SYS_TIME**, **SYSTIME** 을 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

반환 형식:

TIME

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'Asia/Seoul'

-- it returns the current time in TIME type

SELECT CURTIME(), CURRENT_TIME(), CURRENT_TIME, SYS_TIME, SYSTIME;

      CURRENT_TIME      CURRENT_TIME      CURRENT_TIME      SYS_TIME
SYS_TIME
=====
=====
04:22:54 PM      04:22:54 PM      04:22:54 PM      04:22:54 PM      04:22:
54 PM
```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'

SET TIME ZONE 'AMERICA/LOS_ANGELES';

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'America/Los_Angeles'

-- Note that CURTIME() and SYS_TIME return different results

SELECT CURTIME(), CURRENT_TIME(), CURRENT_TIME, SYS_TIME, SYSTIME;
```

```

CURRENT_TIME    CURRENT_TIME    CURRENT_TIME    SYS_TIME
SYS_TIME
=====
=====
11:23:16 PM    11:23:16 PM    11:23:16 PM    04:23:16 PM    04:23:
16 PM

```

⚠ 경고

10.0 이후로 **CURTIME ()**, **CURRENT_TIME ()**이 **SYS_TIME**, **SYSTIME** 과 다르다. 9.x 및 그 이전 버전에서는 동일하다.

CURRENT_TIMESTAMP, LOCALTIME, LOCALTIMESTAMP

CURRENT_TIMESTAMP

CURRENT_TIMESTAMP()

LOCALTIME

LOCALTIME()

LOCALTIMESTAMP

LOCALTIMESTAMP()

CURRENT_TIMESTAMP, **CURRENT_TIMESTAMP ()**, **LOCALTIME**, **LOCALTIME ()**, **LOCALTIMESTAMP**, **LOCALTIMESTAMP ()**는 동일하며, 현재 날짜와 시간을 **TIMESTAMP** 타입으로 반환한다. 산술 연산의 단위는 초(sec)다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **SYS_TIMESTAMP**, **SYSTIMESTAMP** 와 동일하다. 차이점은 **SYS_TIMESTAMP**, **SYSTIMESTAMP** 를 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

반환 형식:

TIMESTAMP

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'Asia/Seoul'

```

```
-- it returns the current date and time in TIMESTAMP type of session
and server timezones.
```

```
SELECT LOCALTIME, SYS_TIMESTAMP;
```

```

CURRENT_TIMESTAMP          SYS_TIMESTAMP
=====
04:34:16 PM 02/05/2016    04:34:16 PM 02/05/2016

```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'
```

```
SET TIME_ZONE 'America/Los_Angeles';
```

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'America/Los_Angeles'

```

```
-- Note that LOCALTIME() and SYS_TIMESTAMP return different results
```

```
SELECT LOCALTIME, SYS_TIMESTAMP;
```

```

CURRENT_TIMESTAMP          SYS_TIMESTAMP
=====
11:34:37 PM 02/04/2016    04:34:37 PM 02/05/2016

```

⚠ 경고

10.0 이후로 **CURRENT_TIMESTAMP**, **CURRENT_TIMESTAMP ()**, **LOCALTIME**, **LOCALTIME ()**, **LOCALTIMESTAMP** 및 **LOCALTIMESTAMP ()**가 **SYS_TIMESTAMP ()**, **SYSTIMESTAMP** 와 다르다. 9.x 및 그 이전 버전에서는 동일하다.

DATE

DATE(*date*)

DATE 함수는 지정된 인자로부터 날짜 부분을 추출하여 'MM/DD/YYYY' 형식 문자열로 반환한다. 지정 가능한 인자는 **DATE**, **TIMESTAMP**, **DATETIME** 타입이며, 리턴 값은 **VARCHAR** 타입이다

인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜와 시간이 모두 0인 값을 입력한 경우에는 연, 월, 일 값이 모두 0인 문자열을 반환한다.

매개변수:

date – **DATE**, **TIMESTAMP**, **DATETIME** 타입이 지정될 수 있다.

반환 형식:

STRING

```
SELECT DATE('2010-02-27 15:10:23');
```

```
'02/27/2010'
```

```
SELECT DATE(NOW());
```

```
'04/01/2010'
```

```
SELECT DATE('0000-00-00 00:00:00');
```

```
'00/00/0000'
```

DATEDIFF

DATEDIFF(*date1*, *date2*)

DATEDIFF 함수는 주어진 두 개의 인자로부터 날짜 부분을 추출하여 두 값의 차이를 일 단위 정수로 반환한다. 지정 가능한 인자는 **DATE**, **TIMESTAMP**, **DATETIME** 타입이며, 리턴 값의 타입은 **INTEGER** 이다.

입력 인자의 날짜와 시간 값이 모두 0이면 시스템 파라미터 **return_null_on_function_errors**의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors** 가 yes이면 **NULL** 을 반환하고 no이면 에러를 반환하며, 기본값은 **no** 이다.

매개변수:

date1, date2 - 날짜를 포함하는 타입(**DATE**, **TIMESTAMP**, **DATETIME**) 또는 해당 타입의 값을 나타내는 문자열이 지정될 수 있다. 유효하지 않은 문자열이 지정되면 에러를 반환한다.

반환 형식:

INT

```
SELECT DATEDIFF('2010-2-28 23:59:59','2010-03-02');
```

```
-2
```

```
SELECT DATEDIFF('0000-00-00 00:00:00', '2010-2-28 23:59:59');
```

```
ERROR: Conversion error in date format.
```

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

```
SELECT IF(DATEDIFF('2002-03-03 12:00:00 AM','1990-01-01 11:59:59 PM') = DATEDIFF(timestamptz'2002-03-03 12:00:00 AM',timestamptz'1990-01-01 11:59:59 PM'),'ok','nok');
```

```
'ok'
```

DATE_SUB, SUBDATE

DATE_SUB(*date*, *INTERVAL expr unit*)

SUBDATE(*date*, *INTERVAL expr unit*)

SUBDATE(*date*, *days*)

DATE_SUB ()와 **SUBDATE** ()는 동일하며, 특정 **DATE** 값에 대해 뺄셈 또는 덧셈을 실행한다. 리턴 값은 **DATE** 타입 또는 **DATETIME** 타입이다. 연산 결과의 날짜가 해당 월의 마지막 날짜를 초과하면, 해당 월의 말일을 적용하여 유효한 **DATE** 값을 반환한다.

입력 인자의 날짜와 시간 값이 모두 0이면 시스템 파라미터 **return_null_on_function_errors** 의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors** 가 yes이면 **NULL** 을 반환하고 no이면 에러를 반환하며, 기본값은 no 이다.

계산 결과가 '0000-00-00 00:00:00'과 '0001-01-01 00:00:00' 사이이면, 날짜와 시간 값이 모두 0인 **DATE** 또는 **DATETIME** 타입의 값을 반환한다. 그러나 JDBC 프로그램에서는 연결 URL 속성인 zeroDateTimeBehavior의 설정에 따라 동작이 달라진다("API 레퍼런스 > JDBC API > JDBC 프로그래밍 > 연결 설정" 참고).

param date:

DATE, **DATETIME** 또는 **TIMESTAMP** 타입의 연산식이며, 시작 날짜를 의미한다. 만약, '2006-07-00'와 같이 유효하지 않은 **DATE** 값이 지정되면, 에러를 반환한다.

param expr:

시작 날짜로부터 뺄 시간 간격 값(interval value)을 의미하며, **INTERVAL** 키워드 뒤에 음수가 명시되면 시작 날짜로부터 시간 간격 값을 더한다.

param unit:

expr 수식에 명시된 시간 간격 값의 단위를 의미하며, *unit* 값에 대한 *expr* 인자의 값은 `ADDDATE()` 의 표를 참고한다.

rtype:

DATE or DATETIME

```
SELECT SYSDATE, SUBDATE(SYSDATE,INTERVAL 24 HOUR), SUBDATE(SYSDATE,
1);
```

```
03/30/2010 12:00:00.000 AM 03/29/2010 03/29/2010
```

```
-- 인수 < 0 일때, 날짜를 더한다
SELECT SYSDATE, SUBDATE(SYSDATE,INTERVAL -24 HOUR), SUBDATE(SYSDATE,
-1);
```

```
03/30/2010 12:00:00.000 AM 03/31/2010 03/31/2010
```

```
SELECT SUBDATE('0000-00-00 00:00:00', -50);
```

```
ERROR: Conversion error in date format.
```



```
SELECT SUBDATE('0001-01-01 00:00:00', 10);
```

```
'12:00:00.000 AM 00/00/0000'
```

DAY, DAYOFMONTH

DAY(*date*)

DAYOFMONTH(*date*)

DAY 함수와 **DAYOFMONTH** 함수는 동일하며, 지정된 인자로부터 1~31 범위의 일(day)을 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 연, 월, 일이 모두 0인 값을 입력한 경우에는 0을 반환한다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT DAYOFMONTH('2010-09-09');
```

```
9
```

```
SELECT DAY('2010-09-09 19:49:29');
```

```
9
```

```
SELECT DAYOFMONTH('0000-00-00 00:00:00');
```

```
0
```

DAYOFWEEK

DAYOFWEEK(*date*)

DAYOFWEEK 함수는 지정된 인자로부터 1~7 범위의 요일(1: 일요일, 2: 월요일, ..., 7: 토요일)을 반환한다. 요일 인덱스는 ODBC 표준과 같다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

입력 인자의 연, 월, 일이 모두 0이면 시스템 파라미터 **return_null_on_function_errors**의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors**가 yes이면 **NULL**을 반환하고 no이면 에러를 반환하며, 기본값은 no이다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT DAYOFWEEK('2010-09-09');
```

```
5
```

```
SELECT DAYOFWEEK('2010-09-09 19:49:29');
```

```
5
```

```
SELECT DAYOFWEEK('0000-00-00');
```

```
ERROR: Conversion error in date format.
```

DAYOFYEAR

DAYOFYEAR(*date*)

DAYOFYEAR 함수는 지정된 인자로부터 1~366 범위의 일(day of year)을 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

입력 인자의 날짜 값이 모두 0이면 시스템 파라미터 **return_null_on_function_errors** 의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors** 가 yes이면 **NULL** 을 반환하고 no 이면 에러를 반환하며, 기본값은 **no** 이다.

매개변수:

date - 날짜

반환 형식:

INT

```
SELECT DAYOFYEAR('2010-09-09');
```

```
252
```

```
SELECT DAYOFYEAR('2010-09-09 19:49:29');
```

```
252
```

```
SELECT DAYOFYEAR('0000-00-00');
```

```
ERROR: Conversion error in date format.
```

EXTRACT

EXTRACT(*field FROM date-time_argument*)

EXTRACT 연산자는 날짜/시간 값을 반환하는 연산식 *date-time_argument* 중 일부분을 추출하여 **INTEGER** 타입으로 반환한다.

입력 인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜와 시간이 모두 0인 값을 입력한 경우에는 0을 반환한다.

매개변수:

- **field** - 날짜/시간 수식에서 추출할 값을 지정한다. (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, MILLISECOND)

- **date-time_argument** – 날짜/시간 값을 반환하는 연산식이다. 이 연산식의 값은 **TIME**, **DATE**, **TIMESTAMP**, **DATETIME** 타입 중 하나여야 하며, **NULL** 이 지정된 경우에는 **NULL** 값이 반환된다.

반환 형식:

INT

```
SELECT EXTRACT(MONTH FROM DATETIME '2008-12-25 10:30:20.123' );
```

12

```
SELECT EXTRACT(HOUR FROM DATETIME '2008-12-25 10:30:20.123' );
```

10

```
SELECT EXTRACT(MILLISECOND FROM DATETIME '2008-12-25 10:30:20.123' );
```

123

```
SELECT EXTRACT(MONTH FROM '0000-00-00 00:00:00');
```

0

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

```
SELECT EXTRACT (MONTH FROM datetimetz'10/15/1986 5:45:15.135 am Europe/Bucharest');
```

10

```
SELECT EXTRACT (MONTH FROM datetimeltz'10/15/1986 5:45:15.135 am
Europe/Bucharest');
```

```
10
```

```
SELECT EXTRACT (MONTH FROM timestamptz'10/15/1986 5:45:15 am Europe/
Bucharest');
```

```
10
```

```
SELECT EXTRACT (MONTH FROM timestamptz'10/15/1986 5:45:15 am Europe/
Bucharest');
```

```
10
```

FROM_DAYS

FROM_DAYS(N)

FROM_DAYS 함수는 **INTEGER** 타입을 인자로 입력하면 **DATE** 타입의 날짜를 반환한다.

FROM_DAYS 함수는 그레고리력(Gregorian Calendar) 출현(1582년) 이전은 고려하지 않았으므로 1582년 앞의 날짜에 대해서는 사용하지 않는 것을 권장한다.

인자로 0~3,652,424 범위의 정수를 입력할 수 있다. 0~365 범위의 값을 인자로 입력하면 0을 반환한다. 최대값인 3,652,424는 9999년의 마지막 날을 의미한다.

매개변수:

N – 0~3,652,424 범위의 정수

반환 형식:

DATE

```
SELECT FROM_DAYS(719528);
```

```
01/01/1970
```

```
SELECT FROM_DAYS('366');
```

```
01/03/0001
```

```
SELECT FROM_DAYS(3652424);
```

```
12/31/9999
```

```
SELECT FROM_DAYS(0);
```

```
00/00/0000
```

FROM_TZ

FROM_TZ(*datetime*, *timezone_string*)

DATETIME 타입 값에 타임존을 더하여 타임존이 없는 날짜/시간 타입을 타임존이 있는 날짜/시간 타입으로 변환한다. 입력 값 타입은 DATETIME이고 결과 값 타입은 DATETIMETZ이다.

매개변수:

- **datetime** – DATETIME
- **timezone_string** – 타임존명('Asia/Seoul') 또는 오프셋 ('+05:00')을 나타내는 문자열

반환 형식:

DATETIMETZ

```
SELECT FROM_TZ(datetime '10/10/2014 00:00:00 AM', 'Europe/Vienna');
```

```
12:00:00.000 AM 10/10/2014 Europe/Vienna CEST
```

```
SELECT FROM_TZ(datetime '10/10/2014 23:59:59 PM', '+03:25:25');
```

```
11:59:59.000 PM 10/10/2014 +03:25:25
```

타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

! 더 보기

[DBTIMEZONE\(\)](#), [SESSIONTIMEZONE\(\)](#), [NEW_TIME\(\)](#), [TZ_OFFSET\(\)](#)

FROM_UNIXTIME

FROM_UNIXTIME(*unix_timestamp* [, *format*])

FROM_UNIXTIME 함수는 *format* 인자가 명시된 경우 **VARCHAR** 타입으로 해당 형식의 문자열을 반환하며, *format* 인자가 생략될 경우 **TIMESTAMP** 타입의 값을 반환한다. *unix_timestamp* 인자로 UNIX의 타임스탬프에 해당하는 **INTEGER** 타입을 입력한다. 리턴 값은 현재의 타임 존으로 표현된다.

*format*에 입력하는 시간 형식은 [DATE_FORMAT\(\)](#)의 날짜/시간 형식 2를 따른다.

TIMESTAMP와 UNIX 타임스탬프는 일대일 대응 관계가 아니기 때문에 변환할 때 [UNIX_TIMESTAMP\(\)](#) 함수나 **FROM_UNIXTIME** 함수를 사용하면 값의 일부가 유실될 수 있다. 자세한 설명은 [UNIX_TIMESTAMP\(\)](#)를 참고한다.

인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜와 시간이 모두 0인 값을 입력한 경우에는 날짜와 시간 값이 모두 0인 문자열을 반환한다. 그러나 JDBC 프로그램에서는 연결 URL 속성인 `zeroDateTimeBehavior`의 설정에 따라 동작이 달라진다("API 레퍼런스 > JDBC API > JDBC 프로그래밍 > 연결 설정" 참고).

매개변수:

- **unix_timestamp** - 양의 정수
- **format** - 시간 형식. [DATE_FORMAT\(\)](#)의 날짜/시간 형식 2를 따른다.

반환 형식:

STRING, INT

```
SELECT FROM_UNIXTIME(1234567890);
```

```
01:31:30 AM 02/14/2009
```

```
SELECT FROM_UNIXTIME('1000000000');
```

```
04:46:40 AM 09/09/2001
```

```
SELECT FROM_UNIXTIME(1234567890, '%M %Y %W');
```

```
'February 2009 Saturday'
```

```
SELECT FROM_UNIXTIME('1234567890', '%M %Y %W');
```

```
'February 2009 Saturday'
```

```
SELECT FROM_UNIXTIME(0);
```

```
12:00:00 AM 00/00/0000
```

HOUR

HOUR(*time*)

HOUR 함수는 지정된 인자로부터 시(hour) 부분을 추출한 정수를 반환한다. 인자로 **TIME**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

time - 시간

반환 형식:

INT


```
SELECT HOUR('12:34:56');
```

```
12
```

```
SELECT HOUR('2010-01-01 12:34:56');
```

```
12
```

```
SELECT HOUR(datetime'2010-01-01 12:34:56');
```

```
12
```

LAST_DAY

LAST_DAY(*date_argument*)

LAST_DAY 함수는 인자로 지정된 **DATE** 값에서 해당 월의 마지막 날짜 값을 **DATE** 타입으로 반환한다.

입력 인자의 연, 월, 일이 모두 0이면 시스템 파라미터 **return_null_on_function_errors**의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors**가 **yes**이면 **NULL**을 반환하고 **no**이면 에러를 반환하며, 기본값은 **no**이다.

매개변수:

date_argument - **DATE** 타입의 연산식을 지정한다.

TIMESTAMP 나 **DATETIME** 값을 지정하려면 **DATE** 타입으로 명시적 변환을 해야 한다. 값이 **NULL**이면 **NULL**을 반환한다.

반환 형식:

DATE

```
--it returns last day of the month in DATE type
SELECT LAST_DAY(DATE '1980-02-01'), LAST_DAY(DATE '2010-02-01');
```

02/28/1980

02/28/2010

```
--it returns last day of the month when explicitly casted to DATE
type
```

```
SELECT LAST_DAY(CAST (SYS_TIMESTAMP AS DATE)), LAST_DAY(CAST
(SYS_DATETIME AS DATE));
```

02/28/2010

02/28/2010

```
SELECT LAST_DAY('0000-00-00');
```

```
ERROR: Conversion error in date format.
```

MAKEDATE

MAKEDATE(year, dayofyear)

MAKEDATE 함수는 지정된 인자로부터 날짜를 반환한다. 인자로 1~9999 범위의 연도와 일(day of year)에 해당하는 **INTEGER** 타입의 값을 지정할 수 있으며, 1/1/1~12/31/9999 범위의 **DATE** 타입의 값을 반환한다. 일(day of year)이 해당 연도를 넘어가면 다음 연도가 된다. 예를 들어, MAKEDATE(1999, 366)은 2000-01-01을 반환한다. 단, 연도에 0~69 범위의 값을 입력하면 2000년~2069년으로 처리하고, 70~99 범위의 값을 입력하면 1970년~1999년으로 처리한다.

year 와 dayofyear 가 모두 0이면 시스템 파라미터 **return_null_on_function_errors** 의 값에 따라 다른 값을 반환한다. **return_null_on_function_errors** 가 yes이면 **NULL** 을 반환하고 no 이면 에러를 반환하며, 기본값은 no 이다.

매개변수:

- **year** – 1~9999 범위의 연도
- **dayofyear** – 연도에 0~99의 값을 입력하면 예외적으로 처리하므로, 실제로는 100년 이후의 연도만 사용된다. 따라서 dayofyear 의 최대값은 3,615,902이며, MAKEDATE(100, 3615902)는 9999/12/31을 반환한다.

반환 형식:

DATE

```
SELECT MAKEDATE(2010,277);
```

```
10/04/2010
```

```
SELECT MAKEDATE(10,277);
```

```
10/04/2010
```

```
SELECT MAKEDATE(70,277);
```

```
10/04/1970
```

```
SELECT MAKEDATE(100,3615902);
```

```
12/31/9999
```

```
SELECT MAKEDATE(9999,365);
```

```
12/31/9999
```

```
SELECT MAKEDATE(0,0);
```

```
ERROR: Conversion error in date format.
```

MAKETIME

MAKETIME(*hour*, *min*, *sec*)

MAKETIME 함수는 지정된 인자로부터 시간을 AM/PM 형태로 반환한다. 인자로 시각, 분, 초에 해당하는 **INTEGER** 타입의 값을 지정할 수 있으며, **DATETIME** 타입의 값을 반환한다.

매개변수:

- **hour** – 시를 나타내는 0~23 범위의 정수
- **min** – 분을 나타내는 0~59 범위의 정수
- **sec** – 초를 나타내는 0~59 범위의 정수

반환 형식:

DATETIME

```
SELECT MAKETIME(13,34,4);
```

```
01:34:04 PM
```

```
SELECT MAKETIME('1','34','4');
```

```
01:34:04 AM
```

```
SELECT MAKETIME(24,0,0);
```

```
ERROR: Conversion error in time format.
```

MINUTE

MINUTE(*time*)

MINUTE 함수는 지정된 인자로부터 0~59 범위의 분(minute)을 반환한다. 인자로 **TIME**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

time – 시간

반환 형식:

INT

```
SELECT MINUTE('12:34:56');
```

34

```
SELECT MINUTE('2010-01-01 12:34:56');
```

34

```
SELECT MINUTE('2010-01-01 12:34:56.7890');
```

34

MONTH

MONTH(*date*)

MONTH 함수는 지정된 인자로부터 1~12 범위의 월(month)을 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜가 모두 0인 값을 입력한 경우에는 0을 반환한다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT MONTH('2010-01-02');
```

1

```
SELECT MONTH('2010-01-02 12:34:56');
```

```
1
```

```
SELECT MONTH('2010-01-02 12:34:56.7890');
```

```
1
```

```
SELECT MONTH('0000-00-00');
```

```
0
```

MONTHS_BETWEEN

MONTHS_BETWEEN(*date_argument*, *date_argument*)

MONTHS_BETWEEN 함수는 주어진 두 개의 **DATE** 값 간의 차이를 월 단위로 반환하며, 리턴 값은 **DOUBLE** 타입이다. 인자로 지정된 두 날짜가 동일하거나, 해당 월의 말일인 경우에는 정수 값을 반환하지만, 그 외의 경우에는 날짜 차이를 31로 나눈 값을 반환한다.

매개변수:

date_argument - **DATE** 타입의 값을 가지는 연산식을 지정한다. **TIMESTAMP**나 **DATETIME** 값도 지정할 수 있다. 값이 **NULL**이면 **NULL** 을 반환한다.

반환 형식:

DOUBLE

```
--첫번째 인수가 이전 날짜일 때 음수의 달수를 반환한다
SELECT MONTHS_BETWEEN(DATE '2008-12-31', DATE '2010-6-30');
```

```
-1.8000000000000000e+001
```

```
--각각의 날짜가 그 달의 마지막 날일 때 정수를 반환한다
SELECT MONTHS_BETWEEN(DATE '2010-6-30', DATE '2008-12-31');
```

```
1.8000000000000000e+001
```

```
--두 개의 인수가 명시적으로 DATE 타입으로 형 변환된 경우는 달 수를 반환한다
SELECT MONTHS_BETWEEN(CAST (SYS_TIMESTAMP AS DATE), DATE
'2008-12-25');
```

```
1.332258064516129e+001
```

```
--두 개의 인수가 명시적으로 DATE 타입으로 형 변환된 경우는 달 수를 반환한다
SELECT MONTHS_BETWEEN(CAST (SYS_DATETIME AS DATE), DATE
'2008-12-25');
```

```
1.332258064516129e+001
```

다음은 타임존 타입의 값을 이용하는 예제이다. 타임존 관련된 기술에 대해서는 다음을 참조하라, 타임존이 있는 날짜/시간 데이터 타입.

```
SELECT MONTHS_BETWEEN(datetimetz'2001-10-11 10:11:12 +02:00',
datetimetz'2001-05-11 10:11:12 +03:00');
```

```
5.0000000000000000e+00
```

NEW_TIME

NEW_TIME(*src_datetime*, *src_timezone*, *dst_timezone*)

어떤 타임존에서 다른 타임존으로 날짜 값을 이동한다. *src_datetime*에는 DATETIME 또는 TIME 타입의 값을 입력하며, 반환 값의 타입은 *src_datetime*과 동일하다.

매개변수:

- **src_datetime** – DATETIME 또는 TIME 타입의 입력 값
- **src_timezone** – 원본 타임존의 지역 이름
- **dst_timezone** – 대상 타임존의 지역 이름

반환 형식:

src_datetime의 타입과 동일한 타입

```
SELECT NEW_TIME(datetime '10/10/2014 10:10:10 AM', 'Europe/Vienna',
'Europe/Bucharest');
```

```
11:10:10.000 AM 10/10/2014
```

TIME 타입의 경우 타임존 인자로 오프셋만 허용하며, 지역 이름은 허용하지 않는다.

```
SELECT NEW_TIME(time '10:10:10 PM', '+03:00', '+10:00');
```

```
05:10:10 AM
```

타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

❗ 더 보기

DBTIMEZONE(), SESSIONTIMEZONE(), FROM_TZ(), TZ_OFFSET()

QUARTER

QUARTER(*date*)

QUARTER 함수는 지정된 인자로부터 1~4 범위의 분기(quarter)를 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT QUARTER('2010-05-05');
```

```
2
```

```
SELECT QUARTER('2010-05-05 12:34:56');
```

```
2
```

```
SELECT QUARTER('2010-05-05 12:34:56.7890');
```

```
2
```

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

```
SELECT QUARTER('2008-04-01 01:02:03 Asia/Seoul');
```

```
2
```

```
SELECT QUARTER(datetimetz'2003-12-31 01:02:03.1234 Europe/Paris');
```

```
4
```

ROUND**ROUND**(*date*, *fmt*)

fmt 에서 지정한 단위의 값으로 반올림한다. DATE 타입의 값을 반환한다.

매개변수:

- **date** – DATE 타입, **TIMESTAMP** 타입 또는 **DATE** 타입의 값.
- **fmt** – 반올림할 단위에 대한 포맷을 지정. 생략되면 “dd”

반환 형식:

DATE

포맷, 단위, 함수의 반환 값은 다음과 같다.

포맷	단위	반환 값
‘yyyy’ 또는 ‘yy’	년도	년도로 반올림한 값
‘mm’ 또는 ‘month’	월	월로 반올림한 값
‘q’	사분기	사분기로 반올림하여 1/1, 4/1, 7/1, 10/1 중 하나의 날짜를 가지는 값
‘day’	주	<i>date</i> 가 있는 주의 시작 또는 다음 주에 해당하는 일요일로 반올림한 값
‘dd’	일	일로 반올림한 값
‘hh’	시	시로 반올림한 값
‘mi’	분	분으로 반올림한 값
‘ss’	초	초로 반올림한 값

```
SELECT ROUND(date'2012-10-26', 'yyyy');
```

```
01/01/2013
```

```
SELECT ROUND(timestamp'2012-10-26 12:10:10', 'mm');
```

```
11/01/2012
```

```
SELECT ROUND(datetime'2012-12-26 12:10:10', 'dd');
```

```
12/27/2012
```

```
SELECT ROUND(datetime'2012-12-26 12:10:10', 'day');
```

```
12/30/2012
```

```
SELECT ROUND(datetime'2012-08-26 12:10:10', 'q');
```

```
10/01/2012
```

```
SELECT TRUNC(datetime'2012-08-26 12:10:10', 'q');
```

```
07/01/2012
```

```
SELECT ROUND(datetime'2012-02-28 23:10:00', 'hh');
```

```
02/28/2012
```

```
SELECT ROUND(datetime'2012-02-28 23:58:59', 'hh');
```

```
02/29/2012
```

```
SELECT ROUND(datetime'2012-02-28 23:59:59', 'mi');
```

```
02/29/2012
```

```
SELECT ROUND(datetime'2012-02-28 23:59:59.500', 'ss');
```

```
02/29/2012
```

반올림이 아니라 절삭하기 위해서는 `TRUNC(date, fmt)` 함수를 사용하면 된다.

SEC_TO_TIME

`SEC_TO_TIME(second)`

SEC_TO_TIME 함수는 지정된 인자로부터 시, 분, 초를 포함한 시간을 반환한다. 인자로 0~86399 범위의 **INTEGER** 타입의 값을 지정할 수 있으며, **TIME** 타입의 값을 반환한다.

매개변수:

second – 0~86399 범위의 초

반환 형식:

TIME

```
SELECT SEC_TO_TIME(82800);
```

```
sec_to_time(82800)
=====
11:00:00 PM
```

```
SELECT SEC_TO_TIME('82800.3');
```

```
sec_to_time('82800.3')
=====
11:00:00 PM
```

```
SELECT SEC_TO_TIME(86399);
```

```
sec_to_time(86399)
=====
11:59:59 PM
```

SECOND

SECOND(*time*)

SECOND 함수는 지정된 인자로부터 0~59 범위의 초(second)를 반환한다. 인자로 **TIME**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

time – 시간

반환 형식:

INT

```
SELECT SECOND('12:34:56');
```

```
second('12:34:56')
=====
56
```

```
SELECT SECOND('2010-01-01 12:34:56');
```

```
second('2010-01-01 12:34:56')
=====
56
```

```
SELECT SECOND('2010-01-01 12:34:56.7890');
```

```
second('2010-01-01 12:34:56.7890')
=====
56
```

SYS_DATE, SYSDATE

SYS_DATE

SYSDATE

SYS_DATE 및 **SYSDATE** 는 서로 바꿔 사용할 수 있으며 서버의 현재 날짜를 **DATE** 타입(MM/DD/YYYY 또는 YYYY-MM-DD)으로 반환한다. 산술 연산의 단위는 일(day)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 `CURDATE()`, `CURRENT_DATE()` 및 `CURRENT_DATE` 와 동일하다. 차이점은 `CURDATE()`, `CURRENT_DATE()` 를 참고하고, 함수에 대한 자세한 내용은 `DBTIMEZONE()`, `SESSIONTIMEZONE()` 을 참고한다.

입력 인자의 년, 월, 일 값이 모두 0이면 **return_null_on_function_errors** 시스템 파라미터에 의해 반환 값이 결정된다. 이 파라미터가 yes이면 **NULL** 을 반환하고 no이면 오류를 반환한다. 기본값은 **no** 이다.

반환 형식:

DATE

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```
dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'Asia/Seoul'
```

```
-- it returns the current date in DATE type
```

```
SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE, SYS_DATE, SYSDATE;
```

```

CURRENT_DATE    CURRENT_DATE    CURRENT_DATE    SYS_DATE
SYS_DATE
=====
===
02/05/2016      02/05/2016      02/05/2016      02/05/2016  02/05/
2016
```

```
-- it returns the date 60 days added to the current date
```

```
SELECT CURDATE()+60;
```

```

    CURRENT_DATE +60
=====
04/05/2016

```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'
```

```
SET TIME_ZONE 'America/Los_Angeles';
```

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

    dbtimezone          sessiontimezone
=====
'Asia/Seoul'          'America/Los_Angeles'

```

```
-- Note that CURDATE() and SYS_DATE returns different results
```

```
SELECT CURDATE(), CURRENT_DATE(), CURRENT_DATE, SYS_DATE, SYSDATE;
```

```

    CURRENT_DATE    CURRENT_DATE    CURRENT_DATE    SYS_DATE
SYS_DATE
=====
02/04/2016        02/04/2016        02/04/2016        02/05/2016  02/05/
2016

```

⚠ 경고

10.0 이후로 **SYS_DATE** 및 **SYSDATE**가 **CURDATE ()**, **CURRENT_DATE**, **CURRENT_DATE ()**와 다르다. 9.x 및 그 이전 버전에서는 동일하다.

SYS_DATETIME, SYSDATETIME

SYS_DATETIME

SYSDATETIME

SYS_DATETIME 및 **SYSDATETIME** 은 서로 바꿔 사용할 수 있으며 서버의 현재 날짜와 시간을 **DATETIME** 타입으로 반환한다. 산술연산의 단위는 밀리초(msec)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **CURRENT_DATETIME()**, **CURRENT_DATETIME**, **NOW()** 와 동일하다. 차

이점은 `CURRENT_DATETIME()`, `NOW()` 를 참고하고, 함수에 대한 자세한 내용은 `DBTIMEZONE()`, `SESSIONTIMEZONE()` 을 참고한다.

반환 형식:

DATETIME

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'Asia/Seoul'
```

-- it returns the current date and time in DATETIME type

```
SELECT NOW(), SYS_DATETIME;
```

```

CURRENT_DATETIME          SYS_DATETIME
=====
04:05:09.292 PM 02/05/2016  04:05:09.292 PM 02/05/2016
```

-- it returns the timestamp value 1 hour added to the current sys_datetime value

```
SELECT TO_CHAR(SYSDATETIME+3600*1000, 'YYYY-MM-DD HH24:MI');
```

```

to_char( SYS_DATETIME +3600*1000, 'YYYY-MM-DD HH24:MI')
=====
'2016-02-05 17:05'
```

-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'

```
SET TIME_ZONE 'America/Los_Angeles';
```

```
SELECT DBTIMEZONE(), SESSIONTIMEZONE();
```

```

dbtimezone          sessiontimezone
=====
'Asia/Seoul'        'America/Los_Angeles'
```

-- Note that NOW() and SYS_DATETIME return different results

```
SELECT NOW(), SYS_DATETIME;
```

```

CURRENT_DATETIME          SYS_DATETIME
```



```
=====
11:08:57.041 PM 02/04/2016      04:08:57.041 PM 02/05/2016
```

⚠ 경고

10.0 이후로 **SYS_DATEIME**, **SYSDATETIME**이 **CURRENT_DATETIME ()**, **NOW ()**와 다르다. 9.x 및 그 이전 버전에서는 동일하다.

SYS_TIME, SYSTIME

SYS_TIME

SYSTIME

SYS_TIME 및 **SYSTIME** 은 서로 바꿔 사용할 수 있으며 서버의 현재 시간을 **TIME** 타입 (HH:MI:SS)으로 반환한다. 산술연산의 단위는 초(sec)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **CURTIME()**, **CURRENT_TIME**, **CURRENT_TIME()** 과 동일하다. 차이점은 **CURTIME()**, **CURRENT_TIME()** 을 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

반환 형식:

TIME

```
select dbtimezone(), sessiontimezone();
```

```
dbtimezone          sessiontimezone
```

```
=====
'Asia/Seoul'        'Asia/Seoul'
```

```
-- it returns the current time in TIME type
```

```
SELECT CURTIME(), CURRENT_TIME(), CURRENT_TIME, SYS_TIME, SYSTIME;
```

```
    CURRENT_TIME    CURRENT_TIME    CURRENT_TIME    SYS_TIME
SYS_TIME
```

```
=====
=====
    04:22:54 PM    04:22:54 PM    04:22:54 PM    04:22:54 PM    04:22:
54 PM
```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'
```

```

SET TIME_ZONE 'America/Los_Angeles';

select dbtimezone(), sessiontimezone();

    dbtimezone          sessiontimezone
=====
    'Asia/Seoul'        'America/Los_Angeles'

-- Note that CURTIME() and SYS_TIME return different results

SELECT CURTIME(), CURRENT_TIME(), CURRENT_TIME, SYS_TIME, SYSTIME;

    CURRENT_TIME    CURRENT_TIME    CURRENT_TIME    SYS_TIME
SYS_TIME
=====
=====
    11:23:16 PM    11:23:16 PM    11:23:16 PM    04:23:16 PM    04:23:
16 PM

```

⚠ 경고

10.0 이후로 **SYS_TIME**, **SYSTIME**이 **CURTIME ()**, **CURRENT_TIME ()**과 다르다. 9.x 및 그 이전 버전에서는 동일하다.

SYS_TIMESTAMP, SYSTIMESTAMP

SYS_TIMESTAMP

SYSTIMESTAMP

SYS_TIMESTAMP 및 **SYSTIMESTAMP**은 서로 바꿔 사용할 수 있으며 서버의 현재 날짜와 시간을 **TIMESTAMP** 타입으로 반환한다. 산술연산의 단위는 초(sec)이다. 현재 세션의 타임존이 서버의 타임존과 동일하면 함수는 **CURRENT_TIMESTAMP**, **CURRENT_TIMESTAMP()**, **LOCALTIME**, **LOCALTIME()**, **LOCALTIMESTAMP**, **LOCALTIMESTAMP()** 와 동일하다. 차이점은 **CURRENT_TIMESTAMP**, **CURRENT_TIMESTAMP()**, **LOCALTIME**, **LOCALTIME()**, **LOCALTIMESTAMP**, **LOCALTIMESTAMP()** 를 참고하고, 함수에 대한 자세한 내용은 **DBTIMEZONE()**, **SESSIONTIMEZONE()** 을 참고한다.

반환 형식:

TIMESTAMP

```

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

    dbtimezone          sessiontimezone

```

```
=====
'Asia/Seoul'          'Asia/Seoul'

-- it returns the current date and time in TIMESTAMP type of session
and server timezones.

SELECT LOCALTIME, SYS_TIMESTAMP;

CURRENT_TIMESTAMP      SYS_TIMESTAMP
=====
04:34:16 PM 02/05/2016    04:34:16 PM 02/05/2016
```

```
-- change session time from 'Asia/Seoul' to 'America/Los_Angeles'

SET TIME_ZONE 'America/Los_Angeles';

SELECT DBTIMEZONE(), SESSIONTIMEZONE();

dbtimezone      sessiontimezone
=====
'Asia/Seoul'    'America/Los_Angeles'

-- Note that LOCALTIME() and SYS_TIMESTAMP return different results

SELECT LOCALTIME, SYS_TIMESTAMP;

CURRENT_TIMESTAMP      SYS_TIMESTAMP
=====
11:34:37 PM 02/04/2016    04:34:37 PM 02/05/2016
```

⚠ 경고

10.0 이후로 **SYS_TIMESTAMP ()**, **SYSTIMESTAMP**가 **CURRENT_TIMESTAMP**, **CURRENT_TIMESTAMP ()**, **LOCALTIME**, **LOCALTIME ()**, **LOCALTIMESTAMP** 및 **LOCALTIMESTAMP ()**와 다르다. 9.x 및 그 이전 버전에서는 동일하다.

TIME

TIME(*time*)

TIME 함수는 지정된 인자로부터 시간 부분을 추출하여 'HH:MI:SS' 형태의 **VARCHAR** 타입 문자열을 반환한다. 인자로 **TIME**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있다.

매개변수:

time – 시간

반환 형식:

STRING

```
SELECT TIME('12:34:56');
```

```
time('12:34:56')
=====
'12:34:56'
```

```
SELECT TIME('2010-01-01 12:34:56');
```

```
time('2010-01-01 12:34:56')
=====
'12:34:56'
```

```
SELECT TIME(datetime'2010-01-01 12:34:56');
```

```
time(datetime '2010-01-01 12:34:56')
=====
'12:34:56'
```

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

```
SELECT TIME(datetimetz'1996-02-03 02:03:04 AM America/Lima PET');
```

```
'02:03:04'
```

```
SELECT TIME(datetimeltz'1996-02-03 02:03:04 AM America/Lima PET');
```

```
'16:03:04'
```

```
SELECT TIME(datetimeltz'2000-12-31 17:34:23.1234 -05:00');
```

```
'07:34:23.123'
```

```
SELECT TIME(datetimeetz'2000-12-31 17:34:23.1234 -05:00');
```

```
'17:34:23.123'
```

TIME_TO_SEC

TIME_TO_SEC(*time*)

TIME_TO_SEC 함수는 지정된 인자로부터 0~86399 범위의 초를 반환한다. 인자로 **TIME**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

time – 시간

반환 형식:

INT

```
SELECT TIME_TO_SEC('23:00:00');
```

```
82800
```

```
SELECT TIME_TO_SEC('2010-10-04 23:00:00');
```

```
82800
```

```
SELECT TIME_TO_SEC('2010-10-04 23:00:00.1234');
```

```
82800
```

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 **타임존이 있는 날짜/시간 데이터 타입**을 참고한다.

```
SELECT TIME_TO_SEC(datetimeltz'1996-02-03 02:03:04 AM America/Lima
PET');
```

```
57784
```

```
SELECT TIME_TO_SEC(datetimetz'1996-02-03 02:03:04 AM America/Lima
PET');
```

```
7384
```

TIMEDIFF

TIMEDIFF(*expr1*, *expr2*)

TIMEDIFF 함수는 지정된 두 개의 시간 인자의 시간 차를 반환한다. 날짜/시간 타입인 **TIME**, **DATE**, **TIMESTAMP**, **DATETIME** 타입을 인자로 입력할 수 있으며, 두 인자의 데이터 타입은 같아야 한다. **TIME** 타입을 반환하며, 따라서 두 인자의 시간 차이는 00:00:00~23:59:59 범위여야 한다. 이 범위를 벗어나면 에러를 반환한다.

매개변수:

expr2 (*expr1*,) – 시간. 두 인자의 데이터 타입은 같아야 한다.

반환 형식:

TIME

```
SELECT TIMEDIFF(time '17:18:19', time '12:05:52');
```

```
05:12:27 AM
```

```
SELECT TIMEDIFF('17:18:19', '12:05:52');
```

```
05:12:27 AM
```

```
SELECT TIMEDIFF('2010-01-01 06:53:45', '2010-01-01 03:04:05');
```

```
03:49:40 AM
```

다음은 타임존 타입의 값을 사용하는 예이다. 타임존과 관련된 설명은 **타임존이 있는 날짜/시간 데이터 타입**을 참고한다.

```
SELECT TIMEDIFF (datetimeltz'2013-10-28 03:11:12 AM Asia/Seoul',  
datetimeltz'2013-10-27 04:11:12 AM Asia/Seoul');
```

```
11:00:00 PM
```

TIMESTAMP

TIMESTAMP(*date*[, *time*])

TIMESTAMP 함수는 인자로 날짜/시간 형식의 문자열이 지정되고, 이를 **DATETIME** 타입으로 반환한다.

단일 인자로 **DATE** 형식 문자열('YYYY-MM-DD' 또는 'MM/DD/YYYY') 또는 **TIMESTAMP** 형식 문자열('YYYY-MM-DD HH:MI:SS' 또는 'HH:MI:SS MM/DD/YYYY')이 지정되면 이를 **DATETIME** 타입으로 반환한다.

두 번째 인자로 **TIME** 형식 문자열('HH:MI:SS.*FF*')이 주어진다면 이를 첫 번째 인자 값에 더한 결과를 **DATETIME** 타입으로 반환한다. 두 번째 인자가 명시되지 않으면, 기본값으로 **12:00:00.000 AM** 이 설정된다.

매개변수:

- **date** - 'YYYY-MM-DD', 'MM/DD/YYYY', 'YYYY-MM-DD HH:MI:SS.*FF*', 'HH:MI:SS.*FF* MM/DD/YYYY' 형식 문자열이 지정될 수 있다.
- **time** - 'HH:MI:SS*[*FF]' 형식 문자열이 지정될 수 있다.

반환 형식:

DATETIME

```
SELECT TIMESTAMP('2009-12-31'), TIMESTAMP('2009-12-31', '12:00:00');
```

```
12:00:00.000 AM 12/31/2009      12:00:00.000 PM 12/31/2009
```

```
SELECT TIMESTAMP('2010-12-31 12:00:00','12:00:00');
```

```
12:00:00.000 AM 01/01/2011
```

```
SELECT TIMESTAMP('13:10:30 12/25/2008');
```

```
01:10:30.000 PM 12/25/2008
```

다음은 타임존 타입의 값에 대한 예이다. 타임존에 관련된 기술은 다음을 참조하라, see [타임존이 있는 날짜/시간 데이터 타입](#).

```
SELECT TIMESTAMP(datetimetz'2010-12-31 12:00:00 America/New_York',
'03:00');
```

```
03:00:00.000 PM 12/31/2010
```

```
SELECT TIMESTAMP(datetimeltz'2010-12-31 12:00:00 America/New_York',
'03:00');
```

```
05:00:00.000 AM 01/01/2011
```

TO_DAYS

TO_DAYS(*date*)

TO_DAYS 함수는 지정된 인자로부터 0년 이후의 날 수를 366~3652424 범위의 값으로 반환한다. 인자로 **DATE** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

TO_DAYS 함수는 그레고리력(Gregorian Calendar) 출현(1582년)보다 앞의 날짜는 고려하지 않았으므로, 1582년보다 앞의 날짜에 대해서는 사용하지 않는 것을 권장한다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT TO_DAYS('2010-10-04');
```

```
to_days('2010-10-04')
=====
734414
```

```
SELECT TO_DAYS('2010-10-04 12:34:56');
```

```
to_days('2010-10-04 12:34:56')
=====
734414
```

```
SELECT TO_DAYS('2010-10-04 12:34:56.7890');
```

```
to_days('2010-10-04 12:34:56.7890')
=====
734414
```

```
SELECT TO_DAYS('1-1-1');
```

```
to_days('1-1-1')
=====
366
```

```
SELECT TO_DAYS('9999-12-31');
```

```
to_days('9999-12-31')
=====
3652424
```

TRUNC

TRUNC(*date*[, *fmt*])*fmt* 에서 지정한 단위 아래의 값을 절삭한다. DATE 타입의 값을 반환한다.

매개변수:

- **date** – DATE 타입, **TIMESTAMP** 타입 또는 **DATETIME** 타입의 값
- **fmt** – 절삭할 단위에 대한 포맷을 지정. 생략되면 “dd”

반환 형식:

DATE

포맷, 단위, 함수의 반환 값은 다음과 같다.

포맷	단위	함수의 반환 값
'yyyy' 또는 'yy'	년도	같은 년도 1월 1일
'mm' 또는 'month'	월	같은 월의 1일
'q'	사분기	사분기의 첫째날 (1/1, 4/1, 7/1, 10/1)
'day'	주	<i>date</i> 가 있는 주의 시작에 해당하는 일요일
'dd'	일	입력 값과 같은 날짜

```
SELECT TRUNC(date'2012-12-26', 'yyyy');
```

```
01/01/2012
```

```
SELECT TRUNC(timestamp'2012-12-26 12:10:10', 'mm');
```

```
12/01/2012
```

```
SELECT TRUNC(datetime'2012-12-26 12:10:10', 'q');
```

```
10/01/2012
```

```
SELECT TRUNC(datetime'2012-12-26 12:10:10', 'dd');
```

```
12/26/2012
```

```
-- It returns the date of Sunday of the week which includes
date'2012-12-26'
SELECT TRUNC(datetime'2012-12-26 12:10:10', 'day');
```

```
12/23/2012
```

절삭이 아니라 반올림하기 위해서는 `ROUND(date, fmt)` 함수를 사용하면 된다.

TZ_OFFSET

TZ_OFFSET(*timezone_string*)

타임존 오프셋 또는 타임존 지역 이름(예: '-05:00' 또는 'Europe/Vienna')으로부터 타임존 오프셋을 반환한다.

매개변수:

timezone_string - 타임존 오프셋 또는 타임존 지역 이름

반환 형식:

STRING

```
SELECT TZ_OFFSET(' +05:00');
```

```
' +05:00'
```

```
SELECT TZ_OFFSET('Asia/Seoul');
```

```
'+09:00'
```

타임존과 관련된 설명은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다.

! 더 보기

DBTIMEZONE(), SESSIONTIMEZONE(), FROM_TZ(), NEW_TIME()

UNIX_TIMESTAMP

UNIX_TIMESTAMP(*[date]*)

UNIX_TIMESTAMP 함수는 인자를 생략할 수 있으며, 인자를 생략하면 '1970-01-01 00:00:00' UTC 이후 현재 시스템 날짜/시간까지의 초 단위 시간 간격(interval)을 반환한다. *date* 인자가 지정되면 '1970-01-01 00:00:00' UTC 이후 지정된 날짜/시간까지의 초 단위 시간 간격을 반환한다.

년, 월, 일에 해당하는 인자 값에는 0이 허용되지 않지만 날짜 및 시간에 해당하는 모든 인자 값에 0을 입력한 경우에는 예외적으로 0이 반환된다.

DATETIME 타입의 인자는 세션 타임존에서 고려한다.

매개변수:

date - **DATE** 타입, **TIMESTAMP** 타입, **TIMESTAMP****TZ** 타입, **TIMESTAMP****PLTZ** 타입, **DATETIME** 타입, **DATETIME****TZ** 타입, **DATETIME****ELTZ** 타입, **DATE** 형식 문자열('YYYY-MM-DD' 또는 'MM/DD/YYYY'), **TIMESTAMP** 형식 문자열('YYYY-MM-DD HH:MI:SS', 'HH:MI:SS MM/DD/YYYY') 또는 'YYYYMMDD' 형식 문자열을 지정할 수 있다.

반환 형식:

INT

```
SELECT UNIX_TIMESTAMP('1970-01-02'), UNIX_TIMESTAMP();
```

```

    unix_timestamp('1970-01-02')    unix_timestamp()
=====
                                54000                1270196737

```

```
SELECT UNIX_TIMESTAMP ('0000-00-00 00:00:00');
```

```

    unix_timestamp('0000-00-00 00:00:00')
=====
                                0

```

```
-- when used without argument, it returns the exact value at the
moment of execution of each occurrence
```

```
SELECT UNIX_TIMESTAMP(), SLEEP(1), UNIX_TIMESTAMP();
```

```

    unix_timestamp()    sleep(1)    unix_timestamp()
=====
    1454661297          0            1454661298

```

UTC_DATE

UTC_DATE()

UTC_DATE 함수는 UTC 날짜를 'YYYY-MM-DD' 형태로 반환한다.

반환 형식:

STRING

```
SELECT UTC_DATE();
```

```

    utc_date()
=====
    01/12/2011

```

UTC_TIME

UTC_TIME()

UTC_TIME 함수는 UTC 시간을 'HH:MM:SS' 형태로 반환한다.

반환 형식:

STRING

```
SELECT UTC_TIME();
```

```
utc_time()
=====
10:35:52 AM
```

WEEK

WEEK(*date*[, *mode*])

WEEK 함수는 지정된 인자로부터 0~53 범위의 주를 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

- **date** – 날짜
- **mode** – 0~7 범위의 값

반환 형식:

INT

함수의 두 번째 인자인 *mode* 는 생략할 수 있으며, 0~7 범위의 값을 입력한다. 이 값으로 한 주가 일요일부터 시작하는지 월요일부터 시작하는지, 리턴 값의 범위가 0~53인지 1~53인지 설정한다. *mode* 를 생략하면 시스템 파라미터 **default_week_format** 의 값(기본값: 0)이 사용된다. *mode* 값의 의미는 다음과 같다.

mode	시작 요일	범위	해당 연도의 첫 번째 주
0	일요일	0~53	일요일이 해당 연도에 속하는 첫 번째 주
1	월요일	0~53	

mode	시작 요일	범위	해당 연도의 첫 번째 주
			3일 이상이 해당 연도에 속하는 첫 번째 주
2	일요일	1~53	일요일이 해당 연도에 속하는 첫 번째 주
3	월요일	1~53	3일 이상이 해당 연도에 속하는 첫 번째 주
4	일요일	0~53	3일 이상이 해당 연도에 속하는 첫 번째 주
5	월요일	0~53	월요일이 해당 연도에 속하는 첫 번째 주
6	일요일	1~53	3일 이상이 해당 연도에 속하는 첫 번째 주
7	월요일	1~53	월요일이 해당 연도에 속하는 첫 번째 주

mode 값이 0, 1, 4, 5 중 하나이고 해당 날짜보다 앞선 연도의 마지막 주에 해당하면 **WEEK** 함수는 0을 반환한다. 이때의 목적은 해당 연도에서 해당 주가 몇 번째 주인지를 아는 것이므로, 1999년의 52번째 주에 해당해도 2000년의 날짜가 0번째 주에 해당되는 0을 반환한다.

```
SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
```

```
year('2000-01-01')    week('2000-01-01', 0)
=====
                2000                0
```

시작 요일이 속해있는 주의 연도를 기준으로 해당 날짜가 몇 번째 주인지 알려면, *mode* 값으로 0, 2, 5, 7 중 하나의 값을 사용한다.

```
SELECT WEEK('2000-01-01',2);
```

```
week('2000-01-01', 2)
=====
52
```

```
SELECT WEEK('2010-04-05');
```

```
week('2010-04-05', 0)
=====
14
```

```
SELECT WEEK('2010-04-05 12:34:56',2);
```

```
week('2010-04-05 12:34:56',2)
=====
14
```

```
SELECT WEEK('2010-04-05 12:34:56.7890',4);
```

```
week('2010-04-05 12:34:56.7890',4)
=====
14
```

WEEKDAY

WEEKDAY(*date*)

WEEKDAY 함수는 지정된 인자로부터 0~6 범위의 요일(0: 월요일, 1: 화요일, ..., 6: 일요일)을 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

date – 날짜

반환 형식:

INT

```
SELECT WEEKDAY('2010-09-09');
```

```
weekday('2010-09-09')
=====
3
```

```
SELECT WEEKDAY('2010-09-09 13:16:00');
```

```
weekday('2010-09-09 13:16:00')
=====
3
```

YEAR

YEAR(*date*)

YEAR 함수는 지정된 인자로부터 1~9999 범위의 연도를 반환한다. 인자로 **DATE**, **TIMESTAMP**, **DATETIME** 타입의 값을 지정할 수 있으며, **INTEGER** 타입의 값을 반환한다.

매개변수:

date - 날짜

반환 형식:

INT

```
SELECT YEAR('2010-10-04');
```

```
year('2010-10-04')
=====
2010
```

```
SELECT YEAR('2010-10-04 12:34:56');
```

```
year('2010-10-04 12:34:56')
=====
2010
```

```
SELECT YEAR('2010-10-04 12:34:56.7890');
```

```
year('2010-10-04 12:34:56.7890')
=====
2010
```

JSON 함수

JSON 함수 소개

이 섹션에서는 JSON 데이터 관련 동작에 대해 기술한다. 지원하는 JSON 함수들은 다음 표와 같다:

->	JSON_CONTAINS_PATH	JSON_MERGE_PATCH	JSON_REPLACE

->>	JSON_DEPTH	JSON_MERGE_PRESE RVE	JSON_SEARCH

JSON_ARRAY	JSON_EXTRACT	JSON_OBJECT	JSON_SET

JSON_ARRAYAGG	JSON_INSERT	JSON_OBJECTAGG	JSON_TABLE

JSON_ARRAY_APPEND	JSON_KEYS	JSON_PRETTY	JSON_TYPE

JSON_ARRAY_INSERT	JSON_LENGTH	JSON_QUOTE	JSON_UNQUOTE

JSON_CONTAINS	JSON_MERGE	JSON_REMOVE	JSON_VALID

함수의 입력 인자는 아래와 같은 몇가지 유형을 가진다.

- *json_doc*: JSON이나 JSON으로 파싱되는 문자열

- *val*: JSON이나 JSON 지원 스칼라 타입 중 하나로 해석될 수 있는 값
- *json key*: 키 이름으로서의 문자열
- *json path/pointer*: **JSON 경로**와 **JSON 포인터**에 설명된 규칙을 따르는 문자열

참고

JSON 함수 문자열 인자의 코드셋은 UTF8을 기준으로 한다. 다른 코드셋의 입력 문자열은 UTF8로 변환된다. UTF8이 아닌 코드셋 문자열에 대한 대소문자 구별 없는 검색은 기대와 다른 결과가 나올 수 있다.

다음의 표는 입력 인자를 해석하는데 있어서 *json_doc*와 *val*의 차이를 보여주고 있다:

입력 타입	<i>json_doc</i>	<i>val</i>
JSON	입력 값이 변하지 않음	입력 값이 변하지 않음
String	JSON 입력 값이 파싱됨	입력 값이 JSON STRING으로 변환됨
Short, Integer	변환 오류	입력 값이 JSON INTEGER로 변환됨
Bigint	변환 오류	입력 값이 JSON BIGINT로 변환됨
Float, Double,	변환 오류	입력 값이 JSON DOUBLE로 변환됨
NULL	NULL	입력 값이 JSON_NULL로 변환됨
Other	변환 오류	변환 오류

JSON_ARRAY

JSON_ARRAY([val1 [, val2] ...])

JSON_ARRAY 함수는 해당 값들(val, val2, ...)을 가진 리스트(텅빈 리스트도 가능)가 포함된 json 배열을 반환한다.

```
SELECT JSON_ARRAY();
```

```
json_array()  
=====  
[]
```

```
SELECT JSON_ARRAY(1, '1', json '{"a":4}', json '[1,2,3]');
```

```
json_array(1, '1', json '{"a":4}', json '[1,2,3]')  
=====  
[1,"1",{ "a":4 },[1,2,3]]
```

JSON_OBJECT

JSON_OBJECT([key1, val1 [, key2, val2] ...])

JSON_OBJECT 함수는 해당 키/값(key, val1, key, val2,...)쌍을 가진 리스트(텅빈 리스트도 가능)가 포함된 json 객체를 반환한다.

```
SELECT JSON_OBJECT();
```

```
json_object()  
=====  
{}
```

```
SELECT JSON_OBJECT('a', 1, 'b', '1', 'c', json '{"a":4}', 'd', json '[1,2,3]');
```

```
json_object('a', 1, 'b', '1', 'c', json '{"a":4}', 'd', json '[1,2,3]')  
=====  
{ "a":1, "b":"1", "c":{"a":4 }, "d":[1,2,3]}
```

JSON_KEYS

JSON_KEYS(*json_doc* [, *json path*])

JSON_KEYS 함수는 해당 패스로 주어진 json 객체의 모든 키값을 가진 json 배열을 반환한다. 해당 경로가 json 객체가 아닌 json 요소를 지정하면 json null이 반환된다. json 경로 인자가 누락되면 키(key)는 json 루트 요소로부터 가져온다. json 경로가 존재하지 않으면 오류가 발생하고 json_doc 인자가 **NULL** 이면 **NULL** 을 반환한다.

```
SELECT JSON_KEYS('{}');
```

```
json_keys('{}')
=====
[]
```

```
SELECT JSON_KEYS('"non-object"');
```

```
json_keys('"non-object"')
=====
null
```

```
SELECT JSON_KEYS('{"a":1, "b":2, "c":{"d":1}}');
```

```
json_keys('{"a":1, "b":2, "c":{"d":1}}')
=====
["a","b","c"]
```

JSON_DEPTH

JSON_DEPTH(*json_doc*)

JSON_DEPTH 함수는 json의 최대 깊이를 반환한다. 깊이는 1부터 시작하며 깊이 레벨은 비어 있지 않은 json 배열이나 비어있지 않은 json 객체에서 1씩 증가한다. 인자가 **NULL** 이면 **NULL** 을 반환한다.

```
SELECT JSON_DEPTH('"scalar"');
```

```
json_depth('"scalar"')
=====
1
```

```
SELECT JSON_DEPTH(['{"a":4}', 2]);
```

```
json_depth(['{"a":4}', 2])
=====
3
```

[예제] deeper json:

```
SELECT JSON_DEPTH(['{"a":[1,2,3,{"k":[4,5]}]}', 2,3,4,5,6,7]);
```

```
json_depth(['{"a":[1,2,3,{"k":[4,5]}]}', 2,3,4,5,6,7])
=====
6
```

JSON_LENGTH

`JSON_LENGTH(json_doc[, json path])`

JSON_LENGTH 함수는 주어진 경로에 있는 json 요소의 길이를 반환한다. 경로 인자가 주어지지 않으면 json 루트 요소의 길이가 반환된다. 인자가 **NULL** 이거나 해당 경로에 어떤 요소도 존재하지 않으면 **NULL** 이 반환된다.

```
SELECT JSON_LENGTH('"scalar"');
```

```
json_length('"scalar"')
=====
1
```

```
SELECT JSON_LENGTH(['{"a":4}', 2], '$.a');
```

```
json_length(['{"a":4}', 2], '$.a')
=====
NULL
```

```
SELECT JSON_LENGTH('[2, {"a":4, "b":4, "c":4}]', '$[1]');
```

```
json_length('[2, {"a":4, "b":4, "c":4}]', '$[1]')
=====
3
```

```
SELECT JSON_LENGTH('["a":[1,2,3,{"k":[4,5,6,7,8]}],2]');
```

```
json_length('["a":[1,2,3,{"k":[4,5,6,7,8]}],2]')
=====
2
```

JSON_VALID

JSON_VALID(*val*)

JSON_VALID 함수는 해당 *val* 인자가 유효한 json_doc일 경우에 1을 그렇지 않은 경우에 0을 반환한다. 해당 인자가 **NULL** 인 경우 **NULL** 을 반환한다.

```
SELECT JSON_VALID('["a":4}, 2]');
1
SELECT JSON_VALID('{ "wrong json object":' );
0
```

JSON_TYPE

JSON_TYPE(*json_doc*)

JSON_TYPE 함수는 문자열 인자인 *json_doc* 의 타입을 반환한다.

```
SELECT JSON_TYPE ('["a":4}, 2]');
'JSON_ARRAY'
SELECT JSON_TYPE ('{"a":4}');
'JSON_OBJECT'
SELECT JSON_TYPE ('"aaa"');
'String'
```

JSON_QUOTE

JSON_QUOTE(*str*)

JSON_QUOTE 함수는 문자열과 이스케이프된 특수 문자들을 큰따옴표로 묶은 json_string을 결과로 반환한다. *str* 인자가 **NULL** 인 경우 **NULL** 을 반환한다.

```
SELECT JSON_QUOTE ('simple');
```

```
json_unquote('simple')
=====
'"simple"'
```

```
SELECT JSON_QUOTE ('');
```

```
json_unquote('')
=====
'"\"'
```

JSON_UNQUOTE

JSON_UNQUOTE(*json_doc*)

JSON_UNQUOTE 함수는 따옴표로 묶이지 않은 json_value 문자열을 반환한다. *json_doc* 인자가 **NULL** 이면 **NULL** 을 반환한다.

```
SELECT JSON_UNQUOTE ('"\u0032"');
```

```
json_unquote('"\u0032"')
=====
'2'
```

```
SELECT JSON_UNQUOTE ('\"');
```

```
json_unquote('\"')
=====
''
```


JSON_PRETTY

JSON_PRETTY(*json_doc*)

JSON_PRETTY 는 *json_doc* 보기 좋게 출력된 문자열을 반환한다. *json_doc* 인자가 **NULL** 이면 **NULL** 을 반환한다.

```
SELECT JSON_PRETTY('[{ "a": "val1", "b": "val2", "c": [1, "elem2", 3, 4, { "key": "val" } ] } ]');
```

```
json_pretty('[{ "a": "val1", "b": "val2", "c": [1, "elem2", 3, 4, { "key": "val" } ] } ]')
=====
'[
{
  "a": "val1",
  "b": "val2",
  "c": [
    1,
    "elem2",
    3,
    4,
    {
      "key": "val"
    }
  ]
}
]'
```

JSON_SEARCH

JSON_SEARCH(*json_doc*, *one/all*, *search_str* [, *escape_char* [, *json path*] ...])

JSON_SEARCH 함수는 해당 *search_str* 과 일치하는 json 문자열을 포함한 하나의 json 경로 혹은 복수의 json 경로를 반환한다. 일치 여부 검사는 내부의 json 문자열과 *search_str* 에 **LIKE** 연산자를 적용하여 수행된다. **JSON_SEARCH** 의 *escape_char* 및 *search_str* 에 대해 **LIKE** 연산자의 대응 부분과 동일한 규칙이 적용된다. **LIKE** 관련 규칙에 대한 추가 설명은 **LIKE** 을 참고한다.

*one/all*에서 'one'을 사용하면 **JSON_SEARCH** 첫번째 일치 결과가 나타났을 때 탐색이 멈추게 된다. 반면에 'all'을 사용하면 *search_str* 과 일치하는 모든 경로를 탐색하게 된다.

주어진 json 경로는 반환된 경로의 필터를 결정하므로 결과로 나온 json 경로의 접두사 (prefix)는 적어도 하나의 주어진 json 경로 인자와 일치해야 한다. json 경로 인자가 누락된 경우, **JSON_SEARCH** 는 루트 요소로부터 탐색을 시작한다.

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":"a"}', 'one', 'a');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":"a"}', 'one', 'a')
=====
"$a[0]"
```

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a')
=====
["$a[0]", "$b", "$c"]
```

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a',
NULL, '$a', '$b');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a', null,
'$a', '$b')
=====
["$a[0]", "$b"]
```

와일드카드는 좀더 일반적인 형식의 경로 필터로 사용될 수 있다. json 경로는 객체 키 식별자로 시작하는 것만 허용된다.

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a',
NULL, '$.*');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":"a"}', 'all', 'a', null,
'$.*')
=====
["$a[0]", "$b", "$c"]
```

객체 키(key) 식별자로 시작하고 json 배열 인덱스를 따르는 json 경로만 허용함으로써 '\$b', '\$d.e[0]' 일치 항목이 필터링 된다:

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":
["a"]}}', 'all', 'a', NULL, '$.*[*]');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'all', 'a', null, '$.*[*]')
=====
["$a[0]","$c[0]"]
```

json 배열 인덱스를 포함하는 json 경로만 허용함으로써 '\$.b'가 필터링 된다.

```
SELECT JSON_SEARCH('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":
["a"]}}', 'all', 'a', NULL, '$**[*]');
```

```
json_search('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'all', 'a', null, '$**[*]')
=====
["$a[0]","$c[0]","$d.e[0]"]
```

JSON_EXTRACT

JSON_EXTRACT(*json_doc*, *json path* [, *json path*] ...)

해당 경로로 지정된 *json_doc*로부터 json 요소를 반환한다. json 경로 인자가 와일드카드를 포함하는 경우 와일드카드에 의해 포함될 수 있는 모든 경로의 지정된 json 요소가 json 배열 결과로 반환된다. 와일드카드를 사용하지 않고 json 경로에서 하나의 요소만 발견된 경우 하나의 json 요소만 반환되며, 그렇지 않은 경우 발견된 json 요소는 json 배열로 구성되어 반환된다. json 경로가 **NULL** 이거나 유효하지 않은 경우 혹은 *json_doc* 인자가 유효하지 않은 경우 에러가 반환된다. json 요소가 발견되지 않거나 *json_doc*이 **NULL** 인 경우 **NULL** 을 반환한다.

```
SELECT JSON_EXTRACT('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":
["a"]}}', '$.a');
```

```
json_extract('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'$.a')
=====
["a","b"] -- at '$.a' we have the json array ["a","b"]
```

```
SELECT JSON_EXTRACT('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":
["a"]}}', '$.a[*]');
```

```
json_extract('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'$.a[*]')
=====
```

"["a","b"]" -- '\$.a[0]'와 '\$.a[1]'는 json 배열로 구성하여, ["a","b"]를 형성한다.

와일드 카드 '*'를 포함한 이전의 쿼리를 'a'로 바꾸면 '\$.c[0]'가 일치할 것인데, 이것은 정확히 객체 키(key) 식별자와 배열 인덱스가 있는 모든 json 경로와 일치할 것이다.

```
SELECT JSON_EXTRACT('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}', '$.*[*]');
```

```
json_extract('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'$.*[*]')
=====
["a","b","a"]
```

다음 json 경로는 json 배열 인덱스로 끝나는 모든 json 경로와 일치할 것이다 (이전의 일치하는 모든 경로 및 '\$.d.e [0]'과 일치):

```
SELECT JSON_EXTRACT('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}', '$**[*]');
```

```
json_extract('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'$**[*]')
=====
["a","b","a","a"]
```

```
SELECT JSON_EXTRACT('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}', '$.d**[*]');
```

```
json_extract('{ "a":["a","b"], "b":"a", "c":["a"], "d":{"e":["a"]}}',
'$d**[*]')
=====
["a"] -- '$.d.e[0]'은 해당 인자의 경로 패밀리와 일치하는 유일한 경로
이며, '.d'로 시작하고 배열 인덱스로 끝나는 경로이다.
```

->

json_doc -> json path

json_doc 인자가 하나의 컬럼으로 제한된 두 개의 인자를 가지는 **JSON_EXTRACT** 의 별칭 연산자. *json* 경로가 **NULL** 이거나 유효하지 않은 경우 오류를 반환한다. **NULL** *json_doc* 인자가 적용된 경우에는 **NULL** 을 반환한다.

```
CREATE TABLE tj (a json);
INSERT INTO tj values ('{"a":1}'), ('{"a":2}'), ('{"a":3}'), (NULL);

SELECT a->'$.a' from tj;
```

```
json_extract(a, '$.a')
=====
1
2
3
NULL
```

->>

json_doc ->> *json path*

JSON_UNQUOTE 의 별칭 (*json_doc*->*json* 경로). 본 연산자는 컬럼인 *json_doc* 인자에만 적용할 수 있다. *json* 경로가 **NULL** 이거나 유효하지 않은 경우 오류가 발생한다. **NULL** *json_doc* 인자에 적용된 경우 **NULL** 을 반환한다.

```
CREATE TABLE tj (a json);
INSERT INTO tj values ('{"a":1}'), ('{"a":2}'), ('{"a":3}'), (NULL);

SELECT a->>'$.a' from tj;
```

```
json_unquote(json_extract(a, '$.a'))
=====
'1'
'2'
'3'
NULL
```

JSON_CONTAINS_PATH

JSON_CONTAINS_PATH(*json_doc*, *one/all*, *json path* [, *json path*] ...)

JSON_CONTAINS_PATH 함수는 해당 경로가 *json_doc* 내에 존재하는지를 검사한다.

one/all 인자 중 'all'이 적용된 경우 모든 경로가 존재하면 1을 반환하고 그렇지 않으면 0을 반환한다.

one/all 인자 중 'one'이 적용된 경우 하나의 경로라도 존재하면 1을 반환하고 그렇지 않으면 0을 반환한다.

해당 인자가 **NULL** 이면 **NULL** 을 반환한다. 해당 인자가 유효하지 않으면 오류가 발생한다.

```
SELECT JSON_CONTAINS_PATH ('[{"0":0},1,"2",{"three":3}]', 'all',
'$[0]', '$[0]."0"', '$[1]', '$[2]', '$[3]');
```

```
json_contains_path('["0":0},1,"2",{"three":3}]', 'all', '$[0]',
'$[0]."0"', '$[1]', '$[2]', '$[3]')
```

```
=====
```

1

```
SELECT JSON_CONTAINS_PATH ('[{"0":0},1,"2",{"three":3}]', 'all',
'$[0]', '$[0]."0"', '$[1]', '$[2]', '$[3]', '$.inexistent');
```

```
json_contains_path('["0":0},1,"2",{"three":3}]', 'all', '$[0]',
'$[0]."0"', '$[1]', '$[2]', '$[3]', '$.inexistent')
```

```
=====
```

0

JSON_CONTAINS_PATH 함수는 json 경로 내에 와일드카드를 지원한다.

```
SELECT JSON_CONTAINS_PATH ('[{"0":0},1,"2",{"three":3}]', 'one',
'$.inexistent', '$[*]."three"');
```

```
json_contains_path('["0":0},1,"2",{"three":3}]', 'one',
'$.inexistent', '$[*]."three"')
```

```
=====
```

1

JSON_CONTAINS

JSON_CONTAINS(*json_doc doc1*, *json_doc doc2* [, *json path*])

JSON_CONTAINS 함수는 *doc2* 가 옵션으로 지정된 경로의 *doc1* 에 포함되는지를 검사한다. 다음과 같이 재귀 규칙이 충족되는 경우 json 요소에 다른 json 요소가 포함된다.

- 타입이 같고 (**JSON_TYPE** ()이 일치하고) 스칼라도 같은 경우 json 스칼라에 다른 json 스칼라가 포함된다. 예외적으로, json integer는 **JSON_TYPE** ()이 다른 경우에도 json double과 비교를 통해 동일한 것으로 간주될 수 있다.
- json 배열 요소에 json_nonarray가 포함되어 있으면 json 배열에 json 스칼라 또는 json 객체가 포함된다.
- 두 번째 json 배열의 모든 요소가 첫 번째 json 배열에 포함되어 있으면 json 배열에 다른 json 배열이 포함된다.
- 두 번째 객체의 모든 (*key2*, *value2*) 쌍에 대해 첫 번째 객체에 *key1* = *key2* 이고 *value2* 가 *value1* 을 포함하는 (*key1*, *value1*) 쌍이 있는 경우 json 객체에는 다른 json 오브젝트가 포함된다.
- 이 외에는 json 요소가 포함되지 않는다.

json 경로 인자를 입력하지 않은 경우 *doc2* 가 *doc1* 의 루트 json 요소에 포함되는지 여부를 반환한다. 인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_CONTAINS ('"simple"', '"simple"');
```

```
json_contains('"simple"', '"simple"')
=====
1
```

```
SELECT JSON_CONTAINS ('["a", "b"]', '"b"');
```

```
json_contains('["a", "b"]', '"b"')
=====
1
```

```
SELECT JSON_CONTAINS ('["a", "b1", ["a", "b2"]]', '["b1", "b2"]');
```

```
json_contains('["a", "b1", ["a", "b2"]]', '["b1", "b2"]')
=====
1
```

```
SELECT JSON_CONTAINS ('{"k1":["a", "b1"], "k2": ["a",
"b2"]}', '{"k1":"b1", "k2":"b2"}');
```

```
json_contains('{"k1":["a", "b1"], "k2": ["a", "b2"]}', '{"k1":"b1",
"k2":"b2"}')
=====
=====
1
```

json 객체는 json 배열과 같은 방식으로 포함을 검사하지 않으며, json 객체의 하위 요소에 포함된 json 객체의 자손이 아닌 json 요소를 가질 수 없다.

```
SELECT JSON_CONTAINS ('["a", "b1", ["a", {"k":"b2"}]]', '["b1",
"b2"]');
```

```
json_contains('["a", "b1", ["a", {"k":"b2"}]]', '["b1", "b2"]')
=====
0
```

```
SELECT JSON_CONTAINS ('["a", "b1", ["a", {"k":["b2"]}]]', '["b1",
{"k":"b2"}]');
```

```
json_contains('["a", "b1", ["a", {"k":["b2"]}]]', '["b1",
{"k":"b2"}]')
=====
===
1
```

JSON_MERGE_PATCH

JSON_MERGE_PATCH(*json_doc*, *json_doc* [, *json_doc*] ...)

JSON_MERGE_PATCH 함수는 둘 이상의 json 문서를 병합하고 병합된 결과 json을 반환한다. **JSON_MERGE_PATCH** 는 병합 충돌 시 두 번째 인자를 사용하는 점에서 **JSON_MERGE_PRESERVE** 와 다르다. **JSON_MERGE_PATCH** 함수는 RFC 7396 <<https://www.rfc-editor.org/info/rfc7396>> 을 준수한다.

두 개의 json 문서 병합은 다음 규칙에 따라 재귀적으로 수행된다:

- 두 개의 비 객체 JSON이 병합되면 병합 결과는 두 번째 값이다.
- 객체가 아닌 json이 json 객체와 병합되면 빈 객체와 두 번째 병합 인자가 병합된다.
- 두 객체가 병합되면 결과 객체는 다음 멤버로 구성된다:
 - 두 번째 객체와 동일한 키를 가진 멤버를 제외한 첫 번째 객체의 모든 멤버.
 - 첫 번째 객체에 동일한 키를 가진 멤버와 모든 멤버가 null인 값을 제외한 두 번째 객체의 모든 멤버. 두 번째 객체의 null 값을 가진 멤버는 무시된다.
 - 두 번째 객체와 동일한 키를 가진 null이 아닌 첫 번째 객체의 멤버. 두 객체 모두에 나타나는 동일한 키 멤버일 때 두 번째 객체의 멤버 값이 null이면 무시된다. 이 두 개의 값은 첫 번째 및 두 번째 객체의 멤버 값에 대해 수행된 병합의 결과가 된다.

병합 작업은 두 개 이상의 인자가 있을 때 연속적으로 실행된다. 처음 두 인자를 병합한 결과는 세 번째와 병합되고 이 결과는 네 번째와 병합된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_MERGE_PATCH ('["a","b","c"]', '"scalar"');
```

```
json_merge_patch(['a","b","c"], '"scalar"')
=====
"scalar"
```

첫 번째 인자가 객체가 아니고 두 번째 인자가 객체 인 경우 병합에 대한 예외. 빈 객체와 두 번째 객체 인자 간에 병합 작업이 수행된다.

```
SELECT JSON_MERGE_PATCH ('["a"]', '{"a":null}');
```

```
json_merge_patch(['a"], '{"a":null}')
=====
{}
```

기술된 객체 병합 예시:

```
SELECT JSON_MERGE_PATCH ('{"a":null,"c":["elem"]}','{"b":null,"c":
{"k":null},"d":"elem"}');
```

```
json_merge_patch('{"a":null,"c":["elem"]}', '{"b":null,"c":
{"k":null},"d":"elem"}')
=====
{"a":null,"c":{"k":null},"d":"elem"}
```

JSON_MERGE_PRESERVE

JSON_MERGE_PRESERVE(json_doc, json_doc [, json_doc] ...)

JSON_MERGE_PRESERVE 함수는 둘 이상의 json 문서를 병합하고 병합된 결과 json을 반환한다. **JSON_MERGE_PRESERVE** 는 병합 충돌 시 두 json 요소를 모두 보존한다는 점에서 **JSON_MERGE_PATCH** 와 다르다.

두 json 문서의 병합은 다음 규칙에 따라 재귀적으로 수행된다:

- 두 개의 json 배열이 병합되면 연결된다.
- 두 개의 비 배열 (스칼라 / 오브젝트) json 요소가 병합되고 그 중 하나만 json 객체인 경우 결과는 두 개의 json 요소를 포함하는 배열이다.
- 비 배열 json 요소가 json 배열과 병합되면 비 배열은 단일 요소 json 배열로 변경된 다음 json 배열 병합 규칙에 따라 json 배열과 병합된다.
- 두 개의 json 객체가 병합되면 다른 json 객체와 비교해 없는 모든 멤버가 유지된다. 일치하는 키의 경우 규칙을 재귀적으로 적용하여 값이 항상 병합된다.

병합 작업은 두 개 이상의 인자가 있을 때 연속적으로 실행된다. 처음 두 인자를 병합 한 결과는 세 번째와 병합되고 이 결과는 네 번째와 병합된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_MERGE_PRESERVE ('"a"', '"b"');
```

```
json_merge('"a"', '"b"')
=====
["a","b"]
```

```
SELECT JSON_MERGE_PRESERVE ('["a","b","c"]', '"scalar"');
```

```
json_merge(['a","b","c"], 'scalar')
=====
["a","b","c","scalar"]
```

JSON_MERGE_PATCH 와 달리 **JSON_MERGE_PRESERVE** 는 병합하는 동안 첫 번째 인자의 요소를 삭제 및 수정하지 않고 합쳐서 가져온다.

```
SELECT JSON_MERGE_PRESERVE ('{"a":null,"c":["elem"]}','{"b":null,"c":{"k":null},"d":"elem"}');
```

```
json_merge('{"a":null,"c":["elem"]}','{"b":null,"c":{"k":null},"d":"elem"}')
=====
{"a":null,"c":["elem","k":null],"b":null,"d":"elem"}
```

JSON_MERGE

JSON_MERGE(*json_doc, json_doc [, json_doc] ...*)

JSON_MERGE 는 **JSON_MERGE_PRESERVE** 의 별칭이다.

JSON_ARRAY_APPEND

JSON_ARRAY_APPEND(*json_doc, json_path, json_val [, json_path, json_val] ...*)

JSON_ARRAY_APPEND 함수는 첫 번째 인자의 수정된 사본을 반환한다. 주어진 각 <*json_path, json_val*> 에 대해 함수는 해당 경로로 지정된 json 배열에 값을 추가한다.

(*json_path, json_val*) 은 왼쪽에서 오른쪽으로 하나씩 평가한다. 한 쌍을 평가하여 작성된 문서는 다음 쌍을 평가하는 새로운 값이 된다.

json 경로가 *json_doc* 내부의 json 배열을 가리키는 경우 *json_val* 이 배열의 끝에 추가된다. json 경로가 비 배열 json 요소를 지정하는 경우 비 배열 요소를 포함하는 단일 요소 json 배열로 변경되고 *json_val* 을 추가한다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_ARRAY_APPEND ('{"a":[1,2]}', '$.a', 'b');
```

```
json_array_append('{"a":[1,2]}', '$.a', 'b')
=====
{"a":[1,2,"b"]}
```

```
SELECT JSON_ARRAY_APPEND ( '{"a":1}', '$.a', 'b');
```

```
json_array_append('{"a":1}', '$.a', 'b')
=====
{"a":[1,"b"]}
```

```
SELECT JSON_ARRAY_APPEND ( '{"a":[1,2]}', '$.a[0]', '1');
```

```
json_array_append('{"a":[1,2]}', '$.a[0]', '1')
=====
{"a":[[1,"1"],2]}
```

JSON_ARRAY_INSERT

JSON_ARRAY_INSERT(*json_doc*, *json path*, *json_val* [, *json path*, *json_val*] ...)

JSON_ARRAY_INSERT 함수는 첫 번째 인자의 수정된 사본을 반환한다. 주어진 각 < *json path*, *json_val* > 에 대해 함수는 해당 경로로 지정된 json 배열에 값을 삽입한다.

(*json path*, *json_val*) 은 왼쪽에서 오른쪽으로 하나씩 평가한다. 한 쌍을 평가하여 작성된 문서는 다음 쌍을 평가하는 새로운 값이 된다.

JSON_ARRAY_INSERT 작업의 규칙은 다음과 같다:

- json 경로가 json_array의 요소를 지정하면 *json_val* 이 지정된 색인에 삽입되어 다음 요소를 오른쪽으로 이동시킨다.
- json 경로가 배열 끝의 다음을 가리키는 경우, 배열의 끝부터 지정된 색인에 삽입될 때까지 null로 채워진다. 그리고 *json_val*이 지정된 색인에 삽입된다.
- json 경로가 *json_doc* 내에 존재하지 않는 경우, json 경로의 마지막 토큰은 배열 인덱스이고 마지막 배열 인덱스 토큰이 없는 json 경로는 *json_doc* 내의 요소를 지정했을 것이다. 마지막 토큰을 제외한 json 경로로 찾은 요소는 단일 요소 json 배열로 대체되고 **JSON_ARRAY_INSERT** 작업은 원래 json 경로로 수행된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않거나 *json_path* 가 *json_doc* 내부의 배열의 장소를 지정하지 않으면 오류가 발생한다.

```
SELECT JSON_ARRAY_INSERT ('[0,1,2]', '$[0]', '1');
```

```
json_array_insert('[0,1,2]', '$[0]', '1')
=====
["1",0,1,2]
```

```
SELECT JSON_ARRAY_INSERT ('[0,1,2]', '$[5]', '1');
```

```
json_array_insert('[0,1,2]', '$[5]', '1')
=====
[0,1,2,null,null,"1"]
```

Examples for **JSON_ARRAY_INSERT**'s third rule.

```
SELECT JSON_ARRAY_INSERT ('{"a":4}', '$[5]', '1');
```

```
json_array_insert('{"a":4}', '$[5]', '1')
=====
[{"a":4},null,null,null,null,"1"]
```

```
SELECT JSON_ARRAY_INSERT ('"a"', '$[5]', '1');
```

```
json_array_insert('"a"', '$[5]', '1')
=====
["a",null,null,null,null,"1"]
```

JSON_INSERT

JSON_INSERT(*json_doc*, *json path*, *json_val* [, *json path*, *json_val*] ...)

JSON_INSERT 함수는 첫 번째 인자의 수정된 사본을 반환한다. 주어진 각 < *json path*, *json_val*> 에 대해 해당 경로에 다른 값이 없으면 값을 삽입한다.

JSON_INSERT 의 삽입 규칙은 다음과 같다:

json 경로가 *json_doc* 내의 다음 json 값 중 하나를 처리하는 경우 *json_val* 이 삽입된다:

- 기존 json 객체에 존재하지 않는 객체 멤버이다. 키(key)가 json 경로의 마지막 요소이고 값이 *json_val* 인 (*key*, *value*)이 json 오브젝트에 추가된다.
- 기존 json 배열 끝을 넘는 배열 색인. 배열은 배열의 끝까지 널로 채워지고 *json_val* 은 지정된 색인에 삽입된다.

<*json path*, *json_val*> 한 쌍을 평가하여 작성된 *json_doc*은 다음 <*json path*, *json_val*>이 평가될 때 새로운 값이 된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

- *json_doc* * 내의 기존 요소에 대한 경로는 무시된다:

```
SELECT JSON_INSERT ('{"a":1}','$.a','b');
```

```
json_insert('{"a":1}', '$.a', 'b')
=====
{"a":1}
```

```
SELECT JSON_INSERT ('{"a":1}','$.b','1');
```

```
json_insert('{"a":1}', '$.b', '1')
=====
{"a":1,"b":"1"}
```

```
SELECT JSON_INSERT ('[0,1,2]','$[4]','1');
```

```
json_insert('[0,1,2]', '$[4]', '1')
=====
[0,1,2,null,"1"]
```

JSON_SET

JSON_SET(*json_doc*, *json path*, *json_val* [, *json path*, *json_val*] ...)

JSON_SET 함수는 첫 번째 인자의 수정된 사본을 반환한다. 주어진 각 < *json path*, *json_val* >에 대해 함수는 해당 경로의 값을 삽입하거나 대체한다. json 경로가 *json_doc* 내부에서 아래의 json 값 중 하나를 지정하면 *json_val* 이 삽입된다.

- 기존 json 객체의 존재하지 않는 객체 멤버. (*key*, *value*) 이 json 경로에서 추론된 키와 *json_val* 값으로 json 객체에 추가된다.
- 기존 json 배열 끝을 넘는 배열 색인. 배열은 배열의 끝까지 널로 채워지고 *json_val* 은 지정된 색인에 삽입된다.

<*json path*, *json_val*> 한 쌍을 평가하여 작성된 json_doc은 다음 <*json path*, *json_val*>이 평가될 때 새로운 값이 된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_SET ('{"a":1}', '$.a', 'b');
```

```
json_set('{"a":1}', '$.a', 'b')
=====
{"a":"b"}
```

```
SELECT JSON_SET ('{"a":1}', '$.b', '1');
```

```
json_set('{"a":1}', '$.b', '1')
=====
{"a":1,"b":"1"}
```

```
SELECT JSON_SET ('[0,1,2]', '$[4]', '1');
```

```
json_set('[0,1,2]', '$[4]', '1')
=====
[0,1,2,null,"1"]
```

JSON_REPLACE

JSON_REPLACE(*json_doc*, *json path*, *json_val* [, *json path*, *json_val*] ...)

JSON_REPLACE 함수는 첫 번째 인자의 수정된 사본을 반환한다. 주어진 각 < *json path*, *json_val* >에 대해 해당 경로에 다른 값이 있는 경우에만 값을 대체한다.

json_path 가 *json_doc* 내에 존재하지 않으면 (*json path*, *json_val*) 이 무시되고 변경되지 않는다.

<*json path*, *json_val*> 한 쌍을 평가하여 작성된 *json_doc*은 다음 <*json path*, *json_val*>이 평가 될 때 새로운 값이 된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않은 경우 오류가 발생한다.

```
SELECT JSON_REPLACE ('{"a":1}', '$.a', 'b');
```

```
json_replace('{"a":1}', '$.a', 'b')
=====
{"a":"b"}
```

json_doc 안에 *json path* 가 없으면 대체 (replace) 동작이 수행되지 않는다.

```
SELECT JSON_REPLACE ('{"a":1}', '$.b', '1');
```

```
json_replace('{"a":1}', '$.b', '1')
=====
{"a":1}
```

```
SELECT JSON_REPLACE ('[0,1,2]', '$[4]', '1');
```

```
json_replace('[0,1,2]', '$[4]', '1')
=====
[0,1,2]
```

JSON_REMOVE

JSON_REMOVE(*json_doc*, *json path* [, *json path*] ...)

JSON_REMOVE 함수는 주어진 모든 경로에서 값을 제거하여 첫 번째 인자의 수정된 사본을 반환한다.

json 경로 인자는 왼쪽에서 오른쪽으로 하나씩 평가된다. *json* 경로를 평가하여 생성된 결과는 다음 *json* 경로가 평가되는 값이 된다.

인자가 **NULL** 이면 **NULL** 을 반환한다. 인자가 유효하지 않거나 경로가 루트를 지정하거나 경로가 없는 경우 오류가 발생한다.


```
SELECT JSON_REMOVE ('[0,1,2]','$[1]');
```

```
json_remove('[0,1,2]','$[1]')
=====
[0,2]
```

```
SELECT JSON_REMOVE ('{"a":1,"b":2}','$.a');
```

```
json_remove('{"a":1,"b":2}','$.a')
=====
{"b":2}
```

JSON_TABLE

JSON_TABLE 함수는 json을 일반 테이블과 유사하게 질의(query)할 수 있는 유사 테이블 구조로 변환하는 것을 용이하게 해준다. 변환은 예를 들면 JSON_ARRAY 요소를 확장함으로써, 하나의 행 또는 복수의 행을 생성한다.

JSON_TABLE 의 전체 문법 :

```
JSON_TABLE(
    expr,
    path COLUMNS (column_list)
) [AS] alias

<column_list>::=
    <column> [, <column>] ...

<column>::=
    name FOR ORDINALITY
    | name type PATH string_path <on_empty> <on_error>
    | name type EXISTS PATH string_path
    | NESTED [PATH] string_path COLUMNS <column_list>

<on_empty>::=
    NULL | ERROR | DEFAULT value ON EMPTY

<on_error>::=
    NULL | ERROR | DEFAULT value ON ERROR
```

`json_doc expr`은 결과가 `json_doc`이 되는 표현식이어야 한다. 상수 `json`, 테이블의 열 또는 함수 또는 연산자의 결과 일 수 있다. `json path`는 유효한 경로 이어야 하며 **COLUMNS** 절에서 평가할 `json` 데이터를 추출하는 데 사용된다. **** COLUMNS **** 절은 열 유형 및 출력을 얻기 위해 수행되는 작업을 정의한다. **[AS] alias** 절이 필요하다.

JSON_TABLE은 네 가지 유형의 열을 지원한다:

- **name FOR ORDINALITY** : 이 유형은 **COLUMNS** 절 내에서 행 번호를 추적한다. 열 유형은 **INTEGER**이다.
- **name type PATH json path [on empty] [on error]** : 이 유형의 열은 지정된 `json` 경로에서 `json_values`를 추출하는 데 사용된다. 추출된 `json` 데이터는 지정된 유형으로 강제 변환된다. 경로가 존재하지 않으면 **on empty** 절이 자동 수행된다. 추출된 `json` 값이 대상 유형으로 변환되지 않으면 **on error** 절이 자동 수행된다.
 - **on empty**은 경로가 존재하지 않는 경우 **JSON_TABLE**의 동작을 결정한다. **on empty**은 다음 값 중 하나를 가질 수 있다:
 - **NULL ON EMPTY** : 열이 **NULL**로 설정된다. 이것이 기본 동작이다.
 - **ERROR ON EMPTY**: 오류가 발생한다.
 - **DEFAULT value ON EMPTY**: 빈 값 대신에 `value`가 사용된다.
 - **on error**는 다음 값 중 하나를 가질 수 있다.:
 - **NULL ON ERROR**: 열이 **NULL**로 설정된다. 이것이 기본 동작이다.
 - **ERROR ON ERROR**: 오류가 발생한다.
 - **DEFAULT value ON ERROR**: 원하는 열 유형으로 강제 변환하지 못한 배열 / 객체 / `json` 스칼라 대신 `value`가 사용된다.
- **name type EXISTS PATH json path**: `json` 경로 위치에 데이터가 있으면 1을 반환하고 그렇지 않으면 0을 반환한다.
- **NESTED [PATH] json path COLUMNS (column list)**는 경로에서 찾은 `json` 데이터에서 부모 결과와 결합된 행과 열의 개별 서브 세트를 생성한다. 결과는 “for each”루프와 유사하게 결합된다. `json` 경로는 부모 경로와 관련이 있다. **COLUMNS** 절에 대해 동일한 규칙이 재귀적으로 적용된다.

```
SELECT * FROM JSON_TABLE (
    '{"a": [1, [2, 3]]}',
    '$.a[*]' COLUMNS ( col INT PATH '$' )
) AS jt;
```

```
col
=====
```

```

1 -- first value found at '$.a[*]' is 1 json
scalar, which is coercible to 1
NULL -- second value found at '$.a[*]' is [2,3]
json array which cannot be coerced to int, triggering NULL ON ERROR
default behavior

```

기본 on_error 동작을 재정의하면 이전 예제와 다른 결과가 나타난다:

```

SELECT * FROM JSON_TABLE (
  '{"a":[1,[2,3]]}',
  '$.a[*]' COLUMNS ( col INT PATH '$' DEFAULT '-1' ON ERROR)
) AS jt;

```

```

col
=====
1 -- first value found at '$.a[*]' is '1' json
scalar, which is coercible to 1
-1 -- second value found at '$.a[*]' is '[2,3]'
json array which cannot be coerced to int, triggering ON ERROR

```

ON EMPTY example:

```

SELECT * FROM JSON_TABLE (
  '{"a":1}',
  '$' COLUMNS ( col1 INT PATH '$.a',
                 col2 INT PATH '$.b',
                 col3 INT PATH '$.c' DEFAULT '0' ON EMPTY)
) AS jt;

```

```

col1      col2      col3
=====
1         NULL      0

```

아래 예에서 '\$.' 경로는 상위 열이 루트 json 객체의 멤버 값을 하나씩 받도록 하는 데 사용된다. 열 a는 처리된 내용을 보여준다. 그런 다음 루트 오브젝트의 각 멤버 값은 ****NESTED*** [PATH] 절에 의해 추가로 처리된다. **NESTED PATH**는 경로 '\$[*]'를 사용하여 배열의 각 요소를 열에 의해 추가로 처리한다. **FOR ORDINALITY** 열은 현재 처리된 요소의 수를 추적한다. 예제 결과에서 우리는 열의 각 새로운 요소에 대해 * ord * 열의 값도 증가함을 알 수 있다. **FOR ORDINALITY nested_ord** 열은 형제 열 (sibling columns)에 의해 처리되는 요소 수의 카운터 역할도 한다. 중첩된 **FOR ORDINALITY** 열은 각 일괄 처리가 완료된 후 재설정된다. 세 번째 멤버의 값인 6은 배열로 취급될 수 없으므로 중첩된 열로 처리할 수 없다. 중첩된 열은 **NULL** 값을 생성한다.

```

SELECT * FROM JSON_TABLE (
    '{"a":[1,2],"b":[3,4,5],"d":6,"c":[7]}' ,
    '$.*' COLUMNS ( ord FOR ORDINALITY,
                     col JSON PATH '$',
                     NESTED PATH '$[*]' COLUMNS ( nested_ord FOR
ORDINALITY,
                                                    nested_col JSON
PATH '$'))
    ) AS jt;

```

ord	col	nested_ord	nested_col
1	[1,2]	1	1
1	[1,2]	2	2
2	[3,4,5]	1	3
2	[3,4,5]	2	4
2	[3,4,5]	3	5
3	6	NULL	NULL
4	[7]	1	7

다음 예는 여러 개의 동일한 레벨 **NESTED [PATH]** 절이 **JSON_TABLE** 에 의해 처리되는 방법을 보여준다. 처리될 값은 각 **NESTED [PATH]** 절에 하나씩 차례로 순서대로 전달된다. **NESTED [PATH]** 절로 값을 처리하는 동안 형제 **NESTED [PATH]** 절은 **NULL** 값으로 열을 채운다.

```

SELECT * FROM JSON_TABLE (
    '{"a":{"key1":[1,2], "key2":[3,4,5]},"b":{"key1":6, "key2":
[7]}}' ,
    '$.*' COLUMNS ( ord FOR ORDINALITY,
                     col JSON PATH '$',
                     NESTED PATH '$.key1[*]' COLUMNS (
nested_ord1 FOR ORDINALITY,
nested_col1 JSON PATH '$'),
                     NESTED PATH '$.key2[*]' COLUMNS (
nested_ord2 FOR ORDINALITY,
nested_col2 JSON PATH '$'))
    ) AS jt;

```

ord	col	nested_ord1
nested_col1	nested_ord2	nested_col2
1	1	1
1	2	1
1	3	1
1	6	NULL
1	7	1

2		NULL	NULL	
	1	{"key1":[1,2], "key2":[3,4,5]}		NULL
NULL		1	3	
	1	{"key1":[1,2], "key2":[3,4,5]}		NULL
NULL		2	4	
	1	{"key1":[1,2], "key2":[3,4,5]}		NULL
NULL		3	5	
	2	{"key1":6, "key2":[7]}		NULL
NULL		1	7	

An example for multiple layers **NESTED [PATH]** clauses:

```

SELECT * FROM JSON_TABLE (
  '{"a":{"key1":[1,2], "key2":[3,4,5]}, "b":{"key1":6, "key2":
[7]}}',
  '$.*' COLUMNS ( ord FOR ORDINALITY,
                    col JSON PATH '$',
                    NESTED PATH '$.*' COLUMNS ( nested_ord1 FOR
ORDINALITY,
                                                    nested_col1 JSON
PATH '$',
                                                    NESTED PATH '$
[*]' COLUMNS ( nested_ord11 FOR ORDINALITY,
                    nested_col11 JSON PATH '$' )),
                    NESTED PATH '$.key2[*]' COLUMNS (
nested_ord2 FOR ORDINALITY,
nested_col2 JSON PATH '$'  ))
) AS jt;

```

ord	col	nested_ord1
nested_col1	nested_ord11	nested_col11
nested_ord2	nested_col2	
=====		
=====		
=====		
2]	1 {"key1":[1,2], "key2":[3,4,5]}	1 [1,
NULL	1 1	NULL
2]	1 {"key1":[1,2], "key2":[3,4,5]}	1 [1,
NULL	2 2	NULL
5]	1 {"key1":[1,2], "key2":[3,4,5]}	2 [3,4,
NULL	1 3	NULL
5]	1 {"key1":[1,2], "key2":[3,4,5]}	2 [3,4,
NULL	2 4	NULL

```

5]      1  {"key1":[1,2],"key2":[3,4,5]}      2  [3,4,
NULL      3  5      NULL
NULL      1  {"key1":[1,2],"key2":[3,4,5]}      NULL
1  3      NULL  NULL
NULL      1  {"key1":[1,2],"key2":[3,4,5]}      NULL
2  4      NULL  NULL
NULL      1  {"key1":[1,2],"key2":[3,4,5]}      NULL
3  5      NULL  NULL
6      2  {"key1":6,"key2":[7]}      1
NULL  NULL      NULL  NULL
[7]      2  {"key1":6,"key2":[7]}      2
NULL  NULL      1  7
NULL      2  {"key1":6,"key2":[7]}      NULL
1  7      NULL  NULL

```

LOB 함수

목차

LOB 함수	570
● BIT_TO_BLOB	571
● BLOB_FROM_FILE	571
● BLOB_LENGTH	572
● BLOB_TO_BIT	572
● CHAR_TO_BLOB	572
● CHAR_TO_CLOB	573
● CLOB_FROM_FILE	573
● CLOB_LENGTH	574

- CLOB_TO_CHAR

574

BIT_TO_BLOB

BIT_TO_BLOB(*blob_type_column_or_value*)**BIT**, **VARYING BIT** 타입을 **BLOB** 타입으로 변환한다.

매개변수:

blob_type_column_or_value – 변환 대상 칼럼 또는 값

반환 형식:

BLOB

BLOB_FROM_FILE

BLOB_FROM_FILE(*file_pathname*)**VARCHAR** 타입의 파일 경로에서 파일 내용을 읽어 **BLOB** 타입 데이터로 반환한다.

매개변수:

file_pathname – CAS나 CSQL과 같은 DB 클라이언트가 구동하는 서버 상의 경로

반환 형식:

BLOB

```
SELECT CAST(BLOB_FROM_FILE('local:/home/cubrid/demo/lob/ces_622/
image_t.00001352246696287352_4131')
AS BIT VARYING) result;

SELECT CAST(BLOB_FROM_FILE('file:/home/cubrid/demo/lob/ces_622/
image_t.00001352246696287352_4131')
AS BIT VARYING) result;
```

BLOB_LENGTH

BLOB_LENGTH(*blob_column*)

BLOB 파일에 저장된 **LOB** 데이터의 길이를 바이트 단위로 반환한다.

매개변수:

blob_column – 길이를 구하고자 하는 BLOB 타입의 칼럼

반환 형식:

INT

BLOB_TO_BIT

BLOB_TO_BIT(*blob_type_column*)

BLOB 타입을 **VARYING BIT** 타입으로 변환한다.

매개변수:

blob_type_column – 변환 대상 칼럼

반환 형식:

VARYING BIT

CHAR_TO_BLOB

CHAR_TO_BLOB(*char_type_column_or_value*)

CHAR, **VARCHAR** 타입을 **BLOB** 타입으로 변환한다.

매개변수:

char_type_column_or_value – 변환 대상 칼럼 또는 값

반환 형식:

BLOB

CHAR_TO_CLOB

CHAR_TO_CLOB(*char_type_column_or_value*)

CHAR, **VARCHAR** 타입을 **CLOB** 타입으로 변환한다.

매개변수:

char_type_column_or_value - 변환 대상 칼럼 또는 값

반환 형식:

CLOB

CLOB_FROM_FILE

CLOB_FROM_FILE(*file_pathname*)

VARCHAR 타입의 파일 경로에서 파일 내용을 읽어 **CLOB** 타입 데이터로 반환한다.

매개변수:

file_pathname - CAS나 CSQL과 같은 DB 클라이언트가 구동하는 서버 상의 경로

반환 형식:

CLOB

file_pathname을 상대 경로로 명시한 경우, 상위 경로는 프로세스의 현재 작업 디렉터리가 된다.

이 함수가 호출된 구문에 대해서는 실행 계획을 캐싱하지 않는다.

```
SELECT CAST(CLOB_FROM_FILE('local:/home/cubrid/demo/lob/ces_622/
image_t.00001352246696287352_4131')
AS VARCHAR) result;

SELECT CAST(CLOB_FROM_FILE('file:/home/cubrid/demo/lob/ces_622/
image_t.00001352246696287352_4131')
AS VARCHAR) result;
```

CLOB_LENGTH

CLOB_LENGTH(*clob_column*)

CLOB 파일에 저장된 **LOB** 데이터의 길이를 바이트 단위로 반환한다.

매개변수:

clob_column – 길이를 구하고자 하는 **CLOB** 타입의 칼럼

반환 형식:

INT

CLOB_TO_CHAR

CLOB_TO_CHAR(*clob_type_column* [USING *charset*])

CLOB 타입을 **VARCHAR** 타입으로 변환한다.

매개변수:

- **clob_type_column** – 변환 대상 칼럼
- **charset** – 변환할 문자열의 문자셋. utf8, euckr, iso885910이
올 수 있다.

반환 형식:

STRING

데이터 타입 변환 함수와 연산자

목차

데이터 타입 변환 함수와 연산자

7

-
- CAST 146
 - DATE_FORMAT 582

● FORMAT	588
● STR_TO_DATE	589
● TIME_FORMAT	591
● TO_CHAR(date_time)	592
● TO_CHAR(number)	603
● TO_DATE	607
● TO_DATETIME	609
● TO_DATETIME_TZ	612
● TO_NUMBER	612
● TO_TIME	614
● TO_TIMESTAMP	616
● TO_TIMESTAMP_TZ	618

CAST

CAST(*cast_operand AS cast_target*)

CAST 연산자를 **SELECT** 문에서 어떤 값의 데이터 타입을 다른 데이터 타입으로 명시적으로 변환하는 데 사용할 수 있다. 조회 리스트 또는 **WHERE** 절의 값 수식을 다른 데이터 타입으로 변환할 수 있다.

매개변수:

- **cast_operand** – 다른 타입으로 변환할 값을 선언한다.
- **cast_target** – 변환할 타입을 지정한다.

반환 형식:

cast_target

경우에 따라 **CAST** 연산자를 쓰지 않고 데이터 타입이 자동으로 변환될 수 있다. 이에 대한 자세한 내용은 **묵시적 타입 변환**을 참고한다.

문자열을 날짜/시간 타입으로 변환하는 것에 대한 자세한 내용은 **문자열을 날짜/시간 타입으로 CAST**를 참고한다.

CUBRID에서 **CAST** 연산자를 사용한 명시적인 타입 변환에 대해서 정리하면 다음의 표와 같다.

From \ To	EN	AN	VC	FC	VB	FB	ES	EL	EC	DS	DT	UT	UTZ	DTZ	DS	MS	SS
EN	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No
AN	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No
VC	Yes	Yes	Yes(s*)	Yes(s*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
FC	Yes	Yes	Yes(s*)	Yes(s*)	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No
VB	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No
FB	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No	No	No

S	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y
	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	e	e	e
																s	s	s

MS	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y
	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	e	e	e
																s	s	s

SQ	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	Y	Y	Y
	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	e	e	e
																s	s	s

(*): 이 경우에 **CAST** 연산은 값 수식과 변환할 데이터 타입이 같은 문자셋을 가질 경우에만 허용된다.

· 데이터 타입 키

- **EN**: 정확한 수치(**INTEGER, SMALLINT, BIGINT, NUMERIC, DECIMAL**)
- **AN**: 근사값 수치(**FLOAT/REAL, DOUBLE**)
- **VC**: 가변 길이 문자열(**VARCHAR (n)**)
- **FC**: 고정 길이 문자열(**CHAR (n)**)
- **VB**: 가변 길이 비트열(**BIT VARYING (n)**)
- **FB**: 고정 길이 비트열(**BIT (n)**)
- **ENUM**: **ENUM** 타입
- **BLOB**: DB 외부에 저장하는 바이너리 데이터(**BLOB**)
- **CLOB**: DB 외부에 저장하는 문자열 데이터(**CLOB**)
- **D**: **DATE**
- **T**: **TIME**
- **DT**: **DATETIME**
- **DTZ**: **DATETIME WITH TIME ZONE** 및 **DATETIME WITH LOCAL TIME ZONE** 데이터 타입
- **UT**: **TIMESTAMP**
- **UTZ**: **TIMESTAMP WITH TIME ZONE** 및 **TIMESTAMP WITH LOCAL TIME ZONE** 데이터 타입

- **S: SET**
- **MS: MULTISSET**
- **SQ: LIST (= SEQUENCE)**

```
--operation after casting character as INT type returns 2  
SELECT (1+CAST ('1' AS INT));
```

2

```
--cannot cast the string which is out of range as SMALLINT  
SELECT (1+CAST('1234567890' AS SMALLINT));
```

ERROR: Cannot coerce value of domain "character" to domain "smallint".

```
--operation after casting returns 1+1234567890  
SELECT (1+CAST('1234567890' AS INT));
```

1234567891

```
--'1234.567890' is casted to 1235 after rounding up  
SELECT (1+CAST('1234.567890' AS INT));
```

1236

```
--'1234.567890' is casted to string containing only first 5 letters.  
SELECT (CAST('1234.567890' AS CHAR(5)));
```

'1234.'

```
--numeric type can be casted to CHAR type only when enough length is  
specified  
SELECT (CAST(1234.567890 AS CHAR(5)));
```

```
ERROR: Cannot coerce value of domain "numeric" to domain "character".
```

```
--numeric type can be casted to CHAR type only when enough length is specified
```

```
SELECT (CAST(1234.567890 AS CHAR(11)));
```

```
'1234.567890'
```

```
--numeric type can be casted to CHAR type only when enough length is specified
```

```
SELECT (CAST(1234.567890 AS VARCHAR));
```

```
'1234.567890'
```

```
--string can be casted to time/date types only when its literal is correctly specified
```

```
SELECT (CAST('2008-12-25 10:30:20' AS TIMESTAMP));
```

```
10:30:20 AM 12/25/2008
```

```
SELECT (CAST('10:30:20' AS TIME));
```

```
10:30:20 AM
```

```
--string can be casted to TIME type when its literal is same as TIME's.
```

```
SELECT (CAST('2008-12-25 10:30:20' AS TIME));
```

```
10:30:20 AM
```

```
--string can be casted to TIME type after specifying its type of the string
```

```
SELECT (CAST(TIMESTAMP'2008-12-25 10:30:20' AS TIME));
```



```
10:30:20 AM
```

```
SELECT CAST('abcde' AS BLOB);
```

```
file:/home1/user1/db/tdb/lob/ces_743/ces_temp.  
00001283232024309172_1342
```

```
SELECT CAST(B'11010000' as varchar(10));
```

```
'd0'
```

```
SELECT CAST('1A' AS BLOB);
```

```
X'1a00'
```

```
--numbers can be casted to TIMESTAMP type  
SELECT CAST (1 AS TIMESTAMP), CAST (1.2F AS TIMESTAMP);
```

```
09:00:01 AM 01/01/1970      09:00:01 AM 01/01/1970
```

```
--numbers cannot be casted to DATETIME type  
SELECT CAST (1 AS DATETIME);
```

```
Cannot coerce 1 to type datetime
```

```
--TIMESTAMP cannot be casted to numbers  
SELECT CAST (TIMESTAMP'09:00:01 AM 01/01/1970' AS INT)
```

```
Cannot coerce timestamp '09:00:01 AM 01/01/1970' to type integer.
```

참고

- **CAST** 변환은 같은 문자셋을 가지는 데이터 타입끼리만 허용된다.
- 근사치 데이터 타입(FLOAT, DOUBLE)이 정수형으로 변환되는 경우, 소수점 아래 자리가 반올림 처리된다.
- 정확한 수치 데이터 타입(NUMERIC)이 정수형으로 변환되는 경우, 소수점 아래 자리가 반올림 처리된다.
- 수치 데이터 타입을 문자열 타입으로 변환하는 경우, 문자열의 길이가 (모든 유효 숫자 자리 + 소수점) 이상이 되도록 충분히 지정해야 한다. 그렇지 않으면 에러가 발생한다.
- 문자열 타입 *A*를 문자열 타입 *B*로 변환하는 경우, *A*의 길이 이상이 되도록 충분히 지정되지 않으면 문자열 끝 부분이 삭제(truncate)되어 저장된다.
- 문자열 타입 *A*를 날짜/시간 데이터 타입 *B*로 변환하는 경우, *A*의 리터럴이 *B* 타입과 일치하는 경우에만 변환된다. 그렇지 않을 경우 에러가 발생한다.
- 문자열로 저장된 수치 데이터는 명시적으로 타입 변환을 해주어야 산술 연산이 가능하다.

DATE_FORMAT

DATE_FORMAT(*date*, *format*)

DATE_FORMAT 함수는 날짜를 포함하는 날짜/시간 타입 값을 지정된 날짜/시간 형식의 문자열로 출력하며, 리턴 값은 **VARCHAR** 타입이다. 지정할 *format* 인자는 아래의 **날짜/시간 형식 2** 표를 참고한다. **날짜/시간 형식 2** 표는 `DATE_FORMAT()` 함수, `TIME_FORMAT()` 함수, `STR_TO_DATE()` 함수에서 사용된다.

매개변수:

- **date** - DATE, TIMESTAMP, DATETIME, DATETIME2, DATETIME2, TIME2, 또는 TIME2PLTZ 타입의 값
- **format** - 출력 형식을 지정한다. '%'로 시작하는 형식 지정자(specifier)를 사용한다.

반환 형식:

STRING

format 인자가 지정되면 지정된 언어에 맞는 형식으로 날짜를 출력한다. 이때 언어는 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용된다. **intl_date_lang**이 지정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *format* 이 “%d %M %Y”인 경우 “2009년 10월 3일”인 날짜를 “3 Oktober 2009”인 문자열로 출력한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

아래 **날짜/시간 형식 2** 표에서 월 이름, 요일 이름, 일 이름, 오전/오후 이름 등은 언어에 따라 다르다.

날짜/시간 형식 2

format 값	의미
%a	Weekday, 영문 약어 (Sun, ..., Sat)
%b	Month, 영문 약어 (Jan, ..., Dec)
%c	Month(1, ..., 12)
%D	Day of the month, 서수 영문 문자열(1st, 2nd, 3rd, ...)
%d	Day of the month, 두 자리 숫자(01, ..., 31)
%e	Day of the month (1, ..., 31)
%f	Milliseconds, 세 자리 숫자 (000, ..., 999)
%H	Hour, 24시간 기준, 두 자리 수 이상 (00, ..., 23, ..., 100, ...)
%h	Hour, 12시간 기준 두 자리 숫자 (01, ..., 12)
%l	Hour, 12시간 기준 두 자리 숫자 (01, ..., 12)

format 값	의미
%i	Minutes, 두 자리 숫자 (00, ..., 59)
%j	Day of year, 세 자리 숫자 (001, ..., 366)
%k	Hour, 24시간 기준, 한 자리 수 이상 (0, ..., 23, ..., 100, ...)
%l	Hour, 12시간 기준 (1, ..., 12)
%M	Month, 영문 문자열 (January, ..., December)
%m	Month, 두 자리 숫자 (01, ..., 12)
%p	AM or PM
%r	Time, 12 시간 기준, 시:분:초 (hh:mi:ss AM or hh:mi:ss PM)
%S	Seconds, 두 자리 숫자 (00, ..., 59)
%s	Seconds, 두 자리 숫자 (00, ..., 59)
%T	Time, 24시간 기준, 시:분:초 (hh:mi:ss)
%U	Week, 두 자리 숫자, 일요일이 첫날인 주 단위 (00, ..., 53)
%u	Week, 두 자리 숫자, 월요일이 첫날인 주 단위 (00, ..., 53)

format 값	의미
%V	Week, 두 자리 숫자, 일요일이 첫날인 주 단위 (01, ..., 53) %X와 결합되어 사용 가능
%v	Week, 두 자리 숫자, 월요일이 첫날인 주 단위 (01, ..., 53) %x 와 결합되어 사용 가능
%W	Weekday, 영문 문자열 (Sunday, ..., Saturday)
%w	Day of the week, 숫자 인덱스 (0=Sunday, ..., 6=Saturday)
%X	Year, 네 자리 숫자, 일요일이 첫날인 주 단위로 계산(0000, ..., 9999) %V와 결합되어 사용 가능
%x	Year, 네 자리 숫자, 월요일이 첫날인 주 단위로 계산(0000, ..., 9999) %v와 결합되어 사용 가능
%Y	Year, 네 자리 숫자 (0001, ..., 9999)
%y	Year, 두 자리 숫자 (00, 01, ..., 99)
%%	특수문자 “%”를 그대로 출력하는 경우
%x	형식 지정자로 쓰이지 않는 영문자 중 임의의 문자 x를 그대로 출력하는 경우
%TZR	타임존 영역 정보(예: US/Pacific)
%TZD	일광 절약 정보(예: KST, KT, EET)

format 값	의미
%TZH	타임존의 시간 오프셋(예: +09, -09)
%TzM	타임존의 분 오프셋 (예: +00, +30)

참고

%TZR, %TZD, %TZH, %TzM은 타임존 타입에서만 사용 가능하다.

참고

TZD 뒤에 숫자를 명시하는 포맷

TZD 뒤에 숫자를 명시하는 포맷을 참고한다.

다음은 시스템 파라미터 **intl_date_lang**의 값이 “en_US”인 경우의 예이다.

```
SELECT DATE_FORMAT(datetime'2009-10-04 22:23:00', '%W %M %Y');
```

```
'Sunday October 2009'
```

```
SELECT DATE_FORMAT(datetime'2007-10-04 22:23:00', '%H:%i:%s');
```

```
'22:23:00'
```

```
SELECT DATE_FORMAT(datetime'1900-10-04 22:23:00', '%D %y %a %d %m %b %j');
```

```
'4th 00 Thu 04 10 Oct 277'
```

```
SELECT DATE_FORMAT(date'1999-01-01', '%X %V');
```

```
'1998 52'
```

다음은 시스템 파라미터 **intl_date_lang** 의 값이 “de_DE”인 경우의 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE";  
SELECT DATE_FORMAT(datetime'2009-10-04 22:23:00', '%W %M %Y');
```

```
'Sonntag Oktober 2009'
```

```
SELECT DATE_FORMAT(datetime'2007-10-04 22:23:00', '%H:%i:%s %p');
```

```
'22:23:00 Nachm.'
```

```
SELECT DATE_FORMAT(datetime'1900-10-04 22:23:00', '%D %y %a %d %m %b  
%j');
```

```
'4 00 Do. 04 10 Okt 277'
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 [TO_CHAR\(\)](#) 의 **Note**를 참고한다.

다음은 타임존 정보를 포함하는 DATETIME TZ 타입의 값을 원하는 형식에 맞게 문자열로 변환하여 출력하는 예제이다.

```
SELECT DATE_FORMAT(datetimetz'2012-02-02 10:10:10 Europe/Zurich  
CET', '%TZR %TZD %TZh %TZM');
```

```
'Europe/Zurich CET 01 00'
```

FORMAT

FORMAT(*x*, *dec*)

FORMAT 함수는 숫자 *x*의 형식이 `#####.#####` 이 되도록, 소수점 위 세 자리마다 자릿수 구분 기호로 구분하고 소수점 기호 아래 숫자가 *dec* 만큼 표현되도록 *dec*의 아랫자리에서 반올림을 수행한 결과를 **VARCHAR** 타입으로 반환한다.

매개변수:

- **x** – 수치 값을 반환하는 임의의 연산식이다.
- **dec** – 소수점 이하 자릿수

반환 형식:

STRING

자릿수 구분 기호와 소수점 기호는 지정한 언어에 맞는 형식으로 출력한다. 이때 언어는 **intl_number_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_number_lang** 값이 지정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”나 “fr_FR”과 같은 유럽 국가의 언어이면 “.”를 숫자의 자릿수 구분 기호로 해석하고 “,”를 소수점 기호로 해석한다([언어별 숫자의 기본 출력](#) 참고).

다음은 시스템 파라미터 **intl_number_lang**의 값을 “en_US”로 설정하여 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_number_lang="en_US"';
SELECT FORMAT(12000.123456,3), FORMAT(12000.123456,0);
```

```
'12,000.123'          '12,000'
```

다음은 시스템 파라미터 **intl_number_lang**의 값을 “de_DE”로 설정하여 생성한 데이터베이스에서 실행한 예이다. 독일, 프랑스 등 유럽 국가 대부분의 숫자 출력 형식은 “.”가 자릿수 구분 기호이고, “,”가 소수점 기호이다.

```
SET SYSTEM PARAMETERS 'intl_number_lang="de_DE"';
SELECT FORMAT(12000.123456,3), FORMAT(12000.123456,0);
```

```
'12.000,123'          '12.000'
```


STR_TO_DATE

STR_TO_DATE(*string*, *format*)

STR_TO_DATE 함수는 인자로 주어진 문자열을 지정된 형식에 따라 해석하여 날짜/시간 값으로 변환하며, **DATE_FORMAT()** 함수와 반대로 동작한다. 리턴 값은 문자열에 포함된 날짜 또는 시간 부분에 따라 타입이 결정된다.

매개변수:

- **string** – 문자열
- **format** – 문자열 해석을 위한 형식을 지정한다. %를 포함하는 문자열을 형식 지정자(specifier)로 사용한다.
DATE_FORMAT() 함수의 **날짜/시간 형식 2** 표를 참고한다.

반환 형식:

DATETIME, DATE, TIME, DATETIME_TZ

지정할 *format* 인자는 **DATE_FORMAT()** 함수의 **날짜/시간 형식 2** 표를 참고한다.

*string*에 유효하지 않은 날짜/시간 값이 포함되거나, *format*에 지정된 형식 지정자를 적용하여 문자열을 해석할 수 없으면 에러를 리턴한다.

format 인자가 지정되면 지정된 언어에 맞는 형식으로 *string* 을 해석한다. 이때 언어는 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용된다. **intl_date_lang** 값이 지정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *format* 이 “%d %M %Y”인 경우 “3 Oktober 2009”인 문자열을 “2009년 10월 3일”인 **DATE** 타입으로 해석한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

인자의 연, 월, 일에는 0을 입력할 수 없으나, 예외적으로 날짜와 시간이 모두 0인 값을 입력한 경우에는 날짜와 시간 값이 모두 0인 **DATE**, **DATETIME** 타입의 값을 반환한다. 그러나 JDBC 프로그램에서는 연결 URL 속성인 zeroDateTimeBehavior의 설정에 따라 동작이 달라진다. 이에 관한 자세한 내용은 **연결 설정**을 참고하면 된다.

다음은 시스템 파라미터 **intl_date_lang** 의 값이 “en_US”인 경우의 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="en_US"';
SELECT STR_TO_DATE('01,5,2013', '%d,%m,%Y');
```

```
05/01/2013
```

```
SELECT STR_TO_DATE('May 1, 2013', '%M %d,%Y');
```

```
05/01/2013
```

```
SELECT STR_TO_DATE('13:30:17', '%H:%i');
```

```
01:30:00 PM
```

```
SELECT STR_TO_DATE('09:30:17 PM', '%r');
```

```
09:30:17 PM
```

```
SELECT STR_TO_DATE('0,0,0000', '%d,%m,%Y');
```

```
00/00/0000
```

다음은 시스템 파라미터 **intl_date_lang**의 값이 “de_DE”인 경우의 예이다. 독일어 Oktober가 10월로 해석된다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE";
SELECT STR_TO_DATE('3 Oktober 2009', '%d %M %Y');
```

```
10/03/2009
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 [TO_CHAR\(\)](#)의 **Note**를 참고한다.

다음은 타임존 정보를 포함하는 날짜/시간 문자열을 DATETIME TZ 타입으로 변환하는 예제이다.

```
SELECT STR_TO_DATE('2001-10-11 02:03:04 AM Europe/Bucharest EEST',
'%Y-%m-%d %h:%i:%s %p %TZR %TZD');
```

```
02:03:04.000 AM 10/11/2001 Europe/Bucharest EEST
```

TIME_FORMAT

TIME_FORMAT(*time*, *format*)

TIME_FORMAT 함수는 시간을 포함하는 날짜/시간 타입 값을 지정된 시간 형식의 문자열로 출력하며, 리턴 값은 **VARCHAR** 타입이다.

매개변수:

- **time** – 시간을 포함하는 타입(TIME, TIMESTAMP, DATETIME, TIMESTAMPTZ 또는 DATETIMETZ)의 값.
- **format** – 문자열 해석을 위한 형식을 지정한다. %를 포함하는 문자열을 형식 지정자(specifier)로 사용한다.
DATE_FORMAT() 함수의 **날짜/시간 형식 2** 표를 참고한다.

반환 형식:

STRING

format 인자가 지정되면 지정된 언어에 맞는 형식으로 날짜를 출력한다. 이때 언어는 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용된다. **intl_date_lang** 값이 지정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *format* 이 “%h:%i:%s %p”인 경우 “08:46:53 PM”인 시간을 “08:46:53 Nachm.”으로 출력한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

다음은 시스템 파라미터 **intl_date_lang** 의 값이 “en_US”인 경우의 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="en_US";
SELECT TIME_FORMAT(time'22:23:00', '%H %i %s');
```

```
'22 23 00'
```

```
SELECT TIME_FORMAT(time'23:59:00', '%H %h %i %s %f');
```

```
'23 11 59 00 000'
```

```
SELECT SYSTIME, TIME_FORMAT(SYSTIME, '%p');
```

```
08:46:53 PM 'PM'
```

다음은 시스템 파라미터 **intl_date_lang**의 값이 “de_DE”인 경우의 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE"';  
SELECT SYSTIME, TIME_FORMAT(SYSTIME, '%p');
```

```
08:46:53 PM 'Nachm.'
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 [TO_CHAR\(\)](#)의 **Note**를 참고한다.

다음은 타임존 정보를 포함하는 값을 명시한 포맷의 문자열로 출력하는 예이다.

```
SELECT TIME_FORMAT(datetimetz'2001-10-11 02:03:04 AM Europe/  
Bucharest EEST', '%h:%i:%s %p %TZR %TZD');
```

```
'02:03:04 AM Europe/Bucharest EEST'
```

TO_CHAR(date_time)

TO_CHAR(*date_time*[, *format*[, *date_lang_string_literal*]])

TO_CHAR (*date_time*) 함수는 날짜/시간 타입 값을 **날짜/시간 형식 1** 표에 따라 문자열로 변환하여 이를 반환하며, 리턴 값의 타입은 **VARCHAR** 이다.

매개변수:

- **date_time** - 날짜/시간 타입(TIME, DATE, TIMESTAMP, DATETIME, DATETIMETZ, DATETIMELTZ, TIMESTAMPTZ, TIMESTAMPLTZ)의 값.
- **format** - 리턴 값의 형식
- **date_lang_string_literal** - 리턴 값에 적용할 언어를 지정한다.

반환 형식:

STRING

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *date_time* 을 출력한다. 자세한 형식은 **날짜/시간 형식 1** 표를 참고하면 된다. 언어는 *date_lang_string_literal* 인자에 의해 정해진다. *date_lang_string_literal* 인자가 생략되면 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_date_lang** 값이 지정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *format*이 “HH:MI:SS AM”인 경우 “08:46:53 PM”인 시간을 “08:46:53 Nachm.”으로 출력한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 “en_US”의 기본 출력 형식을 따라 *date_time*을 문자열로 출력한다(아래 **날짜/시간 타입에 대한 언어별 기본 출력 형식** 표의 en_US 참고).

참고

CUBRID 9.0 미만 버전에서 사용되었던 **CUBRID_DATE_LANG** 환경 변수는 더 이상 사용되지 않는다.

날짜/시간 타입에 대한 언어별 기본 출력 형식

LANG	DATE	TIME	TIMESTAMP	DATETIME	TIMESTAMP WITH TIME ZONE	DATETIME WITH TIME ZONE
en_US	'MM/DD/YYYY'	'HH:MI:SS AM'		'HH:MI:SS. FF AM	'HH:MI:SS AM MM/	'HH:MI:SS. FF AM

LANG	DATE	TIME	TIMESTAMP	DATETIME	TIMESTAMP WITH TIME ZONE	DATETIME WITH TIME ZONE
			'HH:MI:SS AM MM/DD/YYYY'	MM/DD/YYYY'	DD/YYYY TZR'	MM/DD/YYYY TZR'
de_DE	'DD.MM.YY YY'	'HH24:MI:SS S'	'HH24:MI:SS S DD.MM.YY YY'	'HH24:MI:SS S.FF DD.MM.YY YY'	'HH24:MI:SS S DD.MM.YY YY TZR'	'HH24:MI:SS S.FF DD.MM.YY YY TZR'
es_ES	'DD.MM.YY YY'	'HH24:MI:SS S'	'HH24:MI:SS S DD.MM.YY YY'	'HH24:MI:SS S.FF DD.MM.YY YY'	'HH24:MI:SS S DD/MM/YYYY TZR'	'HH24:MI:SS S.FF DD/MM/YYYY TZR'
fr_FR	'DD.MM.YY YY'	'HH24:MI:SS S'	'HH24:MI:SS S DD.MM.YY YY'	'HH24:MI:SS S.FF DD.MM.YY YY'	'HH24:MI:SS S DD/MM/YYYY TZR'	'HH24:MI:SS S.FF DD/MM/YYYY TZR'
it_IT	'DD.MM.YY YY'	'HH24:MI:SS S'	'HH24:MI:SS S DD.MM.YY YY'	'HH24:MI:SS S.FF DD.MM.YY YY'	'HH24:MI:SS S DD/MM/YYYY TZR'	'HH24:MI:SS S.FF DD/MM/YYYY TZR'
ja_JP	'YYYY/MM/DD'	'HH24:MI:SS S'	'HH24:MI:SS S YYYY/MM/DD'	'HH24:MI:SS S.FF YYYY/MM/DD'	'HH24:MI:SS S YYYY/MM/DD TZR'	'HH24:MI:SS S.FF YYYY/MM/DD TZR'

LANG	DATE	TIME	TIMESTAMP	DATETIME	TIMESTAMP WITH TIME ZONE	DATETIME WITH TIME ZONE
km_KH	'DD/MM/YYYY'	'HH24:MI:SS'	'HH24:MI:SS DD/MM/YYYY'	'HH24:MI:SS.SF DD/MM/YYYY'	'HH24:MI:SS DD/MM/YYYY TZR'	'HH24:MI:SS.SF DD/MM/YYYY TZR'
ko_KR	'YYYY.MM.DD'	'HH24:MI:SS'	'HH24:MI:SS YYYY.MM.DD'	'HH24:MI:SS.SF YYYY.MM.DD'	'HH24:MI:SS YYYY.MM.DD TZR'	'HH24:MI:SS.SF YYYY.MM.DD TZR'
tr_TR	'DD.MM.YY YY'	'HH24:MI:SS'	'HH24:MI:SS DD.MM.YY YY'	'HH24:MI:SS.SF DD.MM.YY YY'	'HH24:MI:SS DD.MM.YY YY TZR'	'HH24:MI:SS.SF DD.MM.YY YY TZR'
vi_VN	'DD/MM/YYYY'	'HH24:MI:SS'	'HH24:MI:SS DD/MM/YYYY'	'HH24:MI:SS.SF DD/MM/YYYY'	'HH24:MI:SS DD/MM/YYYY TZR'	'HH24:MI:SS.SF DD/MM/YYYY TZR'
zh_CN	'YYYY-MM-DD'	'HH24:MI:SS'	'HH24:MI:SS YYYY-MM-DD'	'HH24:MI:SS.SF YYYY-MM-DD'	'HH24:MI:SS YYYY-MM-DD TZR'	'HH24:MI:SS.SF YYYY-MM-DD TZR'
ro_RO	'DD.MM.YY YY'	'HH24:MI:SS'	'HH24:MI:SS DD.MM.YY YY'	'HH24:MI:SS.SF DD.MM.YY YY'	'HH24:MI:SS DD.MM.YY YY TZR'	'HH24:MI:SS.SF DD.MM.YY YY TZR'

날짜/시간 형식 1

format 값	의미
CC	세기(Century)
YYYY, YY	4자리 연도, 2자리 연도
Q	분기(1, 2, 3, 4; 1월~3월 = 1)
MM	월(01-12; 1월 = 01) 참고: 분(<i>minute</i>)은 <i>MI</i> 이다.
MONTH	월 이름
MON	축약된 월 이름
DD	날(1-31)
DAY	요일 이름
DY	축약된 요일 이름
D 또는 d	요일(1-7)
AM 또는 PM	오전/오후
A.M. 또는 P.M.	마침표가 포함된 오전/오후
HH 또는 HH12	시(1-12)
HH24	시(0-23)
MI	분(0-59)

format 값	의미
SS	초(0-59)
FF	밀리초(0-999)
- / , . ; : “텍스트”	구두점과 인용구는 그대로 결과에 표현됨
TZD	일광 절약 정보(예: KST, KT, EET)
TZH	타임존의 시간 오프셋(예: +09, -09)
TZM	타임존의 분 오프셋 (예: 00, 30)

참고

TZR, TZD, TZH, TZM은 타임존 타입에서만 사용 가능하다.

참고

TZD 뒤에 숫자를 명시하는 포맷

TZD는 뒤에 숫자를 붙여서도 사용할 수 있다. TZD2~TZD11까지 사용할 수 있는데, 일반 문자를 문자열의 구분자로 사용하는 경우 숫자가 뒤따르는 포맷을 사용할 수 있다.

```
SELECT STR_TO_DATE('09:30:17 20140307XEESTXEurope/
Bucharest', '%h:%i:%s %Y%d%mX%TZD4X%TZR');
```

```
09:30:17.000 AM 07/03/2014 Europe/Bucharest EEST
```

위와 같이 각각의 값을 구분하기 위한 구분자로 일반 문자인 'X'를 사용하는 경우, TZD 값은 길이가 변할 수 있는 값이므로 TZD의 값과 구분자를 구분하기에 모호하다. 이런 경우 TZD의 길이를 명시하여야 한다.

date_lang_string_literal 예

date_lang_string_literal		
형식 구성 요소	'en_US'	'ko_KR'
MONTH	JANUARY	1월
MON	JAN	1
DAY	MONDAY	월요일
DY	MON	월
Month	January	1월
Mon	Jan	1
Day	Monday	월요일
Dy	Mon	월
month	january	1월
mon	jan	1
day	monday	월요일

형식 구성 요소	date_lang_string_literal	
	'en_US'	'ko_KR'
Dy	mon	월
AM	AM	오전
Am	Am	오전
am	am	오전
A.M.	A.M.	오전
A.m.	A.m.	오전
a.m.	a.m.	오전
PM	PM	오후
Pm	Pm	오후
pm	pm	오후
P.M.	P.M.	오후
P.m.	P.m.	오후
p.m.	p.m.	오후

리턴 값 형식의 자릿수의 예

형식 구성 요소	en_US 자릿수	ko_KR 자릿수
MONTH (Month, month)	9	4
MON (Mon, mon)	3	2
DAY (Day, day)	9	6
DY (Dy, dy)	3	2
HH12, HH24	2	2
“텍스트”	텍스트의 길이	텍스트의 길이
나머지 형식	주어진 형식의 길이와 같음	주어진 형식의 길이와 같음

다음은 언어 및 문자셋을 “en_US.iso88591”로 설정하여 생성한 데이터베이스에서 수행한 예이다.

```
-- create database testdb en_US.iso88591

--creating a table having date/time type columns
CREATE TABLE datetime_tbl(a TIME, b DATE, c TIMESTAMP, d DATETIME);
INSERT INTO datetime_tbl VALUES(SYSTIME, SYSDATE, SYSTIMESTAMP,
SYSDATETIME);

--selecting a VARCHAR type string from the data in the specified
format
SELECT TO_CHAR(b, 'DD, DY , MON, YYYY') FROM datetime_tbl;
```

```
'20, TUE , AUG, 2013'
```

```
SELECT TO_CHAR(c, 'HH24:MI, DD, MONTH, YYYY') FROM datetime_tbl;
```

```
'17:00, 20, AUGUST , 2013'
```

```
SELECT TO_CHAR(d, 'HH12:MI:SS:FF pm, YYYY-MM-DD-DAY') FROM
datetime_tbl;
```

```
'05:00:58:358 pm, 2013-08-20-TUESDAY '
```

```
SELECT TO_CHAR(TIMESTAMP'2009-10-04 22:23:00', 'Day Month yyyy');
```

```
'Sunday    October    2009'
```

다음은 위에서 생성한 데이터베이스에서 **TO_CHAR** 함수에 언어 인자를 별도로 부여한 예이다. 문자셋이 ISO-8859-1이면 **TO_CHAR** 함수의 언어 인자를 “tr_TR”과 “ko_KR”로 설정하는 것은 허용하나, 다른 언어는 허용하지 않는다. **TO_CHAR**의 언어 인자로 모든 언어를 사용 가능하게 하려면 데이터베이스 생성 시 문자셋이 UTF8이어야 한다.

```
SELECT TO_CHAR(TIMESTAMP'2009-10-04 22:23:00', 'Day Month
yyyy', 'ko_KR');
```

```
'Iryoil    10wol 2009'
```

```
SELECT TO_CHAR(TIMESTAMP'2009-10-04 22:23:00', 'Day Month
yyyy', 'tr_TR');
```

```
'Pazar    Ekim    2009'
```

참고

- 언어에 따라 월 이름, 일 이름, 요일 이름, 오전/오후 이름의 해석이 변경되는 함수에서 문자셋이 ISO-8859-1인 경우 “en_US” 외에 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다(위의 예 참고). 다만, 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 시스템 파라미터 **intl_date_lang**를 설정하거나 **TO_CHAR** 함수의 언어 인자를 지정하여 CUBRID가 지원하는 모든 언어(위 구문의 *date_lang_string_literal* 참고) 중 하나로 변경할 수 있다. 언어에 따라 날짜/시간 형식의 해석이 변경되는 함수들의 목록은 시스템 파라미터 **intl_date_lang**의 설명을 참고한다.

```
-- change date locale as "de_DE" and run the below query.
-- This case is failed because database locale, en_US's charset
```

```
is ISO-8859-1
-- and 'de_DE' only supports UTF-8 charset.
```

```
SELECT TO_CHAR(TIMESTAMP'2009-10-04 22:23:00', 'Day Month
yyyy', 'de_DE');
```

```
ERROR: before ' , 'Day Month yyyy','de_DE'); '
Locales for language 'de_DE' are not available with charset
'iso8859-1'.
```

다음은 DB 생성 시 로캘을 “en_US.utf8”로 설정하고 생성한 데이터베이스에서 **TO_CHAR** 함수에 언어 인자를 “de_DE”로 지정하고 실행한 예이다.

```
SELECT TO_CHAR(TIMESTAMP'2009-10-04 22:23:00', 'Day Month
yyyy', 'de_DE');
```

```
'Sonntag    Oktober 2009'
```

- 첫번째 인자가 zerodate이고 두번째 인자에 'Month', 'Day'와 같은 리터럴 형식이 지정되면 TO_CHAR 함수는 NULL을 반환한다.

```
SELECT TO_CHAR(timestamp '0000-00-00 00:00:00', 'Month Day
YYYY');
```

```
NULL
```

다음은 타임존 정보를 포함하는 날짜/시간 타입을 TO_CHAR 함수에서 출력하는 예이다.

형식을 정의하지 않으면 아래와 같은 순서로 출력된다.

```
SELECT TO_CHAR(datetimetz'2001-10-11 02:03:04 AM Europe/Bucharest
EEST');
```

```
'02:03:04.000 AM 10/11/2001 Europe/Bucharest EEST'
```

형식을 정의하면 정의한 순서대로 출력된다.

```
SELECT TO_CHAR(datetimetz'2001-10-11 02:03:04 AM Europe/Bucharest
EEST', 'MM/DD/YYYY HH24:MI TZR TZD TZH TZM');
```

```
'10/11/2001 02:03 Europe/Bucharest EEST +03 +00'
```

TO_CHAR(number)

TO_CHAR(*number*[, *format*[, *number_lang_string_literal*]])

TO_CHAR (number) 함수는 수치형 데이터 타입을 **숫자 형식**에 맞는 문자열로 변환하여 **VARCHAR** 타입으로 반환한다.

매개변수:

- **number** – 숫자를 반환하는 수치형 데이터 타입의 연산식을 지정한다. 입력값이 NULL이면 결과로 NULL이 반환된다. 입력값이 문자열 타입이면 해당 문자열을 그대로 반환한다.
- **format** – 리턴 값의 형식을 지정한다. 값이 **NULL**인 경우에는 **NULL**이 반환된다.
- **number_lang_string_literal** – 입력 숫자를 출력할 때 적용할 언어를 지정한다.

반환 형식:

STRING

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *number*를 출력한다. 이때 언어는 *number_lang_string_literal* 인자에 의해 정해진다. *number_lang_string_literal* 인자가 생략되면 **intl_number_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_number_lang** 값이 설정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”나 “fr_FR”과 같은 유럽 국가의 언어이면 “.”를 숫자의 자릿수 구분 기호로 출력하고 “,”를 소수점 기호로 출력한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 지정된 언어의 기본 출력에 따라 *number*를 문자열로 출력한다(**언어별 숫자의 기본 출력** 표 참고).

숫자 형식

형식 구성 요소	예제	설명
9	9999	“9”의 개수는 반환될 유효숫자 자릿수를 나타낸다. 숫자 인자에 대해 형식에서 지정된 유효숫자 자릿수가 부족하면, 소수부에 대해서는 반올림 연산을 수행한다. 숫자 인자의 정수부 자릿수보다 유효숫자 자릿수가 부족하면 #을 출력한다.
0	0999	형식에서 지정된 유효숫자 자릿수가 충분한 경우, 정수부 앞 부분을 공백이 아닌 0으로 채워 반환한다.
S	S9999	지정된 위치에 양수/음수 부호를 출력한다. 부호는 문자열의 시작부분에만 사용할 수 있다.
C	C9999	지정된 위치에 ISO 통화 기호를 반환한다.
, (쉼표)	9,999	지정된 위치에 쉼표(“,”)를 반환한다. 언어의 설정에 따라 쓰임이 달라지는데, 자릿수 구분 기호로 사용될 경우 여러 개가 허용되며, 소수점 기호로 사용될 경우 한 개만 허용된다. 언어별 숫자의 기본 출력 표 참고
. (마침표)	9.999	지정된 위치에 마침표를 출력한다. 언어의 설정에 따라 쓰임이 달라지는데, 자릿수 구분 기호로 사용될 경우 여러

형식 구성 요소	예제	설명
		개가 허용되며, 소수점 기호로 사용될 경우 한 개만 허용된다. 언어별 숫자의 기본 출력 표 참고

EEEE 9.99EEEE 과학적 기수법(scientific notation)을 반환한다.

언어별 숫자의 기본 출력

언어	로캘 이름	자릿수 구분 기호	소수점 기호	숫자 표기 예
영어	en_US	,(쉼표)	.(마침표)	123,456,789.012
독일어	de_DE	.(마침표)	,(쉼표)	123.456.789,012
스페인어	es_ES	.(마침표)	,(쉼표)	123.456.789,012
프랑스어	fr_FR	.(마침표)	,(쉼표)	123.456.789,012
이탈리아어	it_IT	.(마침표)	,(쉼표)	123.456.789,012
일본어	ja_JP	,(쉼표)	.(마침표)	123,456,789.012
캄보디아어	km_KH	.(마침표)	,(쉼표)	123.456.789,012
한국어	ko_KR	,(쉼표)	.(마침표)	123,456,789.012
터키어	tr_TR	.(마침표)	,(쉼표)	123.456.789,012

언어	로캘 이름	자릿수 구분 기호	소수점 기호	숫자 표기 예
베트남어	vi_VN	.(마침표)	,(십표)	123.456.789,012
중국어	zh_CN	,(십표)	.(마침표)	123,456,789.012
루마니아어	ro_RO	.(마침표)	,(십표)	123.456.789,012

다음은 DB 생성 시 로캘을 “en_US.utf8”로 설정하여 생성한 데이터베이스에서 수행한 예이다.

```
--selecting a string casted from a number in the specified format
SELECT TO_CHAR(12345,'S999999'), TO_CHAR(12345,'S099999');
```

```
' +12345 '          '+012345 '
```

```
SELECT TO_CHAR(1234567,'9,999,999,999');
```

```
' 1,234,567 '
```

```
SELECT TO_CHAR(1234567,'9.999.999.999');
```

```
'#####'
```

```
SELECT TO_CHAR(123.4567,'99'), TO_CHAR(123.4567,'999.99999'),
TO_CHAR(123.4567,'99999.999');
```

```
'##'          '123.45670'          ' 123.457 '
```

다음은 시스템 파라미터 **intl_number_lang**의 값을 “de_DE”로 설정하고 수행한 예이다. 독일, 프랑스 등 유럽 국가 대부분의 숫자 출력 형식은 “.”가 자릿수 구분 기호이고, “,”가 소수점 기호이다.

```
SET SYSTEM PARAMETERS 'intl_number_lang="de_DE";
```

```
--selecting a string casted from a number in the specified format
SELECT TO_CHAR(12345,'S999999'), TO_CHAR(12345,'S099999');
```

```
' +12345'                '+012345'
```

```
SELECT TO_CHAR(1234567,'9,999,999,999');
```

```
'#####'
```

```
SELECT TO_CHAR(1234567,'9.999.999.999');
```

```
'      1.234.567'
```

```
SELECT TO_CHAR(123.4567,'99'), TO_CHAR(123.4567,'999,99999'),
TO_CHAR(123.4567,'99999,999');
```

```
'##'                '123,45670'                '  123,457'
```

```
SELECT TO_CHAR(123.4567,'99','en_US'), TO_CHAR(123.
4567,'999.99999','en_US'), TO_CHAR(123.4567,'99999.999','en_US');
```

```
'##'                '123.45670'                '  123.457'
```

```
SELECT TO_CHAR(1.234567,'99.999EEEE','en_US'), TO_CHAR(1.
234567,'99,999EEEE','de_DE'), to_char(123.4567);
```

```
'1.235E+00'                '1,235E+00'                '123,4567'
```

TO_DATE

TO_DATE(*string*[, *format*[, *date_lang_string_literal*]])

TO_DATE 함수는 인자로 지정된 날짜 형식을 기준으로 문자열을 해석하여, 이를 **DATE** 타입의 값으로 변환하여 반환한다. 날짜 형식은 **날짜/시간 형식 1**을 참고한다.

매개변수:

- **string** – 문자열
- **format** – **DATE** 타입으로 변환할 값의 형식을 지정하며, **날짜/시간 형식 1** 표를 참고한다. 값이 **NULL**이면 결과로 **NULL** 이 반환된다.
- **date_lang_string_literal** – 입력 값에 적용할 언어를 지정한다.

반환 형식:

DATE

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *string* 을 해석한다. 이때 언어는 *date_lang_string_literal* 인자에 의해 정해진다. *date_lang_string_literal* 인자가 생략되면 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_date_lang** 값의 설정이 생략되면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *string* 이 “12.mai.2012”이고 *format* 이 “DD.mon.YYYY”인 경우 “2012년 5월 12일”로 해석한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 먼저 CUBRID 기본 형식(**날짜/시간 타입 문자열 권장 형식** 참고)에 따라 *string*을 해석하고, 실패하는 경우 **intl_date_lang**에 의해 설정된 언어의 기본 출력 형식(**날짜/시간 타입에 대한 언어별 기본 출력 형식** 표 참고)에 따라 *string*을 해석한다. **intl_date_lang**이 설정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 **DATE** 타입에 대해 허용하는 문자열은 CUBRID 기본 형식인 “MM/DD/YYYY”과 “de_DE” 기본 형식인 “DD.MM.YYYY”이다.

다음은 DB 생성 시 로캘을 “en_US.utf8”로 설정하여 수행하는 예이다.

```
--selecting a date type value casted from a string in the specified format
```

```
SELECT TO_DATE('12/25/2008');
```

```
12/25/2008
```

```
SELECT TO_DATE('25/12/2008', 'DD/MM/YYYY');
```

```
12/25/2008
```

```
SELECT TO_DATE('081225', 'YYMMDD');
```

```
12/25/2008
```

```
SELECT TO_DATE('2008-12-25', 'YYYY-MM-DD');
```

```
12/25/2008
```

다음은 `intl_date_lang` 의 값이 “de_DE”일 때 `TO_DATE`를 수행하는 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE"';
SELECT TO_DATE('25.12.2012');
```

```
12/25/2012
```

```
SELECT TO_DATE('12/mai/2012','dd/mon/yyyy', 'de_DE');
```

```
05/12/2012
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 `intl_date_lang`에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 `TO_CHAR()` 의 **Note**를 참고한다.

TO_DATETIME

`TO_DATETIME(string[, format[, date_lang_string_literal]])`

TO_DATETIME 함수는 인자로 지정된 **DATETIME** 형식을 기준으로 문자열을 해석하여, 이를 **DATETIME** 타입의 값으로 변환하여 반환한다. **DATETIME** 형식은 `TO_CHAR()` 함수의 **날짜/시간 형식 1**을 참고한다.

매개변수:

- **string** – 문자열
- **format** – DATETIME 타입으로 변환할 값의 형식을 지정하며, **날짜/시간 형식 1**을 참고한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **date_lang_string_literal** – 입력 값에 적용할 언어를 지정한다.

반환 형식:

DATETIME

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *string* 을 해석한다.

예를 들어 언어가 “de_DE”일 때 *string*이 “12/mai/2012 12:10:00 Nachm.”이고 *format*이 “DD/MON/YYYY HH:MI:SS AM”인 경우 “2012년 5월 12일 오후 12시 10분 0초”로 해석한다. 이때 언어는 *date_lang_string_literal* 인자에 의해 정해진다. *date_lang_string_literal* 인자가 생략되면 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_date_lang** 값의 설정이 생략되면 DB 생성 시 지정한 언어가 적용된다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 먼저 CUBRID 기본 형식(**날짜/시간 타입 문자열 권장 형식** 참고)에 따라 *string*을 해석하고, 실패하는 경우 **intl_date_lang**에 의해 설정된 언어의 기본 출력 형식(**날짜/시간 타입에 대한 언어별 기본 출력 형식** 표 참고)에 따라 *string*을 해석한다. **intl_date_lang**이 설정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 **DATETIME** 타입에 대해 허용하는 문자열은 CUBRID 기본 형식인 “HH:MI:SS.FF AM MM/DD/YYYY”와 “de_DE” 기본 형식인 “HH24:MI:SS.FF DD.MM.YYYY”이다.

참고

CUBRID 9.0 미만 버전에서 사용되었던 **CUBRID_DATE_LANG** 환경 변수는 더 이상 사용되지 않는다.

다음은 DB 생성 시 로캘을 “en_US”로 설정하여 생성된 데이터베이스에서 수행하는 예이다.

```
--selecting a datetime type value casted from a string in the
specified format
```

```
SELECT TO_DATETIME('13:10:30 12/25/2008');
```

```
01:10:30.000 PM 12/25/2008
```

```
SELECT TO_DATETIME('08-Dec-25 13:10:30.999', 'YY-Mon-DD  
HH24:MI:SS.FF');
```

```
01:10:30.999 PM 12/25/2008
```

```
SELECT TO_DATETIME('DATE: 12-25-2008 TIME: 13:10:30.999', '"DATE:"  
MM-DD-YYYY "TIME:" HH24:MI:SS.FF');
```

```
01:10:30.999 PM 12/25/2008
```

다음은 **intl_date_lang** 의 값이 “de_DE”일 때 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE";  
SELECT TO_DATETIME('13:10:30.999 25.12.2012');
```

```
01:10:30.999 PM 12/25/2012
```

```
SELECT TO_DATETIME('12/mai/2012 12:10:00 Nachm.', 'DD/MON/YYYY  
HH:MI:SS AM', 'de_DE');
```

```
12:10:00.000 PM 05/12/2012
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 **TO_CHAR()** 의 **Note**를 참고한다.

TO_DATETIME_TZ

TO_DATETIME_TZ(*string*[, *format*[, *date_lang_string_literal*]])

TO_DATETIME_TZ 함수는 입력 문자열에 타임존 정보를 포함할 수 있다는 점을 제외하고는 **TO_DATETIME()** 함수와 동일하다.

반환 형식:

DATETIME_TZ

```
SELECT TO_DATETIME_TZ('13:10:30 Asia/Seoul 12/25/2008', 'HH24:MI:SS
TZR MM/DD/YYYY');
```

```
01:10:30.000 PM 12/25/2008 Asia/Seoul
```

TO_NUMBER

TO_NUMBER(*string*[, *format*])

TO_NUMBER 함수는 인자로 지정된 숫자 형식을 기준으로 문자열을 해석하여, 이를 **NUMERIC** 타입으로 변환하여 반환한다.

매개변수:

- **string** – 문자열을 반환하는 임의의 연산식이다. 값이 NULL 이면 결과로 NULL이 반환된다.
- **format** – 숫자로 반환할 값의 형식을 지정하며, **숫자 형식** 표를 참고한다. 생략되면 NUMERIC(38,0) 값이 리턴된다.

반환 형식:

NUMERIC

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *string* 을 해석한다. 이때 언어는 **intl_number_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_number_lang** 값의 설정이 생략 되면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”나 “fr_FR”과 같은 유럽 국가의 언어이면 “.”를 숫자의 자릿수 구분 기호로 해석하고 “,”를 소수점 기호로 해석한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 **intl_number_lang**에 의해 설정된 언어의 기본 출력 형식을 따라 *string* 을 해석한다(**언어별 숫자의 기본 출력** 표 참고). **intl_number_lang**이 설정되지 않으면 DB 생성 시 지정된 언어가 적용된다.

다음은 시스템 파라미터 **intl_number_lang**의 값이 “en_US”일 때 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_number_lang="en_US";
--selecting a number casted from a string in the specified format
SELECT TO_NUMBER('-1234');
```

```
-1234
```

```
SELECT TO_NUMBER('12345','999999');
```

```
12345
```

```
SELECT TO_NUMBER('12,345.67','99,999.999');
```

```
12345.670
```

```
SELECT TO_NUMBER('12345.67','99999.999');
```

```
12345.670
```

다음은 시스템 파라미터 **intl_number_lang**의 값을 “de_DE”로 설정하여 실행한 예이다. 독일, 프랑스 등 유럽 국가에서는 숫자의 자릿수 구분 기호로 마침표가 사용되며, 소수점 기호로 쉼표가 사용된다.

```
SET SYSTEM PARAMETERS 'intl_number_lang="de_DE";
SELECT TO_NUMBER('12.345,67','99.999,999');
```

```
12345.670
```

TO_TIME

TO_TIME(*string*[, *format*[, *date_lang_string_literal*]])

TO_TIME 함수는 인자로 지정된 시간 형식을 기준으로 문자열을 해석하여, 이를 TIME 타입의 값으로 변환하여 반환한다. 시간 형식은 **날짜/시간 형식 1**을 참고한다.

매개변수:

- **string** - 문자열을 반환하는 임의의 연산식이다. 값이 NULL 이면 결과로 NULL이 반환된다.
- **format** - TIME 타입으로 변환할 값의 형식을 지정하며, **날짜/시간 형식 1** 표를 참고한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **date_lang_string_literal** - 입력 값에 적용할 언어를 지정한다.

반환 형식:

TIME

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *string* 을 해석한다. 이때 언어는 *date_lang_string_literal* 인자에 의해 정해진다. *date_lang_string_literal* 인자가 생략되면 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_date_lang** 값의 설정이 생략되면 DB 생성 시 지정한 언어가 적용된다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

예를 들어 언어가 “de_DE”일 때 *string* 이 “10:23:00 Nachm.”이고 *format* 이 “HH:MI:SS AM”인 경우 “오후 10시 23분 0초”로 해석한다.

format 인자가 생략되면 먼저 CUBRID 기본 형식(**날짜/시간 타입 문자열 권장 형식** 참고)에 따라 *string*을 해석하고, 실패하는 경우 **intl_date_lang**에 의해 설정된 언어의 기본 출력 형식(**날짜/시간 타입에 대한 언어별 기본 출력 형식** 표 참고)에 따라 *string*을 해석한다. **intl_date_lang**이 설정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 **TIME** 타입에 대해 허용하는 문자열은 CUBRID 기본 형식인 “HH:MI:SS AM”과 “de_DE” 기본 형식인 “HH24:MI:SS”이다.

 참고

CUBRID 9.0 미만 버전에서 사용되었던 **CUBRID_DATE_LANG** 환경 변수는 더 이상 사용되지 않는다.

다음은 시스템 파라미터 **intl_date_lang** 의 값이 “en_US”일 때 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="en_US";

--selecting a time type value casted from a string in the specified
format

SELECT TO_TIME ('13:10:30');
```

```
01:10:30 PM
```

```
SELECT TO_TIME('HOUR: 13 MINUTE: 10 SECOND: 30', '"HOUR:" HH24
"MINUTE:" MI "SECOND:" SS');
```

```
01:10:30 PM
```

```
SELECT TO_TIME ('13:10:30', 'HH24:MI:SS');
```

```
01:10:30 PM
```

```
SELECT TO_TIME ('13:10:30', 'HH12:MI:SS');
```

```
ERROR: Conversion error in date format.
```

다음은 **intl_date_lang** 의 값이 “de_DE”일 때 수행하는 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE";
SELECT TO_TIME('13:10:30');
```

```
01:10:30 PM
```

```
SELECT TO_TIME('10:23:00 Nachm.', 'HH:MI:SS AM');
```

```
10:23:00 PM
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 [TO_CHAR\(\)](#) 의 **Note**를 참고한다.

TO_TIMESTAMP

TO_TIMESTAMP(*string*[, *format*[, *date_lang_string_literal*]])

TO_TIMESTAMP 함수는 인자로 지정된 타임스탬프 형식을 기준으로 문자열을 해석하여, 이를 **TIMESTAMP** 타입의 값으로 변환하여 반환한다. 타임스탬프 형식은 **날짜/시간 형식 1**을 참고한다.

매개변수:

- **string** – 문자열을 반환하는 임의의 연산식이다. 값이 NULL 이면 결과로 NULL이 반환된다.
- **format** – **TIMESTAMP** 타입으로 변환할 값의 형식을 지정하며, **날짜/시간 형식 1** 표를 참고한다. 값이 **NULL** 이면 결과로 **NULL** 이 반환된다.
- **date_lang_string_literal** – 입력 값에 적용할 언어를 지정한다.

반환 형식:

TIMESTAMP

format 인자가 지정되면 지정한 언어에 맞는 형식으로 *string* 을 해석한다. 이때 언어는 *date_lang_string_literal* 인자에 의해 정해진다. *date_lang_string_literal* 인자가 생략되면 **intl_date_lang** 시스템 파라미터에 지정한 언어가 적용되며, **intl_date_lang** 값의 설정이 생략되면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 *string* 이 “12/mai/2012 12:10:00 Nachm.”이고 *format* 이 “DD/MON/YYYY HH:MI:SS AM”인 경우 “2012년 5월 12일 오후 12시 10분 0초”로 해석한다. 주어진 문자열과 대응하지 않는 *format* 인자가 지정되면 에러를 반환한다.

format 인자가 생략되면 먼저 CUBRID 기본 형식([날짜/시간 타입 문자열 권장 형식](#) 참고)에 따라 *string*을 해석하고, 실패하는 경우 **intl_date_lang**에 의해 설정된 언어의 기본 출력 형식([날짜/시간 타입에 대한 언어별 기본 출력 형식](#) 표 참고)에 따라 *string*을 해석한다. **intl_date_lang**이 설정되지 않으면 DB 생성 시 지정한 언어가 적용된다.

예를 들어 언어가 “de_DE”일 때 **TIMESTAMP** 타입에 대해 허용하는 문자열은 CUBRID 기본 형식인 “HH:MI:SS AM MM/DD/YYYY”와 “de_DE” 기본 형식인 “HH24:MI:SS DD.MM.YYYY”이다.

다음은 시스템 파라미터 **intl_date_lang**의 값이 “en_US”일 때 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="en_US";
```

```
--selecting a timestamp type value casted from a string in the  
specified format
```

```
SELECT TO_TIMESTAMP('13:10:30 12/25/2008');
```

```
01:10:30 PM 12/25/2008
```

```
SELECT TO_TIMESTAMP('08-Dec-25 13:10:30', 'YY-Mon-DD HH24:MI:SS');
```

```
01:10:30 PM 12/25/2008
```

```
SELECT TO_TIMESTAMP('YEAR: 2008 DATE: 12-25 TIME: 13:10:30',  
'"YEAR:" YYYY "DATE:" MM-DD "TIME:" HH24:MI:SS');
```

```
01:10:30 PM 12/25/2008
```

다음은 **intl_date_lang**의 값이 “de_DE”일 때 수행한 예이다.

```
SET SYSTEM PARAMETERS 'intl_date_lang="de_DE";
```

```
SELECT TO_TIMESTAMP('13:10:30 25.12.2008');
```

```
01:10:30 PM 12/25/2008
```

```
SELECT TO_TIMESTAMP('10:23:00 Nachm.', 'HH12:MI:SS AM');
```

```
10:23:00 PM 08/01/2012
```

참고

문자셋이 ISO-8859-1인 경우 “en_US” 외에 시스템 파라미터 **intl_date_lang**에 의해 변경할 수 있는 언어는 “ko_KR”과 “tr_TR”뿐이다. 문자셋이 UTF-8인 경우 CUBRID가 지원하는 모든 언어 중 하나로 변경할 수 있다. 보다 자세한 설명은 [TO_CHAR\(\)](#)의 **Note**를 참고한다.

TO_TIMESTAMP_TZ

TO_TIMESTAMP_TZ(string[, format[, date_lang_string_literal]])

TO_TIMESTAMP_TZ 함수는 입력 문자열에 타임존 정보를 포함할 수 있다는 점을 제외하고는 [TO_TIMESTAMP\(\)](#) 함수와 동일하다.

rtype:

TIMESTAMPTZ

```
SELECT TO_TIMESTAMP_TZ('13:10:30 Asia/Seoul 12/25/2008',
'HH24:MI:SS TZR MM/DD/YYYY');
```

```
01:10:30 PM 12/25/2008 Asia/Seoul
```

집계/분석 함수

목차

집계/분석 함수

7

- 개요 10
- 집계 함수와 분석 함수 비교 10

● OVER 함수 내에 “ORDER BY” 절을 명시해야 하는 분석 함수	624
● AVG	624
● COUNT	627
● CUME_DIST	629
● DENSE_RANK	633
● FIRST_VALUE	635
● GROUP_CONCAT	637
● LAG	638
● LAST_VALUE	640
● LEAD	641
● MAX	643
● MEDIAN	645
● MIN	646
● NTH_VALUE	647
● NTILE	649
● PERCENT_RANK	651
● PERCENTILE_CONT	656
● PERCENTILE_DISC	659
● RANK	661
● ROW_NUMBER	663
● STDDEV, STDDEV_POP	664

● STDDEV_SAMP	666
● SUM	669
● VARIANCE, VAR_POP	671
● VAR_SAMP	673
● JSON_ARRAYAGG	675
● JSON_OBJECTAGG	676

개요

집계/분석 함수는 데이터를 분석하여 어떤 결과를 추출하고자 할 때 사용하는 함수이다.

- 집계 함수는 그룹 별로 그룹핑된 결과를 리턴하며, 그룹핑 대상이 되는 칼럼만 출력한다.
- 분석 함수는 그룹 별로 그룹핑된 결과를 리턴하되, 그룹핑되지 않은 칼럼을 포함하여 하나의 그룹에 대해 여러 개의 행을 출력할 수 있다.

예를 들어 집계/분석 함수는 다음과 같은 질문에 대한 답을 구하기 위해 사용될 수 있다.

1. 연도별 총 판매 금액은 어떻게 되는가?
2. 연도별로 그룹지어 가장 판매 금액이 높은 월부터 순서대로 출력하려면 어떻게 하는가?
3. 연도별로 그룹지어 연도별, 월별 순서대로 누적 판매 금액을 출력하려면 어떻게 하는가?

1.은 집계 함수로 답을 구할 수 있으며, 2., 3.은 분석 함수로 답을 구할 수 있다. 위의 질문들은 다음의 SQL문으로 작성될 수 있다.

다음은 각 년도의 월별 판매 금액을 저장하고 있는 테이블이다.

```
CREATE TABLE sales_mon_tbl (
    yyyy INT,
    mm INT,
    sales_sum INT
);

INSERT INTO sales_mon_tbl VALUES
    (2000, 1, 1000), (2000, 2, 770), (2000, 3, 630), (2000, 4, 890),
    (2000, 5, 500), (2000, 6, 900), (2000, 7, 1300), (2000, 8, 1800),
    (2000, 9, 2100), (2000, 10, 1300), (2000, 11, 1500), (2000, 12,
    1610),
    (2001, 1, 1010), (2001, 2, 700), (2001, 3, 600), (2001, 4, 900),
    (2001, 5, 1200), (2001, 6, 1400), (2001, 7, 1700), (2001, 8,
```



```
1110),
  (2001, 9, 970), (2001, 10, 690), (2001, 11, 710), (2001, 12,
880),
  (2002, 1, 980), (2002, 2, 750), (2002, 3, 730), (2002, 4, 980),
  (2002, 5, 1110), (2002, 6, 570), (2002, 7, 1630), (2002, 8,
1890),
  (2002, 9, 2120), (2002, 10, 970), (2002, 11, 420), (2002, 12,
1300);
```

1. 연도별 총 판매 금액은 어떻게 되는가?

```
SELECT yyyy, sum(sales_sum)
FROM sales_mon_tbl
GROUP BY yyyy;
```

yyyy	sum(sales_sum)
2000	14300
2001	11870
2002	13450

1. 연도별로 그룹지어 가장 판매 금액이 높은 월부터 순서대로 출력하려면 어떻게 하는가?

```
SELECT yyyy, mm, sales_sum, RANK() OVER (PARTITION BY yyyy ORDER BY
sales_sum DESC) AS rnk
FROM sales_mon_tbl;
```

yyyy	mm	sales_sum	rnk
2000	9	2100	1
2000	8	1800	2
2000	12	1610	3
2000	11	1500	4
2000	7	1300	5
2000	10	1300	5
2000	1	1000	7
2000	6	900	8
2000	4	890	9
2000	2	770	10
2000	3	630	11
2000	5	500	12
2001	7	1700	1
2001	6	1400	2
2001	5	1200	3
2001	8	1110	4
2001	1	1010	5

2001	9	970	6
2001	4	900	7
2001	12	880	8
2001	11	710	9
2001	2	700	10
2001	10	690	11
2001	3	600	12
2002	9	2120	1
2002	8	1890	2
2002	7	1630	3
2002	12	1300	4
2002	5	1110	5
2002	1	980	6
2002	4	980	6
2002	10	970	8
2002	2	750	9
2002	3	730	10
2002	6	570	11
2002	11	420	12

1. 연도별로 그룹지어 연도별, 월별 순서대로 누적 판매 금액을 출력하려면 어떻게 하는가?

```
SELECT yyyy, mm, sales_sum, SUM(sales_sum) OVER (PARTITION BY yyyy
ORDER BY yyyy, mm) AS a_sum
FROM sales_mon_tbl;
```

yyyy	mm	sales_sum	a_sum
2000	1	1000	1000
2000	2	770	1770
2000	3	630	2400
2000	4	890	3290
2000	5	500	3790
2000	6	900	4690
2000	7	1300	5990
2000	8	1800	7790
2000	9	2100	9890
2000	10	1300	11190
2000	11	1500	12690
2000	12	1610	14300
2001	1	1010	1010
2001	2	700	1710
2001	3	600	2310
2001	4	900	3210
2001	5	1200	4410
2001	6	1400	5810
2001	7	1700	7510
2001	8	1110	8620
2001	9	970	9590

2001	10	690	10280
2001	11	710	10990
2001	12	880	11870
2002	1	980	980
2002	2	750	1730
2002	3	730	2460
2002	4	980	3440
2002	5	1110	4550
2002	6	570	5120
2002	7	1630	6750
2002	8	1890	8640
2002	9	2120	10760
2002	10	970	11730
2002	11	420	12150
2002	12	1300	13450

집계 함수와 분석 함수 비교

집계 함수(aggregate functions)는 행들의 그룹에 기반하여 각 그룹 당 하나의 결과를 반환한다. **GROUP BY** 절을 포함하면 각 그룹마다 한 행의 집계 결과를 반환한다. **GROUP BY** 절을 생략하면 전체 행에 대해 한 행의 집계 결과를 반환한다. **HAVING** 절은 **GROUP BY** 절이 있는 질의에 조건을 추가할 때 사용한다.

대부분의 집계 함수는 **DISTINCT**, **UNIQUE** 제약 조건을 사용할 수 있다. **GROUP BY ... HAVING** 절에 대해서는 **GROUP BY ... HAVING 절** 을 참고한다.

분석 함수(analytic functions)는 행들의 결과에 기반하여 집계 값을 계산한다. 분석 함수는 **OVER** 절 뒤의 *<partition_by_clause>*에 의해 지정된 그룹들(이 절이 생략되면 모든 행을 하나의 그룹으로 봄)을 기준으로 한 개 이상의 행을 반환할 수 있다는 점에서 집계 함수와 다르다.

분석 함수는 특정 행 집합에 대해 다양한 통계를 허용하기 위해 기존의 집계 함수들 일부에 **OVER** 라는 새로운 분석 절이 함께 사용된다.

```
function_name ([<argument_list>]) OVER (<analytic_clause>)

<analytic_clause>::=
    [<partition_by_clause>] [<order_by_clause>]

<partition_by_clause>::=
    PARTITION BY value_expr[, value_expr]...

<order_by_clause>::=
    ORDER BY { expression | position | column_alias } [ ASC | DESC ]
```

```
[, { expression | position | column_alias } [ ASC |
DESC ] ] ...
```

- *<partition_by_clause>*: 하나 이상의 *value_expr* 에 기반한 그룹들로, 질의 결과를 분할하기 위해 **PARTITION BY** 절을 사용한다.
- *<order_by_clause>*: *<partition_by_clause>*에 의한 분할(partition) 내에서 데이터의 정렬 방식을 명시한다. 여러 개의 키로 정렬할 수 있다. *<partition_by_clause>*가 생략될 경우 전체 결과 셋 내에서 데이터를 정렬한다. 정렬된 순서에 의해 앞의 값을 포함하여 누적한 레코드의 칼럼 값을 대상으로 함수를 적용하여 계산한다.

분석 함수의 OVER 절 뒤에 함께 사용되는 ORDER BY/PARTITION BY 절의 표현식에 따른 동작 방식은 다음과 같다.

- ORDER BY/PARTITION BY <상수가 아닌 표현식> (예: i, sin(i+1)): 표현식은 정렬/분할(ordering/partitioning)에 사용됨.
- ORDER BY/PARTITION BY <상수> (예: 1): 상수는 SELECT 리스트의 칼럼 위치로 간주됨.
- ORDER BY/PARTITION BY <상수 표현식> (예: 1+0): 상수 표현식은 무시되어, 정렬/분할(ordering/partitioning)에 사용되지 않음.

OVER 함수 내에 “ORDER BY” 절을 명시해야 하는 분석 함수

다음 분석 함수들은 순서가 필요하므로 OVER 함수 내에 “ORDER BY” 절을 명시해야 하는 분석 함수들이다. “ORDER BY” 절이 생략되는 경우 오류가 발생하거나 출력 결과에 대해 정확한 순서를 보장하지 않는다는 점에 주의한다.

- CUME_DIST()
- DENSE_RANK()
- LAG()
- LEAD()
- NTILE()
- PERCENT_RANK()
- RANK()
- ROW_NUMBER()

AVG

AVG([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression)

AVG([*DISTINCT* | *DISTINCTROW* | *UNIQUE* | *ALL*] *expression*) *OVER* (<*analytic_clause*>)

AVG 함수는 집계 함수 또는 분석 함수로 사용되며, 모든 행에 대한 연산식 값의 산술 평균을 구한다. 하나의 연산식 *expression* 만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후 평균을 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해서 평균을 구한다.

param expression:

수치 값을 반환하는 임의의 연산식을 지정한다. 컬렉션 타입의 데이터를 반환하는 연산식은 지정될 수 없다.

param ALL:

모든 값에 대해 평균을 구하기 위해 사용되며, 기본값이다.

param

DISTINCT, **DISTINCTROW**, **UNIQUE**:

중복이 제거된 유일한 값에 대해서만 평균을 구하기 위해 사용된다.

rtype:

DOUBLE

다음은 *demodb* 에서 한국이 획득한 금메달의 평균 수를 반환하는 예제이다.

```
SELECT AVG(gold)
FROM participant
WHERE nation_code = 'KOR';
```

```
          avg(gold)
=====
          9.600000000000000e+00
```

다음은 *demodb* 에서 *nation_code*가 'AU'로 시작하는 국가에 대해 연도 별로 획득한 금메달 수와 해당 연도까지의 금메달 누적에 대한 평균 합계를 출력하는 예제이다.

```
SELECT host_year, nation_code, gold,
       AVG(gold) OVER (PARTITION BY nation_code ORDER BY host_year)
       avg_gold
FROM participant WHERE nation_code like 'AU%';
```

host_year	nation_code	gold	avg_gold
1988	'AUS'	3	3.0000000000000000e+00
1992	'AUS'	7	5.0000000000000000e+00
1996	'AUS'	9	6.333333333333333e+00
2000	'AUS'	16	8.7500000000000000e+00
2004	'AUS'	17	1.0400000000000000e+01
1988	'AUT'	1	1.0000000000000000e+00
1992	'AUT'	0	5.0000000000000000e-01
1996	'AUT'	0	3.333333333333333e-01
2000	'AUT'	2	7.5000000000000000e-01
2004	'AUT'	2	1.0000000000000000e+00

다음은 위 예제에서 **OVER** 분석 절 이하의 “ORDER BY host_year” 절을 제거한 것으로, avg_gold의 값은 모든 연도의 금메달 평균으로 nation_code별로 각 연도에서 모두 같은 값을 가진다.

```
SELECT host_year, nation_code, gold, AVG(gold) OVER (PARTITION BY
nation_code) avg_gold
FROM participant WHERE nation_code LIKE 'AU%';
```

host_year	nation_code	gold	avg_gold
2004	'AUS'	17	1.0400000000000000e+01
2000	'AUS'	16	1.0400000000000000e+01
1996	'AUS'	9	1.0400000000000000e+01

```

1992 'AUS' 7
1.0400000000000000e+01
1988 'AUS' 3
1.0400000000000000e+01
2004 'AUT' 2
1.0000000000000000e+00
2000 'AUT' 2
1.0000000000000000e+00
1996 'AUT' 0
1.0000000000000000e+00
1992 'AUT' 0
1.0000000000000000e+00
1988 'AUT' 1
1.0000000000000000e+00

```

COUNT

COUNT(*)

COUNT(*) OVER (<analytic_clause>)

COUNT([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression)

COUNT([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression) OVER (<analytic_clause>)

COUNT 함수는 집계 함수 또는 분석 함수로 사용되며, 질의문이 반환하는 결과 행들의 개수를 반환한다. 별표(*)를 지정하면 조건을 만족하는 모든 행(**NULL** 값을 가지는 행 포함)의 개수를 반환하며, **DISTINCT** 또는 **UNIQUE** 키워드를 연산식 앞에 지정하면 중복을 제거한 후 유일한 값을 가지는 행(**NULL** 값을 가지는 행은 포함하지 않음)의 개수만 반환한다. 따라서, 반환되는 값은 항상 큰 정수 타입이며, **NULL** 은 반환되지 않는다.

param expression:

임의의 연산식이다.

param ALL:

주어진 expression의 모든 행의 개수를 구하기 위해 사용되며, 기본값이다.

param

DISTINCT,DISTINCTROW,UNIQUE:

중복이 제거된 유일한 값을 가지는 행의 개수를 구하기 위해 사용된다.

rtype:

BIGINT

연산식 *expression* 은 수치형 또는 문자열 타입은 물론, 컬렉션 타입 칼럼과 오브젝트 도메인(사용자 정의 클래스)을 가지는 칼럼도 지정될 수 있다.

다음은 *demodb* 에서 역대 올림픽 중에서 마스코트가 존재했었던 올림픽의 수를 반환하는 예제이다.

```
SELECT COUNT(*)
FROM olympic
WHERE mascot IS NOT NULL;
```

```
count(*)
=====
9
```

다음은 *demodb* 에서 *nation_code*가 'AUT'인 국가의 참가 선수의 종목(event)별 인원 수를 종목이 바뀔 때마다 누적하여 출력한 예제이다. 가장 마지막 줄에는 모든 인원 수가 출력된다.

```
SELECT nation_code, event, name, COUNT(*) OVER (ORDER BY event) co
FROM athlete WHERE nation_code='AUT';
```

nation_code	event	co
name		
=====		
'AUT'	'Athletics'	'Kiesl
Theresia'	2	
'AUT'	'Athletics'	'Graf
Stephanie'	2	
'AUT'	'Equestrian'	'Boor
Boris'	6	
'AUT'	'Equestrian'	'Fruhmann
Thomas'	6	
'AUT'	'Equestrian'	'Munzner
Joerg'	6	
'AUT'	'Equestrian'	'Simon
Hugo'	6	
'AUT'	'Judo'	'Heill


```

Claudia'          9
  'AUT'          'Judo'          'Seisenbacher
Peter'            9
  'AUT'          'Judo'          'Hartl
Roswitha'        9
  'AUT'          'Rowing'        'Jonke
Arnold'          11
  'AUT'          'Rowing'        'Zerbst
Christoph'       11
  'AUT'          'Sailing'       'Hagara
Roman'           15
  'AUT'          'Sailing'       'Steinacher Hans
Peter'           15
  'AUT'          'Sailing'       'Sieber
Christoph'       15
  'AUT'          'Sailing'       'Geritzer
Andreas'         15
  'AUT'          'Shooting'      'Waibel Wolfram
Jr.'             17
  'AUT'          'Shooting'      'Planer
Christian'       17
  'AUT'          'Swimming'      'Rogan
Markus'          18

```

CUME_DIST

`CUME_DIST(expression[, expression] ...) WITHIN GROUP (<order_by_clause>)`

`CUME_DIST() OVER ([<partition_by_clause>] <order_by_clause>)`

CUME_DIST 함수는 집계 함수 또는 분석 함수로 사용되며, 그룹의 값 내에서 명시한 값의 누적 분포 값을 반환한다. **CUME_DIST**에 의해 반환되는 값의 범위는 0보다 크고 1보다 작거나 같다. 같은 값의 입력 인자에 대한 **CUME_DIST** 함수의 반환 값은 항상 같은 누적 분포 값으로 평가된다.

param expression:

수치 또는 문자열을 반환하는 연산식. 칼럼이 올 수 없다.

param order_by_clause:

ORDER BY 절 뒤에 오는 칼럼 이름은 *expression* 개수만큼 매핑되어야 한다.

rtype:

DOUBLE

❗ 더 보기

`PERCENT_RANK()` , `CUME_DIST`와 `PERCENT_RANK` 비교

집계 함수인 경우, **CUME_DIST** 함수는 **ORDER BY** 절에 명시된 순서로 정렬한 후, 집계 그룹에 있는 행에서 가상(hypothetical) 행의 상대적인 위치를 반환한다. 이때, 가상 행이 새로 입력되는 것으로 간주하고 위치를 계산한다. 즉, (“어떤 행의 누적된 RANK” + 1)/ (“집계 그룹 전체 행의 개수” + 1)을 반환한다.

분석 함수인 경우, **PARTITION BY**에 의해 나누어진 그룹별로 각 행을 **ORDER BY** 절에 명시된 순서로 정렬한 후 그룹 내 값의 상대적인 위치를 반환한다. 상대적인 위치는 입력 인자 값보다 작거나 같은 값을 가진 행의 개수를 그룹 내 총 행(*partition_by_clause*에 의해 그룹핑된 행 또는 전체 행)의 개수로 나눈 것이다. 즉, (어떤 행의 누적된 RANK)/(그룹 내 행의 개수)를 반환한다. 예를 들어, 전체 10개의 행 중에서 RANK가 1인 행의 개수가 2개이면 첫번째 행과 두번째 행의 **CUME_DIST** 값은 “2/10 = 0.2”가 된다.

다음은 이 함수의 예에서 사용될 스키마 및 데이터이다.

```
CREATE TABLE scores(id INT PRIMARY KEY AUTO_INCREMENT, math INT,
english INT, pe CHAR, grade INT);
```

```
INSERT INTO scores(math, english, pe, grade)
VALUES(60, 70, 'A', 1),
(60, 70, 'A', 1),
(60, 80, 'A', 1),
(60, 70, 'B', 1),
(70, 60, 'A', 1),
(70, 70, 'A', 1),
(80, 70, 'C', 1),
(70, 80, 'C', 1),
(85, 60, 'C', 1),
(75, 90, 'B', 1);
```

```
INSERT INTO scores(math, english, pe, grade)
VALUES(95, 90, 'A', 2),
(85, 95, 'B', 2),
(95, 90, 'A', 2),
(85, 95, 'B', 2),
(75, 80, 'D', 2),
(75, 85, 'D', 2),
(75, 70, 'C', 2),
(65, 95, 'A', 2),
(65, 95, 'A', 2),
(65, 95, 'A', 2);
```

다음은 집계 함수로 사용되는 예로, *math*, *english*, *pe* 3개의 칼럼에 대한 각각의 누적 분포 값을 더해 3으로 나눈 결과를 출력한다.

```
SELECT CUME_DIST(60, 70, 'D')
WITHIN GROUP(ORDER BY math, english, pe) AS cume
FROM scores;
```

```
1.904761904761905e-01
```

다음은 분석 함수로 사용되는 예로, *math*, *english*, *pe* 3개 칼럼을 기준으로 각 행의 누적 분포를 출력한다.

```
SELECT id, math, english, pe, grade, CUME_DIST() OVER(ORDER BY math,
english, pe) AS cume_dist
FROM scores
ORDER BY cume_dist;
```

grade	id	math	english	pe
		cume_dist		
	1	60	70	
'A'	2	60	70	1.0000000000000000e-01
'A'	4	60	70	1.0000000000000000e-01
'B'	3	60	80	1.5000000000000000e-01
'A'	18	65	95	2.0000000000000000e-01
'A'	19	65	95	3.5000000000000000e-01
'A'	20	65	95	3.5000000000000000e-01
'A'	5	70	60	4.0000000000000000e-01
'A'	6	70	70	4.5000000000000000e-01
'A'	8	70	80	5.0000000000000000e-01
'C'	17	75	70	5.5000000000000000e-01
'C'	15	75	80	6.0000000000000000e-01
'D'	16	75	85	

```

'D'          10          75          2          6.500000000000000e-01
          90
'B'          7          80          1          7.000000000000000e-01
          70
'C'          9          85          1          7.500000000000000e-01
          60
'C'          12         85          1          8.000000000000000e-01
          95
'B'          14         85          2          9.000000000000000e-01
          95
'B'          11         95          2          9.000000000000000e-01
          90
'A'          13         95          2          1.000000000000000e+00
          90
'A'          13         95          2          1.000000000000000e+00

```

다음은 분석 함수로 사용되는 예로, *math*, *english*, *pe* 3개 칼럼을 기준으로 *grade* 칼럼으로 그룹핑하여 각 행의 누적 분포를 출력한다.

```

SELECT id, math, english, pe, grade, CUME_DIST() OVER(PARTITION BY
grade ORDER BY math, english, pe) AS cume_dist
FROM scores
ORDER BY grade, cume_dist;

```

```

      id      math      english      pe
grade      cume_dist
=====
=====
      1          60          70      'A'
1      2.000000000000000e-01
      2          60          70      'A'
1      2.000000000000000e-01
      4          60          70      'B'
1      3.000000000000000e-01
      3          60          80      'A'
1      4.000000000000000e-01
      5          70          60      'A'
1      5.000000000000000e-01
      6          70          70      'A'
1      6.000000000000000e-01
      8          70          80      'C'
1      7.000000000000000e-01
     10          75          90      'B'
1      8.000000000000000e-01
      7          80          70      'C'
1      9.000000000000000e-01
      9          85          60      'C'
1      1.000000000000000e+00
     18          65          95      'A'

```

```

2      3.0000000000000000e-01
19      65      95      'A'
2      3.0000000000000000e-01
20      65      95      'A'
2      3.0000000000000000e-01
17      75      70      'C'
2      4.0000000000000000e-01
15      75      80      'D'
2      5.0000000000000000e-01
16      75      85      'D'
2      6.0000000000000000e-01
12      85      95      'B'
2      8.0000000000000000e-01
14      85      95      'B'
2      8.0000000000000000e-01
11      95      90      'A'
2      1.0000000000000000e+00
13      95      90      'A'
2      1.0000000000000000e+00

```

위의 결과에서 `id`가 1인 행은 `grade`가 1인 10개의 행 중에서 첫번째와 두번째에 위치하며, **CUME_DIST**의 값은 2/10, 즉 0.2가 된다.

`id`가 5인 행은 `grade`가 1인 10개의 행 중에서 다섯번째에 위치하며, **CUME_DIST**의 값은 5/10, 즉 0.5가 된다.

DENSE_RANK

DENSE_RANK() OVER ([<partition_by_clause>] <order_by_clause>)

DENSE_RANK 함수는 분석 함수로만 사용되며, **PARTITION BY** 절에 의한 칼럼 값의 그룹에서 값의 순위를 계산하여 **INTEGER** 로 출력한다. 공동 순위가 존재해도 그 다음 순위는 1을 더한다. 예를 들어, 13위에 해당하는 행이 3개여도 그 다음 행의 순위는 16위가 아니라 14위가 된다. 반면, **RANK()** 함수는 이와 달리 공동 순위의 개수만큼을 더해 다음 순위의 값을 계산한다.

반환 형식:

INT

다음은 역대 올림픽에서 연도별로 금메달을 많이 획득한 국가의 금메달 개수와 순위를 출력하는 예제이다. 공동 순위의 개수는 무시하고 다음 순위 값은 항상 1을 더한다.

```

SELECT host_year, nation_code, gold,
DENSE_RANK() OVER (PARTITION BY host_year ORDER BY gold DESC) AS
d_rank
FROM participant;

```

host_year	nation_code	gold	d_rank
1988	'URS '	55	1
1988	'GDR '	37	2
1988	'USA '	36	3
1988	'KOR '	12	4
1988	'HUN '	11	5
1988	'FRG '	11	5
1988	'BUL '	10	6
1988	'ROU '	7	7
1988	'ITA '	6	8
1988	'FRA '	6	8
1988	'KEN '	5	9
1988	'GBR '	5	9
1988	'CHN '	5	9
...			
1988	'CHI '	0	14
1988	'ARG '	0	14
1988	'JAM '	0	14
1988	'SUI '	0	14
1988	'SWE '	0	14
1992	'EUN '	45	1
1992	'USA '	37	2
1992	'GER '	33	3
...			
2000	'RSA '	0	15
2000	'NGR '	0	15
2000	'JAM '	0	15
2000	'BRA '	0	15
2004	'USA '	36	1
2004	'CHN '	32	2
2004	'RUS '	27	3
2004	'AUS '	17	4
2004	'JPN '	16	5
2004	'GER '	13	6
2004	'FRA '	11	7
2004	'ITA '	10	8
2004	'UKR '	9	9
2004	'CUB '	9	9
2004	'GBR '	9	9
2004	'KOR '	9	9
...			
2004	'EST '	0	17
2004	'SLO '	0	17
2004	'SCG '	0	17
2004	'FIN '	0	17
2004	'POR '	0	17
2004	'MEX '	0	17
2004	'LAT '	0	17
2004	'PRK '	0	17

FIRST_VALUE

FIRST_VALUE(*expression*) [{*RESPECT*|*IGNORE*} *NULLS*] *OVER* (<*analytic_clause*>)

FIRST_VALUE 함수는 분석 함수로만 사용되며, 정렬된 값 집합에서 첫번째 값을 반환한다. 집합 내의 첫번째 값이 null이면 함수는 **NULL**을 반환한다. 그러나, **IGNORE NULLS**를 명시하면 집합 내에서 null이 아닌 첫번째 값을 반환하거나, 모든 값이 null인 경우 **NULL**을 반환한다.

매개변수:

expression - 수치 또는 문자열을 반환하는 칼럼 또는 연산식. **FIRST_VALUE** 함수 또는 다른 분석 함수를 포함할 수 없다.

반환 형식:

*expression*의 타입

! 더 보기

`LAST_VALUE()`, `NTH_VALUE()`

다음은 예제 질의를 실행하기 위한 스키마와 데이터이다.

```
CREATE TABLE test_tbl(groupid int,itemno int);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,1);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,2);
INSERT INTO test_tbl VALUES(1,3);
INSERT INTO test_tbl VALUES(1,4);
INSERT INTO test_tbl VALUES(1,5);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,6);
INSERT INTO test_tbl VALUES(2,7);
```

다음은 **FIRST_VALUE** 함수를 수행하는 질의 및 결과이다.

```
SELECT groupid, itemno, FIRST_VALUE(itemno) OVER(PARTITION BY
groupid ORDER BY itemno) AS ret_val
FROM test_tbl;
```

groupid	itemno	ret_val
=====		
1	NULL	NULL
1	NULL	NULL
1	NULL	NULL
1	1	NULL
1	2	NULL
1	3	NULL
1	4	NULL
1	5	NULL
2	NULL	NULL
2	NULL	NULL
2	NULL	NULL
2	6	NULL
2	7	NULL

참고

CUBRID는 **NULL** 값을 모든 값보다 앞의 순서로 정렬한다. 즉, 아래의 SQL1은 **ORDER BY** 절에 **NULLS FIRST**가 포함된 SQL2로 해석된다.

```
SQL1: FIRST_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY
itemno) AS ret_val
SQL2: FIRST_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY
itemno NULLS FIRST) AS ret_val
```

다음은 **IGNORE NULLS**를 명시하는 예이다.

```
SELECT groupid, itemno, FIRST_VALUE(itemno) IGNORE NULLS
OVER(PARTITION BY groupid ORDER BY itemno) AS ret_val
FROM test_tbl;
```

groupid	itemno	ret_val
=====		
1	NULL	NULL
1	NULL	NULL
1	NULL	NULL
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1
2	NULL	NULL

2	NULL	NULL
2	NULL	NULL
2	6	6
2	7	6

GROUP_CONCAT

`GROUP_CONCAT([DISTINCT] expression [ORDER BY {column | unsigned_int} [ASC | DESC]] [SEPARATOR str_val])`

GROUP_CONCAT 함수는 집계 함수로만 사용되며, 그룹에서 **NULL** 이 아닌 값들을 연결하여 결과 문자열을 **VARCHAR** 타입으로 반환한다. 질의 결과 행이 없거나 **NULL** 값만 있으면 **NULL** 을 반환한다.

매개변수:

- **expression** - 수치 또는 문자열을 반환하는 칼럼 또는 연산식
- **str_val** - 구분자로 쓰일 문자열
- **DISTINCT** - 결과에서 중복되는 값을 제거한다.
- **ORDERBY** - 결과 값의 순서를 지정한다.
- **SEPARATOR** - 결과 값 사이에 구분할 구분자를 지정한다.
생략하면 기본값인 쉼표(,)를 구분자로 사용한다.

반환 형식:

STRING

리턴 값의 최대 크기는 시스템 파라미터 **group_concat_max_len** 의 설정을 따른다. 기본값은 **1024** 바이트이며, 최소값은 4바이트, 최대값은 INT_MAX(약2G)바이트이다.

이 함수는 **string_max_size_bytes** 파라미터의 영향을 받으며, **group_concat_max_len**의 값을 **string_max_size_bytes** 보다 크게 설정한 경우 **GROUP_CONCAT** 결과가 **string_max_size_bytes** 값을 초과하면 오류가 발생한다.

중복되는 값을 제거하려면 **DISTINCT** 절을 사용하면 된다. 그룹 결과의 값 사이에 사용되는 기본 구분자는 쉼표(,)이며, 구분자를 명시적으로 표현하려면 **SEPARATOR** 절과 그 뒤에 구분자로 사용할 문자열을 추가한다. 구분자를 제거하려면 **SEPARATOR** 절 뒤에 빈 문자열(empty string)을 입력한다.

결과 문자열에 문자형 데이터 타입이 아닌 다른 타입이 전달되면, 에러를 반환한다.

GROUP_CONCAT 함수를 사용하려면 다음의 조건을 만족해야 한다.

- 입력 인자로 하나의 표현식(또는 칼럼)만 허용한다.
- **ORDER BY** 를 이용한 정렬은 오직 인자로 사용되는 표현식(또는 칼럼)에 의해서만 가능하다.
- 구분자로 사용되는 문자열은 문자형 타입만 허용하며, 다른 타입은 허용하지 않는다.

```
SELECT GROUP_CONCAT(s_name) FROM code;
```

```
group_concat(s_name)
=====
'X,W,M,B,S,G'
```

```
SELECT GROUP_CONCAT(s_name ORDER BY s_name SEPARATOR ':') FROM code;
```

```
group_concat(s_name order by s_name separator ':')
=====
'B:G:M:S:W:X'
```

```
CREATE TABLE t(i int);
INSERT INTO t VALUES (4),(2),(3),(6),(1),(5);

SELECT GROUP_CONCAT(i*2+1 ORDER BY 1 SEPARATOR '') FROM t;
```

```
group_concat(i*2+1 order by 1 separator '')
=====
'35791113'
```

참고

CUBRID 11.3 부터 내부 문자열 함수의 최적화를 통해 많은 행을 처리하는 **GROUP_CONCAT**/의 성능이 개선되었다.

LAG

LAG(*expression*[, *offset*[, *default*]]) **OVER** ([<*partition_by_clause*>] <*order_by_clause*>)

LAG 함수는 분석 함수로만 사용되며 현재 행을 기준으로 *offset* 앞 행의 *expression* 값을 반환한다. 한 행에 자체 조인(self join) 없이 동시에 여러 개의 행에 접근하고 싶을 때 사용할 수 있다.

매개변수:

- **expression** – 숫자 또는 문자열을 반환하는 칼럼 또는 연산식
- **offset** – 오프셋 위치를 나타내는 정수. 생략 시 기본값 1
- **default** – 현재 위치에서 *offset* 앞에 위치한 *expression* 값이 NULL인 경우 출력하는 값. 기본값 NULL

반환 형식:

NUMBER or STRING

다음은 사번 순으로 정렬하여 같은 행에 앞의 사번을 같이 출력하는 예이다.

```
CREATE TABLE t_emp (name VARCHAR(10), empno INT);
INSERT INTO t_emp VALUES
  ('Amie', 11011),
  ('Jane', 13077),
  ('Lora', 12045),
  ('James', 12006),
  ('Peter', 14006),
  ('Tom', 12786),
  ('Ralph', 23518),
  ('David', 55);

SELECT name, empno, LAG (empno, 1) OVER (ORDER BY empno) prev_empno
FROM t_emp;
```

name	empno	prev_empno
'David'	55	NULL
'Amie'	11011	55
'James'	12006	11011
'Lora'	12045	12006
'Tom'	12786	12045
'Jane'	13077	12786
'Peter'	14006	13077
'Ralph'	23518	14006

이와는 반대로, 현재 행을 기준으로 *offset* 이후 행의 *expression* 값을 반환하는 **LEAD()** 함수를 참고한다.

LAST_VALUE

LAST_VALUE(*expression*) [{RESPECT|IGNORE} NULLS] OVER (<*analytic_clause*>)

LAST_VALUE 함수는 분석 함수로만 사용되며, 정렬된 값 집합에서 마지막 값을 반환한다. 집합 내의 마지막 값이 null이면 함수는 NULL을 반환한다. 그러나, IGNORE NULLS를 명시하면 집합 내에서 null이 아닌 마지막 값을 반환하거나, 모든 값이 null인 경우 NULL을 반환한다.

매개변수:

expression - 수치 또는 문자열을 반환하는 칼럼 또는 연산식. LAST_VALUE 함수 또는 다른 분석 함수를 포함할 수 없다.

반환 형식:

*expression*의 타입

! 더 보기

FIRST_VALUE(), NTH_VALUE()

다음은 예제 질의를 실행하기 위한 스키마와 데이터이다.

```
CREATE TABLE test_tbl(groupid int,itemno int);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,1);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,2);
INSERT INTO test_tbl VALUES(1,3);
INSERT INTO test_tbl VALUES(1,4);
INSERT INTO test_tbl VALUES(1,5);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,6);
INSERT INTO test_tbl VALUES(2,7);
```

다음은 LAST_VALUE 함수를 수행하는 질의 및 결과이다.

```
SELECT groupid, itemno, LAST_VALUE(itemno) OVER(PARTITION BY groupid
ORDER BY itemno) AS ret_val
FROM test_tbl;
```

groupid	itemno	ret_val
1	NULL	NULL
1	NULL	NULL
1	NULL	NULL
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
2	NULL	NULL
2	NULL	NULL
2	NULL	NULL
2	6	6
2	7	7

LAST_VALUE 함수는 현재 행을 기준으로 계산된다. 즉, 아직 바인딩되지 않은 값은 계산에 포함되지 않는다. 예를 들어, 위의 결과에서 (groupid, itemno) = (1, 1)인 LAST_VALUE 함수의 값은 1이고, (groupid, itemno) = (1, 2)인 LAST_VALUE 함수의 값은 2이다.

참고

CUBRID는 NULL 값을 모든 값보다 앞의 순서로 정렬한다. 즉, 아래의 SQL1은 ORDER BY 절에 NULLS FIRST가 포함된 SQL2로 해석된다.

```
SQL1: LAST_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY
itemno) AS ret_val
SQL2: LAST_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY itemno
NULLS FIRST) AS ret_val
```

LEAD

LEAD(*expression*, *offset*[, *default*]) OVER ([<partition_by_clause>] <order_by_clause>)

LEAD 함수는 분석 함수로만 사용되며, 현재 행을 기준으로 *offset* 이후 행의 *expression* 값을 반환한다. 한 행에 자체 조인(self join) 없이 동시에 여러 개의 행에 접근하고 싶을 때 사용할 수 있다.

매개변수:

- **expression** – 숫자 또는 문자열을 반환하는 칼럼 또는 연산식
- **offset** – 오프셋 위치를 나타내는 정수. 생략 시 기본값 1
- **default** – 현재 위치에서 *offset* 앞에 위치한 *expression* 값이 NULL인 경우 출력하는 값. 기본값 NULL

반환 형식:

NUMBER or STRING

다음은 사번 순으로 정렬하여 같은 행에 다음 사번을 같이 출력하는 예이다.

```
CREATE TABLE t_emp (name VARCHAR(10), empno INT);
INSERT INTO t_emp VALUES
('Amie', 11011), ('Jane', 13077), ('Lora', 12045), ('James', 12006),
('Peter', 14006), ('Tom', 12786), ('Ralph', 23518), ('David', 55);

SELECT name, empno, LEAD (empno, 1) OVER (ORDER BY empno) next_empno
FROM t_emp;
```

name	empno	next_empno
'David'	55	11011
'Amie'	11011	12006
'James'	12006	12045
'Lora'	12045	12786
'Tom'	12786	13077
'Jane'	13077	14006
'Peter'	14006	23518
'Ralph'	23518	NULL

다음은 tbl_board 테이블에서 현재 행을 기준으로 앞의 행과 이후 행의 title을 같이 출력하는 예이다.

```
CREATE TABLE tbl_board (num INT, title VARCHAR(50));
INSERT INTO tbl_board VALUES
(1, 'title 1'), (2, 'title 2'), (3, 'title 3'), (4, 'title 4'), (5, 'title 5'),
(6, 'title 6'), (7, 'title 7');

SELECT num, title,
       LEAD (title,1,'no next page') OVER (ORDER BY num) next_title,
```

```
LAG (title,1,'no previous page') OVER (ORDER BY num) prev_title
FROM tbl_board;
```

num	title	next_title	prev_title
1	'title 1'	'title 2'	NULL
2	'title 2'	'title 3'	'title 1'
3	'title 3'	'title 4'	'title 2'
4	'title 4'	'title 5'	'title 3'
5	'title 5'	'title 6'	'title 4'
6	'title 6'	'title 7'	'title 5'
7	'title 7'	NULL	'title 6'

다음은 tbl_board 테이블에서 특정 행을 기준으로 앞의 행과 이후 행의 타이틀을 같이 출력하는 예이다. WHERE 조건이 괄호 안에 있으면 하나의 행만 선택되고, 앞의 행과 이후 행이 존재하지 않게 되어 next_title과 prev_title의 값이 NULL이 됨에 유의한다.

```
SELECT * FROM
(
  SELECT num, title,
    LEAD(title,1,'no next page') OVER (ORDER BY num) next_title,
    LAG(title,1,'no previous page') OVER (ORDER BY num)
prev_title
  FROM tbl_board
)
WHERE num=5;
```

num	title	next_title	prev_title
5	'title 5'	'title 6'	'title 4'

MAX

MAX([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression)

MAX([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression) OVER (<analytic_clause>)

MAX 함수는 집계 함수 또는 분석 함수로 사용되며, 모든 행에 대하여 연산식 값 중 최대 값을 구한다. 하나의 연산식 *expression* 만 인자로 지정된다. 문자열을 반환하는 연산식에 대해서는 사전 순서를 기준으로 뒤에 나오는 문자열이 최대 값이 되고, 수치를 반환하는 연산식에 대해서는 크기가 가장 큰 값이 최대 값이다.

param expression:

수치 또는 문자열을 반환하는 하나의 연산식을 지정한다. 컬렉션 타입의 데이터를 반환하는 연산식은 지정할 수 없다.

param ALL:

모든 값에 대해 최대 값을 구하기 위해 사용되며, 기본 값이다.

param

DISTINCT,DISTINCTROW,UNIQUE:

중복이 제거된 유일한 값에 대해서 최대 값을 구하기 위해 사용된다.

rtype:

expression의 타입

다음은 올림픽 대회 중 한국이 획득한 최대 금메달의 수를 반환하는 예제이다.

```
SELECT MAX(gold) FROM participant WHERE nation_code = 'KOR';
```

```
max(gold)
=====
12
```

다음은 역대 올림픽 대회 중 국가 코드와 연도 순대로 nation_code가 'AU'로 시작하는 국가가 획득한 금메달 수와 해당 국가의 역대 최대 금메달의 수를 같이 출력하는 예제이다.

```
SELECT host_year, nation_code, gold,
       MAX(gold) OVER (PARTITION BY nation_code) mx_gold
FROM participant
WHERE nation_code LIKE 'AU%'
ORDER BY nation_code, host_year;
```

host_year	nation_code	gold	mx_gold
1988	'AUS'	3	17
1992	'AUS'	7	17

1996	'AUS'	9	17
2000	'AUS'	16	17
2004	'AUS'	17	17
1988	'AUT'	1	2
1992	'AUT'	0	2
1996	'AUT'	0	2
2000	'AUT'	2	2
2004	'AUT'	2	2

MEDIAN

MEDIAN(*expression*)

MEDIAN(*expression*) OVER ([<partition_by_clause>])

MEDIAN 함수는 집계 함수 또는 분석 함수로 사용되며, 중앙값(median value)을 반환한다. 중앙값은 데이터의 최소값과 최대값의 중앙에 위치하게 되는 값을 말한다.

param expression:

숫자 또는 날짜로 변환될 수 있는 값을 가진 컬럼 또는 연산식

rtype:

DOUBLE 또는 **DATETIME**

다음은 예제 질의를 실행하기 위한 테이블 스키마 및 데이터이다.

```
CREATE TABLE tbl (col1 int, col2 double);
INSERT INTO tbl VALUES(1,2), (1,1.5), (1,1.7), (1,1.8), (2,3), (2,4), (3,5);
```

다음은 집계 함수로 사용되는 예로서, col1을 기준으로 각 그룹별로 집계한 col2의 중앙값을 반환한다.

```
SELECT col1, MEDIAN(col2)
FROM tbl GROUP BY col1;
```

```
col1  median(col2)
=====
1     1.7500000000000000e+00
```

```

2  3.5000000000000000e+00
3  5.0000000000000000e+00

```

다음은 분석 함수로 사용되는 예로서, col1을 기준으로 각 그룹별 col2의 중앙값을 반환한다.

```

SELECT col1, MEDIAN(col2) OVER (PARTITION BY col1)
FROM tbl;

```

```

col1 median(col2) over (partition by col1)
=====
1  1.7500000000000000e+00
1  1.7500000000000000e+00
1  1.7500000000000000e+00
1  1.7500000000000000e+00
2  3.5000000000000000e+00
2  3.5000000000000000e+00
3  5.0000000000000000e+00

```

MIN

MIN([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression)

MIN([DISTINCT | DISTINCTROW | UNIQUE | ALL] expression) OVER (<analytic_clause>)

MIN 함수는 집계 함수 또는 분석 함수로 사용되며, 모든 행에 대하여 연산식 값 중 최소 값을 구한다. 하나의 연산식 *expression* 만 인자로 지정된다. 문자열을 반환하는 연산식에 대해서는 사전 순서를 기준으로 앞에 나오는 문자열이 최소 값이 되고, 수치를 반환하는 연산식에 대해서는 크기가 가장 작은 값이 최소 값이다.

param expression:

수치 또는 문자열을 반환하는 하나의 연산식을 지정한다. 컬렉션 타입의 데이터를 반환하는 연산식은 지정할 수 없다.

param ALL:

모든 값에 대해 최소 값을 구하기 위해 사용되며, 기본 값이다.

param

DISTINCT,DISTINCTROW,UNIQUE:

중복이 제거된 유일한 값에 대해서 최소 값을 구하기 위해 사용된다.

rtype:

expression의 타입

다음은 *demodb* 에서 올림픽 대회 중 한국이 획득한 최소 금메달의 수를 반환하는 예제이다.

```
SELECT MIN(gold) FROM participant WHERE nation_code = 'KOR';
```

```
min(gold)
=====
7
```

다음은 역대 올림픽 대회 중 국가 코드와 연도 순대로 nation_code가 'AU'로 시작하는 국가가 획득한 금메달 수와 해당 국가의 역대 최소 금메달의 수를 같이 출력하는 예제이다.

```
SELECT host_year, nation_code, gold,
       MIN(gold) OVER (PARTITION BY nation_code) mn_gold
FROM participant WHERE nation_code like 'AU%' ORDER BY nation_code,
host_year;
```

host_year	nation_code	gold	mn_gold
1988	'AUS'	3	3
1992	'AUS'	7	3
1996	'AUS'	9	3
2000	'AUS'	16	3
2004	'AUS'	17	3
1988	'AUT'	1	0
1992	'AUT'	0	0
1996	'AUT'	0	0
2000	'AUT'	2	0
2004	'AUT'	2	0

NTH_VALUE

NTH_VALUE(expression, N) [{RESPECT|IGNORE} NULLS] OVER (<analytic_clause>)

NTH_VALUE 함수는 분석 함수로만 사용되며, 정렬된 값 집합에서 N번째 행의 expression 값을 반환한다.

매개변수:

- **expression** – 수치 또는 문자열을 반환하는 칼럼 또는 연산식
- **N** – 양의 정수로 해석될 수 있는 상수, 바인드 변수, 칼럼 또는 표현식

반환 형식:

*expression*의 타입

! 더 보기

`FIRST_VALUE()`, `LAST_VALUE()`

{RESPECT|IGNORE} NULLS 구문은 *expression*의 null 값을 계산에 포함시킬지 여부를 결정한다. 기본 값은 **RESPECT NULLS**이다.

다음은 예제 질의를 실행하기 위한 스키마와 데이터이다.

```
CREATE TABLE test_tbl(groupid int,itemno int);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,1);
INSERT INTO test_tbl VALUES(1,null);
INSERT INTO test_tbl VALUES(1,2);
INSERT INTO test_tbl VALUES(1,3);
INSERT INTO test_tbl VALUES(1,4);
INSERT INTO test_tbl VALUES(1,5);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,null);
INSERT INTO test_tbl VALUES(2,6);
INSERT INTO test_tbl VALUES(2,7);
```

다음은 *N*의 값을 2로 하여 **NTH_VALUE** 함수를 수행하는 질의 및 결과이다.

```
SELECT groupid, itemno, NTH_VALUE(itemno, 2) IGNORE NULLS
OVER(PARTITION BY groupid ORDER BY itemno NULLS FIRST) AS ret_val
FROM test_tbl;
```

groupid	itemno	ret_val
=====		
1	NULL	NULL
1	NULL	NULL
1	NULL	NULL
1	1	NULL
1	2	2
1	3	2
1	4	2
1	5	2
2	NULL	NULL
2	NULL	NULL
2	NULL	NULL
2	6	NULL
2	7	7

참고

CUBRID는 NULL을 모든 값보다 앞의 순서로 정렬한다. 즉, 아래의 SQL1은 ORDER BY 절에 NULLS FIRST가 포함된 SQL2로 해석된다.

```
SQL1: NTH_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY itemno)
      AS ret_val
SQL2: NTH_VALUE(itemno) OVER(PARTITION BY groupid ORDER BY itemno
      NULLS FIRST) AS ret_val
```

NTILE

NTILE(expression) OVER ([<partition_by_clause>] <order_by_clause>)

NTILE 함수는 분석 함수로만 사용되며, 순차적인 데이터 집합을 입력 인자 값에 의해 일련의 버킷으로 나누며, 각 행에 적당한 버킷 번호를 1부터 할당한다.

매개변수:

expression – 버킷의 개수. 숫자 값을 반환하는 임의의 연산식을 지정한다.

반환 형식:

INT

NTILE 함수는 주어진 버킷 개수로 행의 개수를 균등하게 나누어 버킷 번호를 부여한다. 즉, NTILE 함수는 equi-height histogram을 생성해준다. 각 버킷에 있는 행의 개수는 최대 1개까지 차이가 생길 수 있다. 나머지 값(행의 개수를 버킷 개수로 나눈 나머지)이 각 버킷에 대해 1번 버킷부터 하나씩 배포된다.

반면에 **WIDTH_BUCKET()** 함수는 주어진 버킷 개수로 주어진 범위를 균등하게 나누어 버킷 번호를 부여한다. 즉, 버킷마다 각 범위의 넓이는 균등하다.

다음은 8명의 고객을 생년월일을 기준으로 5개의 버킷으로 나누되, 각 버킷의 수가 균등하도록 나누는 예이다. 1, 2, 3번 버킷에는 2개의 행이, 4, 5번 버킷에는 2개의 행이 존재한다.

```
CREATE TABLE t_customer(name VARCHAR(10), birthdate DATE);
INSERT INTO t_customer VALUES
  ('Amie', date'1978-03-18'),
  ('Jane', date'1983-05-12'),
  ('Lora', date'1987-03-26'),
  ('James', date'1948-12-28'),
  ('Peter', date'1988-10-25'),
  ('Tom', date'1980-07-28'),
  ('Ralph', date'1995-03-17'),
  ('David', date'1986-07-28');

SELECT name, birthdate, NTILE(5) OVER (ORDER BY birthdate) age_group
FROM t_customer;
```

name	birthdate	age_group
'James'	12/28/1948	1
'Amie'	03/18/1978	1
'Tom'	07/28/1980	2
'Jane'	05/12/1983	2
'David'	07/28/1986	3
'Lora'	03/26/1987	3
'Peter'	10/25/1988	4
'Ralph'	03/17/1995	5

다음은 8명의 학생을 점수가 높은 순으로 5개의 버킷으로 나눈 후, 이름 순으로 출력하되, 각 버킷의 행의 개수는 균등하게 나누는 예이다. t_score 테이블의 score 칼럼에는 8개의 행이 존재하므로, 8을 5로 나눈 나머지 3개 행이 1번 버킷부터 각각 할당되어 1,2,3번 버킷은 4,5번 버킷에 비해 1개의 행이 더 존재한다. NTILE 함수는 점수의 범위와는 무관하게 행의 개수를 기준으로 균등하게 grade를 나눈다.

```
CREATE TABLE t_score(name VARCHAR(10), score INT);
INSERT INTO t_score VALUES
  ('Amie', 60),
  ('Jane', 80),
```

```
( 'Lora', 60),
( 'James', 75),
( 'Peter', 70),
( 'Tom', 30),
( 'Ralph', 99),
( 'David', 55);
```

```
SELECT name, score, NTILE(5) OVER (ORDER BY score DESC) grade
FROM t_score
ORDER BY name;
```

name	score	grade
'Ralph'	99	1
'Jane'	80	1
'James'	75	2
'Peter'	70	2
'Amie'	60	3
'Lora'	60	3
'David'	55	4
'Tom'	30	5

PERCENT_RANK

PERCENT_RANK(*expression* [, *expression*] ...) WITHIN GROUP (<order_by_clause>)

PERCENT_RANK() OVER ([<partition_by_clause>] <order_by_clause>)

PERCENT_RANK 함수는 집계 함수 또는 분석 함수로 사용되며, 그룹에서 행의 상대적인 위치를 순위 퍼센트로 반환한다. CUME_DIST 함수(누적 분포 값을 반환)와 유사하다. PERCENT_RANK가 반환하는 값의 범위는 0부터 1까지이다. PERCENT_RANK의 첫번째 값은 항상 0이다.

param expression:

수치 또는 문자열을 반환하는 연산식. 칼럼이 올 수 없다.

rtype:

DOUBLE

[🔍 더 보기](#)

CUME_DIST() , RANK()

집계 함수인 경우, 집계 그룹 전체 행에서 선택된 가상(hypothetical) 행의 RANK에서 1을 뺀 값에 대해 집계 그룹 내의 행의 개수로 나눈 값을 반환한다. 즉, (가상 행의 RANK - 1)/(집계 그룹 행의 개수)를 반환한다.

분석 함수인 경우, PARTITION BY에 의해 나누어진 그룹별로 각 행을 ORDER BY 절에 명시된 순서로 정렬했을 때 (그룹별 RANK - 1)/(그룹 행의 개수 - 1)을 반환한다. 예를 들어, 전체 10개의 행 중에서 첫번째 순서(RANK=1)로 등장한 행의 개수가 2개이면 첫번째 행과 두번째 행의 PERCENT_RANK 값은 $(1-1)/(10-1)=0$ 이 된다.

다음은 입력 값 VAL이 존재할 때 집계 함수로 사용되는 CUME_DIST와 PERCENT_RANK의 반환 값을 비교한 표이다.

VAL	RANK()	DENSE_RANK()	CUME_DIST(VAL)	PERCENT_RANK(VAL)
100	1	1	0.33 => $(1+1)/(5+1)$	0 => $(1-1)/5$
200	2	2	0.67 => $(2+1)/(5+1)$	0.2 => $(2-1)/5$
200	2	2	0.67 => $(2+1)/(5+1)$	0.2 => $(2-1)/5$
300	4	3	0.83 => $(4+1)/(5+1)$	0.6 => $(4-1)/5$
400	5	4	1 => $(5+1)/(5+1)$	0.8 => $(5-1)/5$

다음은 입력 값 VAL이 존재할 때 분석 함수로 사용되는 CUME_DIST와 PERCENT_RANK의 반환 값을 비교한 표이다.

VAL	RANK()	DENSE_RANK()	CUME_DIST()	PERCENT_RANK()
100	1	1	0.2 => 1/5	0 => (1-1)/(5-1)
200	2	2	0.6 => 3/5	0.25 => (2-1)/(5-1)
200	2	2	0.6 => 3/5	0.25 => (2-1)/(5-1)
300	4	3	0.8 => 4/5	0.75 => (4-1)/(5-1)
400	5	4	1 => 5/5	1 => (5-1)/(5-1)

위의 표와 관련된 스키마 및 질의의 예는 다음과 같다.

```
CREATE TABLE test_tbl(VAL INT);
INSERT INTO test_tbl VALUES (100), (200), (200), (300), (400);

SELECT CUME_DIST(100) WITHIN GROUP (ORDER BY val) AS cume FROM
test_tbl;
SELECT PERCENT_RANK(100) WITHIN GROUP (ORDER BY val) AS pct_rnk FROM
test_tbl;

SELECT CUME_DIST() OVER (ORDER BY val) AS cume FROM test_tbl;
SELECT PERCENT_RANK() OVER (ORDER BY val) AS pct_rnk FROM test_tbl;
```

다음은 아래에서 보여줄 질의에서 사용된 스키마 및 데이터이다.

```
CREATE TABLE scores(id INT PRIMARY KEY AUTO_INCREMENT, math INT,
english INT, pe CHAR, grade INT);

INSERT INTO scores(math, english, pe, grade)
VALUES(60, 70, 'A', 1),
(60, 70, 'A', 1),
(60, 80, 'A', 1),
(60, 70, 'B', 1),
(70, 60, 'A', 1),
```

```

(70, 70, 'A', 1) ,
(80, 70, 'C', 1) ,
(70, 80, 'C', 1),
(85, 60, 'C', 1),
(75, 90, 'B', 1);
INSERT INTO scores(math, english, pe, grade)
VALUES(95, 90, 'A', 2),
(85, 95, 'B', 2),
(95, 90, 'A', 2),
(85, 95, 'B', 2),
(75, 80, 'D', 2),
(75, 85, 'D', 2),
(75, 70, 'C', 2),
(65, 95, 'A', 2),
(65, 95, 'A', 2),
(65, 95, 'A', 2);

```

다음은 집계 함수로 사용되는 예로, *math*, *english*, *pe* 3개의 칼럼에 대한 **PERCENT_RANK** 값을 더한 후 3으로 나눈 결과를 출력한다.

```

SELECT PERCENT_RANK(60, 70, 'D')
WITHIN GROUP(ORDER BY math, english, pe) AS percent_rank
FROM scores;

```

```
1.5000000000000000e-01
```

다음은 분석 함수로 사용되는 예로, *math*, *english*, *pe* 3개 칼럼을 기준으로 행 전체의 **PERCENT_RANK** 값을 출력한다.

```

SELECT id, math, english, pe, grade, PERCENT_RANK() OVER(ORDER BY
math, english, pe) AS percent_rank
FROM scores
ORDER BY percent_rank;

```

grade	id	math	english	pe	percent_rank
	1	60	70		
'A'	2	60	70	1	0.0000000000000000e+00
'A'	4	60	70	1	0.0000000000000000e+00
'B'	3	60	80	1	1.052631578947368e-01
'A'				1	1.578947368421053e-01

```

18          65          95
'A'          2      2.105263157894737e-01
19          65          95
'A'          2      2.105263157894737e-01
20          65          95
'A'          2      2.105263157894737e-01
5           70          60
'A'          1      3.684210526315789e-01
6           70          70
'A'          1      4.210526315789473e-01
8           70          80
'C'          1      4.736842105263158e-01
17          75          70
'C'          2      5.263157894736842e-01
15          75          80
'D'          2      5.789473684210527e-01
16          75          85
'D'          2      6.315789473684210e-01
10          75          90
'B'          1      6.842105263157895e-01
7           80          70
'C'          1      7.368421052631579e-01
9           85          60
'C'          1      7.894736842105263e-01
12          85          95
'B'          2      8.421052631578947e-01
14          85          95
'B'          2      8.421052631578947e-01
11          95          90
'A'          2      9.473684210526315e-01
13          95          90
'A'          2      9.473684210526315e-01

```

다음은 분석 함수로 사용되는 예로, *math*, *english*, *pe* 3개 칼럼을 기준으로 *grade* 칼럼으로 그룹핑하여 **PERCENT_RANK** 값을 출력한다.

```

SELECT id, math, english, pe, grade, RANK(), PERCENT_RANK()
OVER(PARTITION BY grade ORDER BY math, english, pe) AS percent_rank
FROM scores
ORDER BY grade, percent_rank;

```

```

      id      math  english  pe
grade  percent_rank
=====
=====
      1         60        70
'A'    1      0.000000000000000e+00
      2         60        70
'A'    1      0.000000000000000e+00

```

	4	60	70	
'B'			1	2.222222222222222e-01
	3	60	80	
'A'			1	3.333333333333333e-01
	5	70	60	
'A'			1	4.444444444444444e-01
	6	70	70	
'A'			1	5.555555555555556e-01
	8	70	80	
'C'			1	6.666666666666666e-01
	10	75	90	
'B'			1	7.777777777777778e-01
	7	80	70	
'C'			1	8.888888888888888e-01
	9	85	60	
'C'			1	1.000000000000000e+00
	18	65	95	
'A'			2	0.000000000000000e+00
	19	65	95	
'A'			2	0.000000000000000e+00
	20	65	95	
'A'			2	0.000000000000000e+00
	17	75	70	
'C'			2	3.333333333333333e-01
	15	75	80	
'D'			2	4.444444444444444e-01
	16	75	85	
'D'			2	5.555555555555556e-01
	12	85	95	
'B'			2	6.666666666666666e-01
	14	85	95	
'B'			2	6.666666666666666e-01
	11	95	90	
'A'			2	8.888888888888888e-01
	13	95	90	
'A'			2	8.888888888888888e-01

위의 결과에서 *id*가 1인 행은 *grade*가 1인 10개의 행 중에서 첫번째와 두번째에 위치하며, **PERCENT_RANK**의 값은 $(1-1)/(10-1)=0$ 이 된다. *id*가 5인 행은 *grade*가 1인 10개의 행 중에서 다섯번째에 위치하며, **PERCENT_RANK**의 값은 $(5-1)/(10-1)=0.44$ 가 된다.

PERCENTILE_CONT

PERCENTILE_CONT(expression1) WITHIN GROUP (ORDER BY expression2 [ASC | DESC]) [OVER (<partition_by_clause>)]

PERCENTILE_CONT 함수는 집계 함수 또는 분석 함수로 사용되며, 연속 분포(continuous distribution) 모델을 가정한 역 분포 함수이다. 백분위 값을 입력 받아 정렬된 값들 중 백분위에 해당하는 보간 값(interpolated value)을 반환한다. 계산 시 NULL 값은 무시된다.

이 함수는 입력 인자로 숫자형 타입 또는 숫자로 변환될 수 있는 문자열이 사용되며, 반환하는 값의 타입은 DOUBLE이다.

매개변수:

- **expression1** – 백분위 값. 0과 1사이의 숫자여야 한다.
- **expression2** – ORDER BY 절에 뒤따르는 칼럼 이름. 칼럼 개수는 *expression1*의 칼럼 개수와 동일해야 한다.

반환 형식:

DOUBLE

더 보기

PERCENTILE_DISC와 PERCENTILE_CONT 의 차이

집계 함수인 경우, **PERCENTILE_DISC** 함수는 **ORDER BY** 절에 명시된 순서로 결과 값을 정렬한 후, 집계 그룹에 있는 행에서 백분위에 해당하는 보간 값을 반환한다.

분석 함수인 경우, **PARTITION BY**에 의해 나누어진 그룹별로 각 행을 **ORDER BY** 절에 명시된 순서로 정렬한 후, 그룹 내의 행에서 백분위에 해당하는 보간 값을 반환한다.

참고

PERCENTILE_CONT와 PERCENTILE_DISC 의 차이

PERCENTILE_CONT와 PERCENTILE_DISC는 다른 결과를 반환할 수 있다.

PERCENTILE_CONT는 연속적인 보간을 수행한 이후 계산된 결과를 반환한다.

PERCENTILE_DISC는 집계된 값의 집합으로부터 값을 반환한다.

아래 예에서 백분위 값이 0.5이면 PERCENTILE_CONT 함수는 짝수 원소를 가진 그룹에 대해 두 개의 중간값의 평균을 반환하는 반면, PERCENTILE_DISC 함수는 두 개의 중간 값 중 첫번째 값을 반환한다. 홀수 개수의 원소를 가진 집계 그룹에 대해서는, 두 함수 모두 중간 원소의 값을 반환한다.

실제로 MEDIAN 함수는 기본 백분위수 값(0.5)이 포함된 PERCENTILE_CONT의 특수한 경우이다. 자세한 내용은 [MEDIAN\(\)](#) 을 참고한다.

다음은 이 함수의 예에서 사용될 스키마 및 데이터이다.

```
CREATE TABLE scores([id] INT PRIMARY KEY AUTO_INCREMENT, [math] INT,
english INT, [class] CHAR);
```

```
INSERT INTO scores VALUES
(1, 30, 70, 'A'),
(2, 40, 70, 'A'),
(3, 60, 80, 'A'),
(4, 70, 70, 'A'),
(5, 72, 60, 'A'),
(6, 77, 70, 'A'),
(7, 80, 70, 'C'),
(8, 70, 80, 'C'),
(9, 85, 60, 'C'),
(10, 78, 90, 'B'),
(11, 95, 90, 'D'),
(12, 85, 95, 'B'),
(13, 95, 90, 'B'),
(14, 85, 95, 'B'),
(15, 75, 80, 'D'),
(16, 75, 85, 'D'),
(17, 75, 70, 'C'),
(18, 65, 95, 'C'),
(19, 65, 95, 'D'),
(20, 65, 95, 'D');
```

다음은 집계 함수로 사용되는 예로, *math* 칼럼에 대한 중앙값을 출력한다.

```
SELECT PERCENTILE_CONT(0.5)
WITHIN GROUP(ORDER BY math) AS pcont
FROM scores;
```

```
pcont
=====
7.500000000000000e+01
```

다음은 분석 함수로 사용되는 예로, *class* 칼럼의 값이 같은 것끼리 그룹핑한 집합 내에서 *math* 칼럼에 대한 중앙값(median)을 출력한다.

```
SELECT math, [class], PERCENTILE_CONT(0.5)
WITHIN GROUP(ORDER BY math)
OVER (PARTITION BY [class]) AS pcont
FROM scores;
```

math	class	pcont
30	'A'	6.500000000000000e+01
40	'A'	6.500000000000000e+01
60	'A'	6.500000000000000e+01
70	'A'	6.500000000000000e+01
72	'A'	6.500000000000000e+01
77	'A'	6.500000000000000e+01
78	'B'	8.500000000000000e+01
85	'B'	8.500000000000000e+01
85	'B'	8.500000000000000e+01
95	'B'	8.500000000000000e+01
65	'C'	7.500000000000000e+01
70	'C'	7.500000000000000e+01
75	'C'	7.500000000000000e+01
80	'C'	7.500000000000000e+01
85	'C'	7.500000000000000e+01
65	'D'	7.500000000000000e+01
65	'D'	7.500000000000000e+01
75	'D'	7.500000000000000e+01
75	'D'	7.500000000000000e+01
95	'D'	7.500000000000000e+01

class 'A'에서 math의 값은 총 6개인데, PERCENTILE_CONT는 이산 값으로부터 연속된 값이 존재함을 가정하므로, 중앙값은 3번째 값 60과 4번째 값 70의 평균인 65가 된다.

PERCENTILE_CONT는 연속된 값을 가정하므로 연속된 값의 표현이 가능한 DOUBLE 타입으로 변환된 값을 출력한다.

PERCENTILE_DISC

PERCENTILE_DISC(expression1) WITHIN GROUP (ORDER BY expression2 [ASC | DESC]) [OVER (<partition_by_clause>)]

PERCENTILE_DISC 함수는 집계 함수 또는 분석 함수로 사용되며, 이산 분포(discrete distribution) 모델을 가정한 역 분포 함수이다. 백분위 값을 입력 받아 정렬된 값들 중 백분위에 해당하는 이산 값(discrete value)을 반환한다. 계산 시 NULL 값은 무시된다.

이 함수는 입력 인자로 숫자형 타입 또는 숫자로 변환될 수 있는 문자열이 사용되며, 반환 타입은 입력 값의 타입과 동일하다.

매개변수:

- **expression1** – 백분위 값. 0과 1사이의 숫자여야 한다.
- **expression2** – ORDER BY 절에 뒤따르는 칼럼 이름. 칼럼 개수는 *expression1*의 칼럼 개수와 동일해야 한다.

반환 형식:

expression2 의 타입과 동일.

! 더 보기

PERCENTILE_DISC 와 PERCENTILE_CONT 의 차이

집계 함수의 경우, 이것은 **ORDER BY** 절에 기술된 순서로 결과를 정렬한다; 그리고 집계 그룹에 있는 행에서 백분위에 위치한 값을 반환한다.

분석 함수인 경우, **PARTITION BY**에 의해 나누어진 그룹별로 각 행을 **ORDER BY** 절에 명시된 순서로 정렬한 후 그룹 내의 행에서 백분위에 위치한 값을 반환한다.

이 함수의 예에서 사용된 스키마와 데이터는 `PERCENTILE_CONT()` 에서 사용된 것과 동일하다.

다음은 집계 함수로 사용되는 예로, *math* 칼럼에 대한 중앙값(median)을 출력한다.

```
SELECT PERCENTILE_DISC(0.5)
WITHIN GROUP(ORDER BY math) AS pdisc
FROM scores;
```

```
pdisc
=====
75
```

다음은 분석 함수로 사용되는 예로, *class* 칼럼의 값이 같은 것끼리 그룹핑한 집합 내에서 *math* 칼럼에 대한 중앙값(median)을 출력한다.

```
SELECT math, [class], PERCENTILE_DISC(0.5)
WITHIN GROUP(ORDER BY math)
OVER (PARTITION BY [class]) AS pdisc
FROM scores;
```


math	class	pdisc
30	'A'	60
40	'A'	60
60	'A'	60
70	'A'	60
72	'A'	60
77	'A'	60
78	'B'	85
85	'B'	85
85	'B'	85
95	'B'	85
65	'C'	75
70	'C'	75
75	'C'	75
80	'C'	75
85	'C'	75
65	'D'	75
65	'D'	75
75	'D'	75
75	'D'	75
95	'D'	75

class 'A'에서 math의 값은 총 6개인데, PERCENTILE_DISC는 중간값이 두 개일 때 앞의 값을 출력하므로, 중앙값은 3번째 값 60과 4번째 값 70 중 앞의 것인 60이 된다.

RANK

RANK() OVER ([<partition_by_clause>] <order_by_clause>)

RANK 함수는 분석 함수로만 사용되며, **PARTITION BY** 절에 의한 칼럼 값의 그룹에서 값의 순위를 계산하여 **INTEGER** 로 출력한다. 공동 순위가 존재하면 그 다음 순위는 공동 순위의 개수를 더한 숫자이다. 예를 들어, 13위에 해당하는 행이 3개이면 그 다음 행의 순위는 14위가 아니라 16위가 된다. 반면, **DENSE_RANK()** 함수는 이와 달리 순위에 1을 더해 다음 순위의 값을 계산한다.

반환 형식:

INT

다음은 역대 올림픽에서 연도별로 금메달을 많이 획득한 국가의 금메달 개수와 순위를 출력하는 예제이다. 공동 순위의 다음 순위 값은 공동 순위의 개수를 더한다.

```
SELECT host_year, nation_code, gold,
RANK() OVER (PARTITION BY host_year ORDER BY gold DESC) AS g_rank
FROM participant;
```

host_year	nation_code	gold	g_rank
1988	'URS'	55	1
1988	'GDR'	37	2
1988	'USA'	36	3
1988	'KOR'	12	4
1988	'HUN'	11	5
1988	'FRG'	11	5
1988	'BUL'	10	7
1988	'ROU'	7	8
1988	'ITA'	6	9
1988	'FRA'	6	9
1988	'KEN'	5	11
1988	'GBR'	5	11
1988	'CHN'	5	11
...			
1988	'CHI'	0	32
1988	'ARG'	0	32
1988	'JAM'	0	32
1988	'SUI'	0	32
1988	'SWE'	0	32
1992	'EUN'	45	1
1992	'USA'	37	2
1992	'GER'	33	3
...			
2000	'RSA'	0	52
2000	'NGR'	0	52
2000	'JAM'	0	52
2000	'BRA'	0	52
2004	'USA'	36	1
2004	'CHN'	32	2
2004	'RUS'	27	3
2004	'AUS'	17	4
2004	'JPN'	16	5
2004	'GER'	13	6
2004	'FRA'	11	7
2004	'ITA'	10	8
2004	'UKR'	9	9
2004	'CUB'	9	9
2004	'GBR'	9	9
2004	'KOR'	9	9
...			
2004	'EST'	0	57
2004	'SLO'	0	57
2004	'SCG'	0	57
2004	'FIN'	0	57
2004	'POR'	0	57
2004	'MEX'	0	57
2004	'LAT'	0	57
2004	'PRK'	0	57

ROW_NUMBER

ROW_NUMBER() OVER ([<partition_by_clause>] <order_by_clause>)

ROW_NUMBER 함수는 분석 함수로만 사용되며, **PARTITION BY** 절에 의한 칼럼 값의 그룹에서 각 행에 고유한 일련번호를 1부터 순서대로 부여하여 **INTEGER** 로 출력한다.

반환 형식:

INT

다음은 역대 올림픽에서 연도별로 금메달을 많이 획득한 국가의 금메달 개수에 따라 일련번호를 출력하되, 금메달 개수가 같은 경우에는 nation_code의 알파벳 순서대로 출력하는 예제이다.

```
SELECT host_year, nation_code, gold,
ROW_NUMBER() OVER (PARTITION BY host_year ORDER BY gold DESC) AS
r_num
FROM participant;
```

host_year	nation_code	gold	r_num
1988	'URS'	55	1
1988	'GDR'	37	2
1988	'USA'	36	3
1988	'KOR'	12	4
1988	'FRG'	11	5
1988	'HUN'	11	6
1988	'BUL'	10	7
1988	'ROU'	7	8
1988	'FRA'	6	9
1988	'ITA'	6	10
1988	'CHN'	5	11
...			
1988	'YEM'	0	152
1988	'YMD'	0	153
1988	'ZAI'	0	154
1988	'ZAM'	0	155
1988	'ZIM'	0	156
1992	'EUN'	45	1
1992	'USA'	37	2
1992	'GER'	33	3
...			
2000	'VIN'	0	194
2000	'YEM'	0	195
2000	'ZAM'	0	196
2000	'ZIM'	0	197
2004	'USA'	36	1
2004	'CHN'	32	2

2004	'RUS'	27	3
2004	'AUS'	17	4
2004	'JPN'	16	5
2004	'GER'	13	6
2004	'FRA'	11	7
2004	'ITA'	10	8
2004	'CUB'	9	9
2004	'GBR'	9	10
2004	'KOR'	9	11
...			
2004	'UGA'	0	195
2004	'URU'	0	196
2004	'VAN'	0	197
2004	'VEN'	0	198
2004	'VIE'	0	199
2004	'VIN'	0	200
2004	'YEM'	0	201
2004	'ZAM'	0	202

STDDEV, STDDEV_POP

STDDEV(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

STDDEV_POP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

STDDEV(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<*analytic_clause*>)

STDDEV_POP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<*analytic_clause*>)

STDDEV 함수와 **STDDEV_POP** 함수는 동일하며, 이 함수는 집계 함수 또는 분석 함수로 사용된다. 이 함수는 모든 행에 대한 연산식 값들에 대한 표준편차, 즉 모표준 편차를 반환한다. **STDDEV_POP** 함수가 SQL:1999 표준이다. 하나의 연산식 *expression* 만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후, 모표준 편차를 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해 모표준 편차를 구한다.

param expression:

수치를 반환하는 하나의 연산식을 지정한다.

param ALL:

모든 값에 대해 표준 편차를 구하기 위해 사용되며, 기본값이다.

param
DISTINCT,DISTINCTROW,UNIQUE:

중복이 제거된 유일한 값에 대해서만 표준 편차를 구하기 위해 사용된다.

rtype:

DOUBLE

리턴 값은 `VAR_POP()` 리턴 값의 제곱근과 같으며 **DOUBLE** 타입이다. 결과 계산에 사용할 행이 없으면 **NULL** 을 반환한다.

다음은 함수에 적용된 공식이다.

$$\text{STDDEV}_{\text{POP}} = \left[\left(\frac{1}{N} \right) * \text{SUM}(\{x_i - \text{AVG}(x)\}^2) \right]^{\frac{1}{2}}$$

⚠ 경고

CUBRID 2008 R3.1 이하 버전에서 **STDDEV** 함수는 `STDDEV_SAMP()` 와 같은 기능을 수행했다.

다음은 전체 과목에 대해 전체 학생의 모표준 편차를 출력하는 예제이다.

```
CREATE TABLE student (name VARCHAR(32), subjects_id INT, score
DOUBLE);
INSERT INTO student VALUES
('Jane',1, 78), ('Jane',2, 50), ('Jane',3, 60),
('Bruce', 1, 63), ('Bruce', 2, 50), ('Bruce', 3, 80),
('Lee', 1, 85), ('Lee', 2, 88), ('Lee', 3, 93),
('Wane', 1, 32), ('Wane', 2, 42), ('Wane', 3, 99),
('Sara', 1, 17), ('Sara', 2, 55), ('Sara', 3, 43);

SELECT STDDEV_POP (score) FROM student;
```

```
stddev_pop(score)
=====
2.329711474744362e+01
```

다음은 각 과목(subjects_id)별로 전체 학생의 점수와 모표준 편차를 함께 출력하는 예제이다.

```

SELECT subjects_id, name, score,
STDDEV_POP(score) OVER(PARTITION BY subjects_id) std_pop
FROM student
ORDER BY subjects_id, name;

```

subjects_id	name	score	std_pop
=====			
=====			
1	'Bruce'	2.632869157402243e+01	6.300000000000000e+01
1	'Jane'	2.632869157402243e+01	7.800000000000000e+01
1	'Lee'	2.632869157402243e+01	8.500000000000000e+01
1	'Sara'	2.632869157402243e+01	1.700000000000000e+01
1	'Wane'	2.632869157402243e+01	3.200000000000000e+01
2	'Bruce'	1.604992211819110e+01	5.000000000000000e+01
2	'Jane'	1.604992211819110e+01	5.000000000000000e+01
2	'Lee'	1.604992211819110e+01	8.800000000000000e+01
2	'Sara'	1.604992211819110e+01	5.500000000000000e+01
2	'Wane'	1.604992211819110e+01	4.200000000000000e+01
3	'Bruce'	2.085185843036539e+01	8.000000000000000e+01
3	'Jane'	2.085185843036539e+01	6.000000000000000e+01
3	'Lee'	2.085185843036539e+01	9.300000000000000e+01
3	'Sara'	2.085185843036539e+01	4.300000000000000e+01
3	'Wane'	2.085185843036539e+01	9.900000000000000e+01

STDDEV_SAMP

STDDEV_SAMP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

STDDEV_SAMP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<analytic_clause>)

STDDEV_SAMP 함수는 집계 함수 또는 분석 함수로 사용되며, 표본 표준편차를 구한다. 하나의 연산식 *expression* 만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후, 표본 표준편차를 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해 표본 표준편차를 구한다.

param expression:

수치를 반환하는 하나의 연산식을 지정한다.

param ALL:

모든 값에 대해 표준 편차를 구하기 위해 사용되며, 기본값이다.

param

DISTINCT, DISTINCTROW, UNIQUE:

중복이 제거된 유일한 값에 대해서만 표준 편차를 구하기 위해 사용된다.

rtype:

DOUBLE

리턴 값은 **VAR_SAMP()** 리턴 값의 제곱근과 같으며 **DOUBLE** 타입이다. 결과 계산에 사용할 행이 없으면 **NULL** 을 반환한다.

다음은 함수에 적용된 공식이다.

$$\text{STDDEV}_{\text{SAMP}} = \left[\left\{ \frac{1}{N-1} \right\} * \text{SUM}(\{x_i - \text{mean}(x)\}^2) \right]^{\frac{1}{2}}$$

다음은 전체 과목에 대해 전체 학생의 표본 표준 편차를 출력하는 예제이다.

```
CREATE TABLE student (name VARCHAR(32), subjects_id INT, score
DOUBLE);
INSERT INTO student VALUES
('Jane', 1, 78), ('Jane', 2, 50), ('Jane', 3, 60),
('Bruce', 1, 63), ('Bruce', 2, 50), ('Bruce', 3, 80),
('Lee', 1, 85), ('Lee', 2, 88), ('Lee', 3, 93),
('Wane', 1, 32), ('Wane', 2, 42), ('Wane', 3, 99),
('Sara', 1, 17), ('Sara', 2, 55), ('Sara', 3, 43);
```

```
SELECT STDDEV_SAMP (score) FROM student;
```

```
      stddev_samp(score)
=====
      2.411480477888654e+01
```

다음은 각 과목(subjects_id)별로 전체 학생의 점수와 표본 표준편차를 함께 출력하는 예제이다.

```
SELECT subjects_id, name, score,
STDDEV_SAMP(score) OVER(PARTITION BY subjects_id) std_samp
FROM student
ORDER BY subjects_id, name;
```

```
      subjects_id  name
score              std_samp
=====
=====
          1  'Bruce'          6.300000000000000e+01
2.943637205907005e+01
          1  'Jane'          7.800000000000000e+01
2.943637205907005e+01
          1  'Lee'           8.500000000000000e+01
2.943637205907005e+01
          1  'Sara'          1.700000000000000e+01
2.943637205907005e+01
          1  'Wane'          3.200000000000000e+01
2.943637205907005e+01
          2  'Bruce'          5.000000000000000e+01
1.794435844492636e+01
          2  'Jane'          5.000000000000000e+01
1.794435844492636e+01
          2  'Lee'           8.800000000000000e+01
1.794435844492636e+01
          2  'Sara'          5.500000000000000e+01
1.794435844492636e+01
          2  'Wane'          4.200000000000000e+01
1.794435844492636e+01
          3  'Bruce'          8.000000000000000e+01
2.331308645374953e+01
          3  'Jane'          6.000000000000000e+01
2.331308645374953e+01
          3  'Lee'           9.300000000000000e+01
2.331308645374953e+01
          3  'Sara'          4.300000000000000e+01
2.331308645374953e+01
```



```
3 'Wane' 9.900000000000000e+01
2.331308645374953e+01
```

SUM

SUM([*DISTINCT* | *DISTINCTROW* | *UNIQUE* | *ALL*] *expression*)

SUM([*DISTINCT* | *DISTINCTROW* | *UNIQUE* | *ALL*] *expression*) OVER (<*analytic_clause*>)

SUM 함수는 집계 함수 또는 분석 함수로 사용되며, 모든 행에 대한 연산식 값들의 합계를 반환한다. 하나의 연산식 *expression* 만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후 합계를 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해 합계를 구한다. 단일 값 수식을 **SUM** 함수의 입력으로 사용할 수 있다.

param *expression*:

수치를 반환하는 하나의 연산식을 지정한다.

param *ALL*:

모든 값에 대해 합계를 구하기 위해 사용되며, 기본으로 지정된다.

param
DISTINCT, *DISTINCTROW*, *UNIQUE*:

중복이 제거된 유일한 값에 대해서만 합계를 구하기 위해 사용된다.

rtype:

*expression*의 타입

다음은 *demodb* 에서 역대 올림픽에서 획득한 금메달 수의 합계를 기준으로 10위권 국가와 금메달 총 수를 출력하는 예제이다.

```
SELECT nation_code, SUM(gold)
FROM participant
GROUP BY nation_code
ORDER BY SUM(gold) DESC
LIMIT 10;
```

nation_code	sum(gold)
'USA'	190
'CHN'	97
'RUS'	85
'GER'	79
'URS'	55
'FRA'	53
'AUS'	52
'ITA'	48
'KOR'	48
'EUN'	45

다음은 *demodb* 에서 nation_code가 'AU'로 시작하는 국가에 대해 연도별로 획득한 금메달 수와 해당 연도까지의 금메달 누적 합계를 출력하는 예제이다.

```
SELECT host_year, nation_code, gold,
       SUM(gold) OVER (PARTITION BY nation_code ORDER BY host_year)
       sum_gold
FROM participant
WHERE nation_code LIKE 'AU%';
```

host_year	nation_code	gold	sum_gold
1988	'AUS'	3	3
1992	'AUS'	7	10
1996	'AUS'	9	19
2000	'AUS'	16	35
2004	'AUS'	17	52
1988	'AUT'	1	1
1992	'AUT'	0	1
1996	'AUT'	0	1
2000	'AUT'	2	3
2004	'AUT'	2	5

다음은 위 예제에서 **OVER** 함수 이하의 "ORDER BY host_year" 절을 제거한 것으로, sum_gold의 값은 모든 연도의 금메달 합계로 각 연도에서 모두 같은 값을 가진다.

```
SELECT host_year, nation_code, gold, SUM(gold) OVER (PARTITION BY
nation_code) sum_gold
FROM participant
WHERE nation_code LIKE 'AU%';
```

host_year	nation_code	gold	sum_gold

2004	'AUS'	17	52
2000	'AUS'	16	52
1996	'AUS'	9	52
1992	'AUS'	7	52
1988	'AUS'	3	52
2004	'AUT'	2	5
2000	'AUT'	2	5
1996	'AUT'	0	5
1992	'AUT'	0	5
1988	'AUT'	1	5

VARIANCE, VAR_POP

VARIANCE(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

VAR_POP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

VARIANCE(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<*analytic_clause*>)

VAR_POP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<*analytic_clause*>)

VARIANCE 함수와 **VAR_POP** 함수는 동일하며, 집계 함수 또는 분석 함수로 사용된다. 이 함수는 모든 행에 대한 연산식 값들에 대한 분산, 즉 모분산을 반환한다. 분모는 모든 행의 개수이다. 하나의 연산식 *expression* 만 인자로 지정되며, 연산식 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후, 모분산을 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해 모분산을 구한다.

param expression:

수치를 반환하는 하나의 연산식을 지정한다.

param ALL:

모든 값에 대해 모분산을 구하기 위해 사용되며, 기본값이다.

param

DISTINCT, DISTINCTROW, UNIQUE:

중복이 제거된 유일한 값에 대해서만 모분산을 구하기 위해 사용된다.

rtype:

DOUBLE

리턴 값은 **DOUBLE** 타입이며, 결과 계산에 사용할 행이 없으면 **NULL** 을 반환한다.

다음은 함수에 적용된 공식이다.

$$\text{VAR}_{\text{POP}} = \left(\frac{1}{N} \right) * \text{SUM}(\{x_i - \text{AVG}(x)\}^2)$$

참고

CUBRID 2008 R3.1 이하 버전에서 **VARIANCE** 함수는 **VAR_SAMP()** 와 같은 기능을 수행했다.

다음은 전체 과목에 대해 전체 학생의 모분산을 출력하는 예제이다.

```
CREATE TABLE student (name VARCHAR(32), subjects_id INT, score
DOUBLE);
INSERT INTO student VALUES
('Jane', 1, 78), ('Jane', 2, 50), ('Jane', 3, 60),
('Bruce', 1, 63), ('Bruce', 2, 50), ('Bruce', 3, 80),
('Lee', 1, 85), ('Lee', 2, 88), ('Lee', 3, 93),
('Wane', 1, 32), ('Wane', 2, 42), ('Wane', 3, 99),
('Sara', 1, 17), ('Sara', 2, 55), ('Sara', 3, 43);

SELECT VAR_POP(score) FROM student;
```

```
var_pop(score)
=====
5.42755555555550e+02
```

다음은 각 과목(subjects_id)별로 전체 학생의 점수와 모분산을 함께 출력하는 예제이다.

```
SELECT subjects_id, name, score, VAR_POP(score) OVER(PARTITION BY
subjects_id) v_pop
FROM student
ORDER BY subjects_id, name;
```

```
subjects_id  name
score          v_pop
=====
=====
1  'Bruce'          6.30000000000000e+01
```

```

6.93199999999998e+02      1  'Jane'      7.800000000000000e+01
6.93199999999998e+02      1  'Lee'       8.500000000000000e+01
6.93199999999998e+02      1  'Sara'     1.700000000000000e+01
6.93199999999998e+02      1  'Wane'     3.200000000000000e+01
6.93199999999998e+02      2  'Bruce'    5.000000000000000e+01
2.57599999999999e+02      2  'Jane'    5.000000000000000e+01
2.57599999999999e+02      2  'Lee'     8.800000000000000e+01
2.57599999999999e+02      2  'Sara'    5.500000000000000e+01
2.57599999999999e+02      2  'Wane'    4.200000000000000e+01
2.57599999999999e+02      3  'Bruce'    8.000000000000000e+01
4.348000000000002e+02      3  'Jane'    6.000000000000000e+01
4.348000000000002e+02      3  'Lee'     9.300000000000000e+01
4.348000000000002e+02      3  'Sara'    4.300000000000000e+01
4.348000000000002e+02      3  'Wane'    9.900000000000000e+01
4.348000000000002e+02

```

VAR_SAMP

VAR_SAMP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*)

VAR_SAMP(*[DISTINCT | DISTINCTROW | UNIQUE | ALL] expression*) OVER (<*analytic_clause*>)

VAR_SAMP 함수는 집계 함수 또는 분석 함수로 사용되며, 표본 분산을 반환한다. 분모는 모든 행의 개수 - 1이다. 하나의 *expression*만 인자로 지정되며, *expression* 앞에 **DISTINCT** 또는 **UNIQUE** 키워드를 포함시키면 연산식 값 중 중복을 제거한 후, 표본 분산을 구하고, 키워드가 생략되거나 **ALL** 인 경우에는 모든 값에 대해 표본 분산을 구한다.

param expression:

수치를 반환하는 하나의 연산식을 지정한다.

param ALL:

모든 값에 대해 표본 분산을 구하기 위해 사용되며, 기본값이다.

param
DISTINCT,DISTINCTROW,UNIQUE:

중복이 제거된 유일한 값에 대해서만 표본 분산을 구하기 위해 사용된다.

rtype:

DOUBLE

리턴 값은 **DOUBLE** 타입이며, 결과 계산에 사용할 행이 없으면 **NULL** 을 반환한다.

다음은 함수에 적용된 공식이다.

$$\text{VAR}_{\text{SAMP}} = \left\{ \frac{1}{N-1} \right\} * \text{SUM}(\{x_i - \text{mean}(x)\}^2)$$

다음은 전체 과목에 대해 전체 학생의 표본 분산을 출력하는 예제이다.

```
CREATE TABLE student (name VARCHAR(32), subjects_id INT, score
DOUBLE);
INSERT INTO student VALUES
('Jane',1, 78), ('Jane',2, 50), ('Jane',3, 60),
('Bruce', 1, 63), ('Bruce', 2, 50), ('Bruce', 3, 80),
('Lee', 1, 85), ('Lee', 2, 88), ('Lee', 3, 93),
('Wane', 1, 32), ('Wane', 2, 42), ('Wane', 3, 99),
('Sara', 1, 17), ('Sara', 2, 55), ('Sara', 3, 43);

SELECT VAR_SAMP(score) FROM student;
```

```
var_samp(score)
=====
5.815238095238092e+02
```

다음은 각 과목(subjects_id)별로 전체 학생의 점수와 표본 분산을 함께 출력하는 예제이다.

```
SELECT subjects_id, name, score, VAR_SAMP(score) OVER(PARTITION BY
subjects_id) v_samp
FROM student
ORDER BY subjects_id, name;
```

```

subjects_id  name
score                v_samp
=====
=====
          1  'Bruce'          6.300000000000000e+01
8.665000000000000e+02
          1  'Jane'           7.800000000000000e+01
8.665000000000000e+02
          1  'Lee'            8.500000000000000e+01
8.665000000000000e+02
          1  'Sara'           1.700000000000000e+01
8.665000000000000e+02
          1  'Wane'           3.200000000000000e+01
8.665000000000000e+02
          2  'Bruce'          5.000000000000000e+01
3.220000000000000e+02
          2  'Jane'           5.000000000000000e+01
3.220000000000000e+02
          2  'Lee'            8.800000000000000e+01
3.220000000000000e+02
          2  'Sara'           5.500000000000000e+01
3.220000000000000e+02
          2  'Wane'           4.200000000000000e+01
3.220000000000000e+02
          3  'Bruce'          8.000000000000000e+01
5.435000000000000e+02
          3  'Jane'           6.000000000000000e+01
5.435000000000000e+02
          3  'Lee'            9.300000000000000e+01
5.435000000000000e+02
          3  'Sara'           4.300000000000000e+01
5.435000000000000e+02
          3  'Wane'           9.900000000000000e+01
5.435000000000000e+02

```

JSON_ARRAYAGG

JSON_ARRAYAGG(*json_val*)

평가된 행으로부터 JSON 배열을 만드는 집계 함수이다.

```

CREATE TABLE t_score(name VARCHAR(10), score INT);
INSERT INTO t_score VALUES
  ('Amie', 60),
  ('Jane', 80),
  ('Lora', 60),
  ('James', 75),
  ('Peter', 70),

```

```

('Tom', 30),
('Ralph', 99),
('David', 55),
('Amie', 65);

```

```

SELECT JSON_ARRAYAGG (name) AS test_takers from t_score;

```

```

test_takers
=====
["Amie","Jane","Lora","James","Peter","Tom","Ralph","David","Amie"]

```

JSON_OBJECTAGG

JSON_OBJECTYAGG(*key*, *json_val expr*)

각 행의 평가에서 수집된 (key, json_val) 표현식으로부터 JSON 객체를 생성한다.

```

CREATE TABLE t_score(name VARCHAR(10), score INT);
INSERT INTO t_score VALUES
  ('Amie', 60),
  ('Jane', 80),
  ('Lora', 60),
  ('James', 75),
  ('Peter', 70),
  ('Tom', 30),
  ('Ralph', 99),
  ('David', 55);

```

```

SELECT JSON_OBJECTAGG (name, score) AS test_scores from t_score;

```

```

test_scores
=====
{"Amie":60,"Jane":80,"Lora":60,"James":75,"Peter":70,"Tom":
30,"Ralph":99,"David":55}

```

클릭 카운터 함수

목차

클릭 카운터 함수

7

- INCR, DECR

677

INCR, DECR

INCR(*column*)

DECR(*column*)

INCR 함수는 **SELECT** 절에 포함되어 인자로 주어진 칼럼의 값을 1 증가시켜 주는 기능을 한다.

DECR 함수는 해당 칼럼의 값을 1 감소시킨다.

매개변수:

column – SMALLINT, INT 또는 BIGINT 타입의 칼럼 이름

반환 형식:

SMALLINT, INT 또는 BIGINT

INCR 함수와 **DECR** 함수는 '클릭 카운터' 함수로 불리며, 게시판 유형의 웹 서비스에서 게시물의 조회수를 증가시키는데 유용하게 사용될 수 있다. 게시물의 내용을 **SELECT** 하고 곧바로 게시물의 조회수를 **UPDATE** 로 1 증가하는 유형의 시나리오에서 하나의 **SELECT** 문에 **INCR** 함수를 사용함으로써 한 번에 게시물 내용 조회와 조회수 증가 작업을 수행할 수 있다.

INCR 함수는 인자로 명시된 칼럼 값을 증가시킨다. 단, 인자로는 정수 타입의 숫자형만 올 수 있고, 값이 **NULL** 인 경우 **INCR** 함수를 수행하여도 값은 **NULL** 을 유지한다. 즉, 값이 설정되어야 **INCR** 함수를 써서 값을 증가시킬 수 있다. **DECR** 함수는 인자로 명시된 칼럼 값을 감소시킨다.

SELECT 절에 **INCR** 함수를 명시한 경우, 카운터 값은 1이 증가하고 질의 결과는 증가하기 전의 값으로 출력한다. 그리고, **INCR** 함수는 질의 처리 과정에서 참여한 행(tuple)이 아니라 최종 결과에 참여한 행에 대해서만 값을 증가시킨다.

SELECT 리스트에 **INCR** 또는 **DECR** 함수를 명시하지 않고 클릭 카운터를 증가 또는 감소시키고자 하는 경우, **WHERE** 절 뒤에 **WITH INCREMENT FOR column** 또는 **WITH DECREMENT FOR column** 을 명시하면 된다.

```
CREATE TABLE board (id INT, cnt INT, content VARCHAR(8096));
SELECT content FROM board WHERE id=1 WITH INCREMENT FOR cnt;
```

참고

- **INCR/DECR** 함수는 사용자 정의 트랜잭션과 별도로 시스템 내부에서 사용되는 top operation 이 적용되어 트랜잭션의 **COMMIT/ROLLBACK** 과 상관없이 데이터베이스에 자동으로 적용된다.
- 하나의 **SELECT** 문에 **INCR/DECR** 함수를 여러 개 사용할 경우, 해당 질의 내의 각각의 **INCR/DECR** 함수 중 하나라도 실패하면 모두 실패한다.
- **INCR/DECR** 함수는 최상위 **SELECT** 문에만 적용된다. **INSERT ... SELECT ...** 구문과 **UPDATE table SET col = SELECT ...** 등과 같은 **SUB SELECT** 문은 지원하지 않는다. 다음은 **INCR** 함수가 허용되지 않는 예이다.

```
SELECT b.content, INCR(b.read_count) FROM (SELECT * FROM board
WHERE id = 1) AS b
```

- **INCR/DECR** 함수가 포함된 **SELECT** 문의 경우, 결과 행의 개수가 둘 이상이면 오류로 처리한다. 최종 결과가 하나인 경우에만 유효하다.
- **INCR/DECR** 함수는 숫자 타입에 대해서만 사용할 수 있다. 적용 가능한 타입은 **SMALLINT**, **INTEGER**, **BIGINT**와 같은 정수형 데이터 타입으로 제한된다. 기타 타입에는 사용할 수 없다.
- **INCR** 함수 호출 시 결과 값은 현재 값이며, 저장 값은 현재 값 +1인 값이 저장된다. 결과를 저장 값과 같은 값을 조회하고자 할 경우는 다음과 같이 수행한다.

```
SELECT content, INCR(read_count) + 1 FROM board WHERE id = 1;
```

- 정의된 타입의 최대값을 초과할 경우 **INCR** 함수는 해당 칼럼을 0으로 초기화 한다. 반대로 최소값에 **DECR** 함수가 적용되어도 0으로 초기화된다.
- **INCR / DECR** 함수는 **UPDATE** 트리거와 무관하게 실행되므로 데이터 일관성이 보장되지 않을 수 있다. 다음은 **INCR** 함수가 **UPDATE** 트리거와 무관하게 실행되기 때문에 데이터베이스의 일관성이 위반되는 예이다.

```
CREATE TRIGGER event_tr BEFORE UPDATE ON event EXECUTE REJECT;
SELECT INCR(players) FROM event WHERE gender='M';
```

- **INCR / DECR** 함수는 HA 구성의 슬레이브 노드나 read-only 모드의 CSQL 인터프리터(csql -r) 또는 Read Only, Standby Only 모드처럼 쓰기가 금지된 모드(cubrid_broker.conf의 ACCESS_MODE=RO 또는 SO)의 브로커에서 사용 시 오류를 반환한다.

예제

먼저, board 테이블에는 아래와 같이 3건의 데이터가 입력되었다고 가정한다.

```
CREATE TABLE board (
  id INT,
  title VARCHAR(100),
  content VARCHAR(4000),
  read_count INT
);
INSERT INTO board VALUES (1, 'aaa', 'text...', 0);
INSERT INTO board VALUES (2, 'bbb', 'text...', 0);
INSERT INTO board VALUES (3, 'ccc', 'text...', 0);
```

다음은 id 값이 1인 데이터의 read_count 칼럼의 값을 **INCR** 함수로 증가시키는 예이다.

```
SELECT content, INCR(read_count) FROM board WHERE id = 1;
```

content	read_count
'text...'	0

예와 같이 **SELECT** 문에 **INCR** 함수를 사용함으로써 해당 칼럼 값은 read_count + 1이 된다. 결과는 다음과 같은 **SELECT** 문을 통해 확인해 볼 수 있다.

```
SELECT content, read_count FROM board WHERE id = 1;
```

content	read_count
'text...'	1

ROWNUM 함수

ROWNUM, INST_NUM

ROWNUM

INST_NUM()

ROWNUM 함수는 질의 결과로 생성될 각 레코드에 대한 순서를 나타내는 번호를 반환한다. 첫 번째 결과 레코드는 1, 두 번째 결과 레코드는 2를 가진다.

반환 형식:

BIGINT

일반적인 **SELECT** 문에서는 **ROWNUM**, **INST_NUM()**을, **ORDER BY** 절을 포함한 **SELECT** 문에서는 **ORDERBY_NUM()**을, **GROUP BY** 절을 포함한 **SELECT** 문에서는 **GROUPBY_NUM()**을 사용할 수 있다. **ROWNUM** 함수를 사용하면 질의의 결과 레코드 수를 다양한 방법으로 제한할 수 있다. 예를 들어, 질의 결과의 처음 10건만 조회한다거나, 짝수 번째 또는 홀수 번째 레코드만 반환하도록 할 수 있다.

ROWNUM 함수는 결과 타입이 큰 정수형이고, **SELECT** 절과 **WHERE** 절과 같이 질의 내에 수식이 위치할 수 있는 모든 곳에 사용할 수 있다. 하지만, **ROWNUM** 함수 결과를 속성 또는 연관된 부질의 (correlated subquery)와 비교하는 것은 허용되지 않는다.

참고

- **WHERE** 절에 명시된 **ROWNUM** 함수는 **INST_NUM()** 함수와 같은 의미를 가진다.
- **ROWNUM** 함수는 각각의 **SELECT** 문장에 종속된다. 즉, **ROWNUM** 함수가 부질의에 쓰인 경우, 부질의를 수행하는 동안에 부질의 결과에 대하여 일련 번호를 반환한다. 내부적으로, **ROWNUM** 함수 결과는 조회된 레코드를 질의 결과 셋에 쓰기 직전에 생성된다. 이 순간에 질의 결과 셋의 레코드에 대한 일련 번호를 생성하는 카운터 값이 증가된다.
- **SELECT** 결과에 일련번호를 부여할 목적으로 사용하는 경우, 정렬 과정이 없는 경우 **ROWNUM**을, **ORDER BY** 절이 있으면 **ORDERBY_NUM()** 함수를, **GROUP BY** 절이 있으면 **GROUPBY_NUM()** 함수를 사용한다.
- **SELECT** 문에 **ORDER BY** 절이 포함된 경우 **WHERE** 절에 명시된 **ORDERBY_NUM()** 함수의 값은 **ORDER BY** 절 처리를 위한 정렬 과정 이후에 생성된다. 정렬 과정 이후에 결과 행의 개수를 제한하려면 **FOR ORDERBY_NUM()** 절을 사용한다.
- **SELECT** 문에 **GROUP BY** 절이 포함된 경우 **HAVING** 절에 명시된 **GROUPBY_NUM()** 함수의 값은 질의 결과가 그룹화된 이후에 생성된다. 그룹화된 이후에 결과 행의 개수를 제한하려면 **HAVING GROUPBY_NUM()** 절을 사용한다.
- 정렬된 결과 행들의 개수를 제한하는 목적으로 **FOR ORDERBY_NUM()** 또는 **HAVING GROUPBY_NUM()** 구문 대신 **LIMIT** 절을 사용할 수 있다.
- **ROWNUM** 함수는 **SELECT** 문 뿐만 아니라 **INSERT**, **DELETE**, **UPDATE** 와 같은 SQL 문에도 쓸 수 있다. 예를 들어, **INSERT INTO table_name SELECT ... FROM ... WHERE ...** 질의와 같이 한 테이블의 행(row) 중 일부를 조회하여 다른 테이블에 삽입하고자 할 때, **WHERE** 절에 **ROWNUM** 함수를 사용할 수 있다.

다음은 *demodb* 에서 1988 올림픽에서 금메달 개수를 기준으로 4위권 국가 이름을 반환하는 예제이다.

```
--Limiting 4 rows using ROWNUM in the WHERE condition
SELECT * FROM
(SELECT nation_code FROM participant WHERE host_year = 1988
ORDER BY gold DESC) AS T
WHERE ROWNUM <5;
```

```
nation_code
=====
'URS'
'GDR'
'USA'
'KOR'
```

LIMIT 절은 정렬된 결과 행의 개수를 제한하므로 아래의 질의 결과는 위의 결과와 동일하다.

```
--Limiting 4 rows using LIMIT
SELECT ROWNUM, nation_code FROM participant WHERE host_year = 1988
ORDER BY gold DESC
LIMIT 4;
```

```
rownum nation_code
=====
143 'URS'
51 'GDR'
145 'USA'
78 'KOR'
```

아래의 ROWNUM 조건은 정렬되기 이전에 행의 개수를 제한하므로 질의 결과가 위와 다르다.

```
--Unexpected results : ROWNUM operated before ORDER BY
SELECT ROWNUM, nation_code FROM participant
WHERE host_year = 1988 AND ROWNUM < 5
ORDER BY gold DESC;
```

```
rownum nation_code
=====
1 'AFG'
2 'AHO'
3 'AND'
4 'ANG'
```

ORDERBY_NUM

ORDERBY_NUM()

ORDERBY_NUM() 함수는 **ROWNUM** 혹은 **INST_NUM()** 함수와 함께, 결과 행들의 개수를 제한하는 목적으로 사용된다. 단, 차이점은 **ORDER BY** 절 뒤에 결합되어 사용되고, 이미 정렬을 수행한 결과에 대해 순서를 부여한다는 점이다. 즉, **ORDER BY** 절이 포함된 **SELECT** 문장에서 조건 절에 **ROWNUM** 을 이용하여 일부 결과 행들만 조회하는 경우, **ROWNUM** 이 먼저 적용된 후 **ORDER BY** 에 의한 정렬이 수행된다. 반면, **ORDERBY_NUM()** 함수를 이용하여 일부 결과 행들만 조회하는 경우, **ORDER BY** 에 의한 정렬이 이루어진 결과에 대해서 **ROWNUM** 이 적용된다.

반환 형식:

BIGINT

다음은 *demodb* 의 *history* 테이블에서 3위에서 5위까지의 선수 이름과 기록을 조회하는 예제이다.

```
--Ordering first and then limiting rows using FOR ORDERBY_NUM()
SELECT ORDERBY_NUM(), athlete, score FROM history
ORDER BY score FOR ORDERBY_NUM() BETWEEN 3 AND 5;
```

orderby_num()	athlete	score
3	'Luo Xuejuan'	'01:07.0'
4	'Rodal Vebjorn'	'01:43.0'
5	'Thorpe Ian'	'01:45.0'

아래의 **LIMIT** 절을 사용한 질의는 위의 질의와 동일한 결과를 출력한다.

```
SELECT ORDERBY_NUM(), athlete, score FROM history
ORDER BY score LIMIT 2, 3;
```

아래의 **ROWNUM**을 사용하여 결과 행의 개수를 제한한 질의는 정렬 이전에 개수를 제한한 이후에 **ORDER BY** 정렬을 수행한다.

```
--Limiting rows first and then Ordering using ROWNUM
SELECT ROWNUM athlete, score FROM history
WHERE ROWNUM BETWEEN 3 AND 5 ORDER BY score;
```

athlete	score
'Thorpe Ian'	'01:45.0'

```
'Thorpe Ian'      '03:41.0'
'Hackett Grant'   '14:43.0'
```

GROUPBY_NUM

GROUPBY_NUM()

GROUPBY_NUM() 함수는 **ROWNUM** 혹은 **INST_NUM()** 함수와 함께, 결과 행들의 개수를 제한하는 목적으로 사용된다. 단, 차이점은 **GROUP BY ... HAVING** 절 뒤에 결합되어 사용되며, 이미 정렬을 수행한 결과에 대해 순서를 부여한다는 점이다. 또한, **INST_NUM()** 함수는 스칼라(scalar) 함수이지만, **GROUPBY_NUM()** 함수는 집계 함수의 일종이다.

즉, **GROUP BY** 절이 포함된 **SELECT** 문장에서 조건 절에 **ROWNUM** 을 이용하여 일부 결과 행들만 조회하는 경우, **ROWNUM** 이 먼저 적용된 후 **GROUP BY** 에 의한 그룹 정렬이 수행된다. 반면, **GROUPBY_NUM()** 함수를 이용하여 일부 결과 행들만 조회하는 경우, **GROUP BY** 에 의한 그룹 정렬이 이루어진 결과에 대해서 **ROWNUM** 이 적용된다.

반환 형식:

BIGINT

다음은 *demodb* 의 *history* 테이블에서 과거 5개의 올림픽에 대해서 최단 기록을 조회하는 예제이다.

```
--Group-ordering first and then limiting rows using GROUPBY_NUM()
SELECT  GROUPBY_NUM(), host_year, MIN(score) FROM history
GROUP BY host_year HAVING GROUPBY_NUM() BETWEEN 1 AND 5;
```

groupby_num()	host_year	min(score)
1	1968	'8.9'
2	1980	'01:53.0'
3	1984	'13:06.0'
4	1988	'01:58.0'
5	1992	'02:07.0'

아래의 **LIMIT** 절을 사용한 질의는 위의 질의와 동일한 결과를 출력한다.

```
SELECT  GROUPBY_NUM(), host_year, MIN(score) FROM history
GROUP BY host_year LIMIT 5;
```

아래의 **ROWNUM**을 사용하여 결과 행의 개수를 제한한 질의는 그룹핑 이전에 개수를 제한한 이후에 **GROUP BY** 정렬을 수행한다.

```
--Limiting rows first and then Group-ordering using ROWNUM
SELECT host_year, MIN(score) FROM history
WHERE ROWNUM BETWEEN 1 AND 5 GROUP BY host_year;
```

```
host_year  min(score)
=====
2000      '03:41.0'
2004      '01:45.0'
```

정보 함수

목차

정보 함수	7
● CHARSET	685
● COERCIBILITY	686
● COLLATION	686
● CURRENT_USER, USER	687
● DATABASE, SCHEMA, CURRENT_SCHEMA	688
● DBTIMEZONE	689
● DEFAULT	689
● DISK_SIZE	690
● ESTIMATED_AVG_ROW_LENGTH	691
● ESTIMATED_DATA_FREE	691
● ESTIMATED_DATA_LENGTH	692
● ESTIMATED_TABLE_ROWS	693

● INDEX_CARDINALITY	693
● INET_ATON	695
● INET_NTOA	696
● LAST_INSERT_ID	696
● LIST_DBS	699
● ROW_COUNT	700
● SESSIONTIMEZONE	701
● USER, SYSTEM_USER	701
● VERSION	702

CHARSET

CHARSET(*expr*)

expr 의 문자셋을 반환한다.

매개변수:

expr - 문자셋을 구할 대상 표현식

반환 형식:

STRING

```
SELECT CHARSET('abc');
```

```
'iso88591'
```

```
SELECT CHARSET(_utf8'abc');
```

```
'utf8'
```

```
SET NAMES utf8;
SELECT CHARSET('abc');
```

```
'utf8'
```

COERCIBILITY

COERCIBILITY(*expr*)

*expr*의 콜레이션 변환도(coercibility)를 반환한다. 콜레이션 변환도는 칼럼(표현식)들이 서로 다른 콜레이션과 문자셋을 가지고 있을 때 어떤 콜레이션과 문자셋으로 변환할 것인지를 결정한다. 어떤 연산을 수행하는 두 개의 칼럼(표현식)이 있을 때, 높은 변환도를 가진 인자는 더 낮은 변환도를 가진 인자의 콜레이션으로 변환된다. 이와 관련하여 **콜레이션 변환도**를 참고한다.

매개변수:

expr – 콜레이션 변환도를 구할 대상 표현식

반환 형식:

INT

```
SELECT COERCIBILITY(USER());
```

```
7
```

```
SELECT COERCIBILITY(_utf8'abc');
```

```
10
```

COLLATION

COLLATION(*expr*)

*expr*의 콜레이션을 반환한다.

매개변수:

expr – 콜레이션을 구할 대상 표현식

반환 형식:

STRING

```
SELECT COLLATION('abc');
```

```
'iso88591_bin'
```

```
SELECT COLLATION(_utf8'abc');
```

```
'utf8_bin'
```

CURRENT_USER, USER

CURRENT_USER

USER

CURRENT_USER와 **USER** 의사 칼럼(pseudo column)은 동일하며, 현재 데이터베이스에 로그인한 사용자의 이름을 문자열로 반환한다.

기능이 비슷한 **SYSTEM_USER()** 함수와 **USER()** 함수는 사용자 이름을 CSQL 또는 CAS가 실행된 호스트 이름과 함께 반환한다.

반환 형식:

STRING

```
--selecting the current user on the session  
SELECT USER;
```

```

CURRENT_USER
=====
'PUBLIC'

```

```
SELECT USER(), CURRENT_USER;
```

```

user()                CURRENT_USER
=====
'PUBLIC@cubs006.cub'  'PUBLIC'

```

```

--selecting all users of the current database from the system table
SELECT name, id, password FROM db_user;

```

name	id	password
'DBA'	NULL	NULL
'PUBLIC'	NULL	NULL
'SELECT_ONLY_USER'	NULL	db_password
'ALMOST_DBA_USER'	NULL	db_password
'SELECT_ONLY_USER2'	NULL	NULL

DATABASE, SCHEMA, CURRENT_SCHEMA

DATABASE()

SCHEMA()

CURRENT_SCHEMA()

DATABASE 함수는 현재 연결된 데이터베이스 이름을 **VARCHAR** 타입의 문자열로 반환한다.

SCHEMA 함수와 **CURRENT_SCHEMA** 함수는 동일하며, 현재 스키마 이름을 **VARCHAR** 타입의 문자열로 반환한다. 사용자 스키마를 지원하는 11.2 버전부터 스키마 이름은 현재 사용자 이름과 같다.

반환 형식:

STRING

```
SELECT DATABASE(), SCHEMA(), CURRENT_SCHEMA();
```

```

database()          schema()          current_schema()
=====
'demodb'           'DBA'           'DBA'

```

DBTIMEZONE

DBTIMEZONE()

데이터베이스 서버의 타임존(오프셋 또는 지명)을 문자열로 출력한다(예: '-05:00', 또는 'Europe/Vienna').

```
SELECT DBTIMEZONE();
```

```

dbtimezone
=====
'Asia/Seoul'

```

❶ 더 보기

SESSIONTIMEZONE(), FROM_TZ(), NEW_TIME(), TZ_OFFSET()

DEFAULT

DEFAULT(*column_name*)

DEFAULT

DEFAULT와 **DEFAULT** 함수는 칼럼에 정의된 기본값을 반환한다. 해당 칼럼에 기본값이 지정되지 않으면 **NULL** 또는 에러를 출력한다. **DEFAULT**는 인자가 없는 반면, **DEFAULT** 함수는 칼럼 이름을 입력 인자로 하는 차이가 있다. **DEFAULT**는 **INSERT** 문의 입력 데이터, **UPDATE** 문의 **SET** 절에서 사용될 수 있고, **DEFAULT** 함수는 모든 곳에서 사용될 수 있다.

기본값이 정의되지 않은 칼럼에 어떠한 제약 조건이 정의되어 있지 않거나 **UNIQUE** 제약 조건이 정의된 경우에는 **NULL**을 반환하고, 해당 칼럼에 **NOT NULL** 또는 **PRIMARY KEY** 제약 조건이 정의된 경우에는 에러를 반환한다.

```

CREATE TABLE info_tbl(id INT DEFAULT 0, name VARCHAR);
INSERT INTO info_tbl VALUES (1,'a'),(2,'b'),(NULL,'c');

SELECT id, DEFAULT(id) FROM info_tbl;

```

id	default(id)
1	0
2	0
NULL	0

```
UPDATE info_tbl SET id = DEFAULT WHERE id IS NULL;
DELETE FROM info_tbl WHERE id = DEFAULT(id);
INSERT INTO info_tbl VALUES (DEFAULT, 'd');
```

참고

CUBRID 9.0 미만 버전에서는 테이블 생성 시 DATE, DATETIME, TIME, TIMESTAMP 칼럼의 DEFAULT 값을 SYS_DATE, SYS_DATETIME, SYS_TIME, SYS_TIMESTAMP로 지정하면, CREATE TABLE 시점의 값이 저장된다. 따라서 데이터가 INSERT되는 시점의 값을 입력하려면 INSERT 구문의 VALUES 절에 해당 함수를 입력해야 한다.

DISK_SIZE

DISK_SIZE(*expr*)

이 함수는 *expr* 값을 저장하는 데 필요한 바이트 크기를 반환한다. 주로 데이터베이스 힙 파일에 값을 저장하는 데 필요한 크기를 확인할 때 사용한다.

매개변수:

expr – 연산식

반환 형식:

INTEGER

```
SELECT DISK_SIZE('abc'), DISK_SIZE(1);
```

disk_size('abc')	disk_size(1)
7	4

값의 실제 내용에 따라 크기가 다르며, **문자열 압축** 도 고려한다.

```
CREATE TABLE t1(s1 VARCHAR(10), s2 VARCHAR(300), c1 CHAR(10), c2
CHAR(300));
INSERT INTO t1 VALUES(REPEAT('a', 10), REPEAT('b', 300), REPEAT('c',
10), REPEAT('d', 300));
INSERT INTO t1 VALUES('a', 'b', 'c', 'd');
SELECT DISK_SIZE(s1), DISK_SIZE(s2), DISK_SIZE(c1), DISK_SIZE(c2)
FROM t1;
```

disk_size(s1)	disk_size(s2)	disk_size(c1)	disk_size(c2)
12	24	10	300
4	4	10	300

ESTIMATED_AVG_ROW_LENGTH

ESTIMATED_AVG_ROW_LENGTH(*table_name*)

ESTIMATED_AVG_ROW_LENGTH 함수는 *table_name*으로 식별되는 테이블의 추정 평균 행 길이를 바이트 단위로 반환한다. 이 값은 추정치임에 유의한다.

매개변수:

table_name – 테이블 이름

반환 형식:

BIGINT

*table_name*이 **NULL**이거나, 테이블이 존재하지 않거나 뷰인 경우 **NULL**을 반환한다.

```
SELECT ESTIMATED_AVG_ROW_LENGTH('game');
```

estimated_avg_row_length('game')
48

ESTIMATED_DATA_FREE

ESTIMATED_DATA_FREE(*table_name*)

ESTIMATED_DATA_FREE 함수는 *table_name*으로 식별되는 테이블에 할당된 힙 페이지 내의 추정 여유(미사용) 공간을 바이트 단위로 반환한다. 이 값은 추정치임에 유의한다.

매개변수:

table_name – 테이블 이름

반환 형식:

BIGINT

*table_name*이 **NULL**이거나, 테이블이 존재하지 않거나 뷰인 경우 **NULL**을 반환한다.

```
SELECT ESTIMATED_DATA_FREE('game');
```

```
estimated_data_free('game')
=====
                           91320
```

ESTIMATED_DATA_LENGTH

ESTIMATED_DATA_LENGTH(*table_name*)

ESTIMATED_DATA_LENGTH 함수는 *table_name*으로 식별되는 테이블에 할당된 힙의 추정 총 데이터 길이를 바이트 단위로 반환한다. 이 값은 추정치임에 유의한다.

매개변수:

table_name – 테이블 이름

반환 형식:

BIGINT

*table_name*이 **NULL**이거나, 테이블이 존재하지 않거나 뷰인 경우 **NULL**을 반환한다.

```
SELECT ESTIMATED_DATA_LENGTH('game');
```



```
estimated_data_length('game')
=====
506664
```

ESTIMATED_TABLE_ROWS

ESTIMATED_TABLE_ROWS(*table_name*)

ESTIMATED_TABLE_ROWS 함수는 *table_name*으로 식별되는 테이블의 추정 행 수를 반환한다. 이 값은 추정치임에 유의한다.

매개변수:

table_name – 테이블 이름

반환 형식:

BIGINT

*table_name*이 **NULL**이거나, 테이블이 존재하지 않거나 뷰인 경우 **NULL**을 반환한다.

```
SELECT ESTIMATED_TABLE_ROWS('game');
```

```
estimated_table_rows('game')
=====
8653
```

INDEX_CARDINALITY

INDEX_CARDINALITY(*table_name*, *index*, *key_pos*)

INDEX_CARDINALITY 함수는 테이블에서 인덱스 카디널리티(cardinality)를 반환한다. 인덱스 카디널리티는 인덱스를 정의하는 고유한 값의 개수이다. 인덱스 카디널리티는 다중 칼럼 인덱스의 부분 키에 대해서도 적용할 수 있는데, 이때 세 번째 인자로 칼럼의 위치를 지정하여 부분 키에 대한 고유 값의 개수를 나타낸다. 이 값은 추정치임에 유의한다.

갱신된 결과를 얻으려면 반드시 **UPDATE STATISTICS** 문을 먼저 수행해야 한다.

매개변수:

· **table_name** – 테이블 이름

· **index** – *table_name*으로 식별되는 테이블 내에 존재하는
인덱스 이름

· **key_pos** –

부분 키의 위치. *key_pos*는 0부터 시작하여 키를 구성하는
칼럼 개수보다 작은 범위를 갖는다. 즉, 첫 번째 칼럼의
*key_pos*는 0이다. 단일 칼럼 인덱스의 경우에는 0이다. 다
음 타입 중 하나가 될 수 있다.

- 숫자형 타입으로 변환할 수 있는 문자열.
- 정수형으로 변환할 수 있는 숫자형 타입. **FLOAT**나
DOUBLE 타입은 **ROUND** 함수로 변환한 값이 된다.

반환 형식:

INT

리턴 값은 0 또는 양의 정수이며, 입력 인자 중 하나라도 **NULL**이면 **NULL**을 반환한다. 입력 인자인 테
이블이나 인덱스가 발견되지 않거나 *key_pos*가 지정된 범위를 벗어나면 **NULL**을 리턴한다.

```
CREATE TABLE t1( i1 INTEGER ,
i2 INTEGER not null,
i3 INTEGER unique,
s1 VARCHAR(10),
s2 VARCHAR(10),
s3 VARCHAR(10) UNIQUE);

CREATE INDEX i_t1_i1 ON t1(i1 DESC);
CREATE INDEX i_t1_s1 ON t1(s1(7));
CREATE INDEX i_t1_i1_s1 on t1(i1,s1);
CREATE UNIQUE INDEX i_t1_i2_s2 ON t1(i2,s2);

INSERT INTO t1 VALUES (1,1,1,'abc','abc','abc');
INSERT INTO t1 VALUES (2,2,2,'zabc','zabc','zabc');
INSERT INTO t1 VALUES (2,3,3,'+abc','+abc','+abc');

UPDATE STATISTICS ON t1;
SELECT INDEX_CARDINALITY('t1','i_t1_i1_s1',0);
```

```
index_cardinality('t1', 'i_t1_i1_s1', 0)
=====
2
```

```
SELECT INDEX_CARDINALITY('t1','i_t1_i1_s1',1);
```

```
index_cardinality('t1', 'i_t1_i1_s1', 1)
=====
3
```

```
SELECT INDEX_CARDINALITY('t1','i_t1_i1_s1',2);
```

```
index_cardinality('t1', 'i_t1_i1_s1', 2)
=====
NULL
```

```
SELECT INDEX_CARDINALITY('t123','i_t1_i1_s1',1);
```

```
index_cardinality('t123', 'i_t1_i1_s1', 1)
=====
NULL
```

INET_ATON

INET_ATON(*ip_string*)

INET_ATON 함수는 IPv4 주소의 문자열을 입력받아 이에 해당하는 숫자를 반환한다. 'a.b.c.d' 형식의 IP 주소 문자열을 입력하면 "a * 256 ^ 3 + b * 256 ^ 2 + c * 256 + d"가 반환된다. 반환 타입은 **BIGINT** 이다.

매개변수:

ip_string – IPv4 주소 문자열

반환 형식:

BIGINT

다음 예제에서 192.168.0.10은 "192 * 256 ^ 3 + 168 * 256 ^ 2 + 0 * 256 + 10"으로 계산된다.

```
SELECT INET_ATON('192.168.0.10');
```

```
inet_aton('192.168.0.10')
=====
3232235530
```

INET_NTOA

INET_NTOA(*expr*)

INET_NTOA 함수는 숫자를 입력받아 IPv4 주소 형식의 문자열을 반환한다. 반환 타입은 **VARCHAR** 이다.

매개변수:

expr - 숫자 표현식

반환 형식:

STRING

```
SELECT INET_NTOA(3232235530);
```

```
inet_ntoa(3232235530)
=====
'192.168.0.10'
```

LAST_INSERT_ID

LAST_INSERT_ID()

LAST_INSERT_ID 함수는 **AUTO_INCREMENT** 칼럼의 값이 자동 증가할 때 가장 최근에 **INSERT** 된 값을 반환한다.

반환 형식:

BIGINT

LAST_INSERT_ID 함수가 반환하는 값은 다음의 특징을 가진다.

- **INSERT** 문 수행에 성공했던 가장 최근의 **LAST_INSERT_ID** 값이 유지된다. **INSERT** 문 수행에 실패하는 경우 **LAST_INSERT_ID()** 값은 변동이 없으나 **AUTO_INCREMENT** 값은 내부적으로 증가한다.

따라서, 다음 **INSERT** 문 수행이 성공한 이후 **LAST_INSERT_ID()** 값은 내부적으로 증가된 **AUTO_INCREMENT** 값을 반영한다.

```
CREATE TABLE tbl(a INT PRIMARY KEY AUTO_INCREMENT, b INT UNIQUE);
INSERT INTO tbl VALUES (null, 1);
INSERT INTO tbl VALUES (null, 1);
```

```
ERROR: Operation would have caused one or more unique constraint
violations.
```

```
INSERT INTO tbl VALUES (null, 1);
```

```
ERROR: Operation would have caused one or more unique constraint
violations.
```

```
SELECT LAST_INSERT_ID();
```

```
1
```

```
-- In 2008 R4.x, above value was 3.
```

```
INSERT INTO tbl VALUES (null, 2);
SELECT LAST_INSERT_ID();
```

```
4
```

- 다중 행 **INSERT** 문(**INSERT INTO tbl VALUES (), (), ..., ()**)에서 **LAST_INSERT_ID()**는 첫 번째로 입력된 **AUTO_INCREMENT** 값을 반환한다. 즉, 두 번째 행부터는 입력이 되어도 **LAST_INSERT_ID()** 값이 변하지 않는다.

```
INSERT INTO tbl VALUES (null, 11), (null, 12), (null, 13);
SELECT LAST_INSERT_ID();
```

```
5
```

```
INSERT INTO tbl VALUES (null, 21);
SELECT LAST_INSERT_ID();
```

8

- **INSERT** 문이 실행에 성공한 경우, **LAST_INSERT_ID ()** 값은 트랜잭션이 롤백되어도 예전의 **LAST_INSERT_ID ()** 값으로 복구되지 않는다.

```
-- csql> ;autocommit off
CREATE TABLE tbl2(a INT PRIMARY KEY AUTO_INCREMENT, b INT UNIQUE);
INSERT INTO tbl2 VALUES (null, 1);
COMMIT;

SELECT LAST_INSERT_ID();
```

1

```
INSERT INTO tbl2 VALUES (null, 2);
INSERT INTO tbl2 VALUES (null, 3);

ROLLBACK;

SELECT LAST_INSERT_ID();
```

3

- 트리거 내에서 사용한 **LAST_INSERT_ID()** 값은 트리거 밖에서 확인할 수 없다.
- **LAST_INSERT_ID**는 각 응용 클라이언트의 세션마다 독립적으로 유지된다.

```
CREATE TABLE ss (id INT AUTO_INCREMENT NOT NULL PRIMARY KEY, text
VARCHAR(32));
INSERT INTO ss VALUES (NULL, 'cubrid');
SELECT LAST_INSERT_ID ();
```

```
last_insert_id()
=====
1
```

```
INSERT INTO ss VALUES (NULL, 'database'), (NULL, 'manager');
SELECT LAST_INSERT_ID ();
```

```
last_insert_id()
=====
2
```

```
CREATE TABLE tbl (id INT AUTO_INCREMENT);
INSERT INTO tbl values (500), (NULL), (NULL);
SELECT LAST_INSERT_ID();
```

```
last_insert_id()
=====
1
```

```
INSERT INTO tbl VALUES (500), (NULL), (NULL);
SELECT LAST_INSERT_ID();
```

```
last_insert_id()
=====
3
```

```
SELECT * FROM tbl;
```

```
id
=====
500
1
2
500
3
4
```

LIST_DBS

LIST_DBS()

LIST_DBS 함수는 디렉터리 파일(**\$CUBRID_DATABASES/databases.txt**)에 존재하는 모든 데이터베이스 리스트를 공백 문자로 구분하여 출력한다.

반환 형식:

STRING

```
SELECT LIST_DBS();
```

```
list_dbs()
=====
'testdb demodb'
```

ROW_COUNT

ROW_COUNT()

ROW_COUNT 함수는 가장 마지막에 수행된 **UPDATE, INSERT, DELETE, REPLACE** 문에 영향을 받는 행의 개수를 정수로 반환한다.

INSERT ON DUPLICATE KEY UPDATE 문에 의해 INSERT된 각 행은 1을 UPDATE된 행은 각각 2를 반환한다. **REPLACE** 문에 대해서는 DELETE와 INSERT를 합한 개수를 반환한다.

트리거로 인해 수행되는 문장들은 해당 문장의 ROW_COUNT에 영향을 끼치지 않는다.

반환 형식:

INT

```
CREATE TABLE rc (i int);
INSERT INTO rc VALUES (1),(2),(3),(4),(5),(6),(7);
SELECT ROW_COUNT();
```

```
row_count()
=====
7
```

```
UPDATE rc SET i = 0 WHERE i > 3;
SELECT ROW_COUNT();
```



```
row_count()
=====
4
```

```
DELETE FROM rc WHERE i = 0;
SELECT ROW_COUNT();
```

```
row_count()
=====
4
```

SESSIONTIMEZONE

SESSIONTIMEZONE()

세션의 타임존(오프셋 또는 지명)을 문자열로 출력한다(예: '-05:00', 또는 'Europe/Vienna').

```
SELECT SESSIONTIMEZONE();
```

```
sessiontimezone
=====
'Asia/Seoul'
```

! 더 보기

[DBTIMEZONE\(\)](#) , [FROM_TZ\(\)](#) , [NEW_TIME\(\)](#) , [TZ_OFFSET\(\)](#)

USER, SYSTEM_USER

USER()

SYSTEM_USER()

USER 함수와 **SYSTEM_USER** 함수는 동일하며, 사용자 이름을 CSQL 또는 CAS가 실행된 호스트 이름과 함께 반환한다.

기능이 비슷한 [USER](#) 와 [CURRENT_USER](#) 의사 칼럼(pseudo column)은 현재 데이터베이스에 로그인한 사용자의 이름을 문자열로 반환한다.

반환 형식:

STRING

```
--selecting the current user on the session
SELECT SYSTEM_USER ();
```

```
user()
=====
'PUBLIC@cubrid_host'
```

```
SELECT USER(), CURRENT_USER;
```

```
user()                CURRENT_USER
=====
'PUBLIC@cubrid_host'  'PUBLIC'
```

```
--selecting all users of the current database from the system table
SELECT name, id, password FROM db_user;
```

name	id	password
'DBA'	NULL	NULL
'PUBLIC'	NULL	NULL
'SELECT_ONLY_USER'	NULL	db_password
'ALMOST_DBA_USER'	NULL	db_password
'SELECT_ONLY_USER2'	NULL	NULL

VERSION

VERSION()

CUBRID 서버 버전을 나타내는 버전 문자열을 반환한다.

반환 형식:

STRING

```
SELECT VERSION();
```

```
version()
=====
'9.1.0.0203'
```

암호화 함수

목차

암호화 함수	7
● MD5	703
● SHA1	704
● SHA2	705

MD5

MD5(string)

입력 문자열에 대해 MD5 128비트 체크섬(checksum) 결과를 반환한다. 결과 값은 32개의 16진수로 표현된 문자열로 나타나며, 이 값은 예를 들면 해시 키를 생성할 때 사용할 수도 있다.

매개변수:

string – 입력 문자열. **VARCHAR** 이 아닌 값이 입력되면 **VARCHAR** 으로 변환한다.

반환 형식:

STRING

리턴 값은 **VARCHAR** (32) 타입이며, 입력 인자가 **NULL** 이면 **NULL** 을 리턴한다.

```
SELECT MD5('cubrid');
```

```
md5('cubrid')
=====
'685c62385ce717a04f909047d0a55a16'
```

```
SELECT MD5(255);
```

```
md5(255)
=====
'fe131d7f5a6b38b23cc967316c13dae2'
```

```
SELECT MD5('01/01/2010');
```

```
md5('01/01/2010')
=====
'4a2f373c30426a1b8e9cf002ef0d4a58'
```

```
SELECT MD5(CAST('2010-01-01' as DATE));
```

```
md5( cast('2010-01-01' as date))
=====
'4a2f373c30426a1b8e9cf002ef0d4a58'
```

SHA1

SHA1(*string*)

SHA1 함수는 입력 문자열에 대해 160비트의 체크섬을 계산하는데, 이는 RFC 3174(보안 해시 알고리즘)에 기술되어 있다.

매개변수:

string - 암호화할 대상 문자열

반환 형식:

STRING

40개의 16진수 문자열을 반환하며, 입력 인자가 NULL이면 NULL을 반환한다.

```
SELECT SHA1('cubrid');
```

```
sha1('cubrid')
=====
'0562A8E9C814E660F5FFEB0DAC739ABFBBB1CB69 '
```

SHA2

SHA2(*string*, *hash_length*)

SHA2 함수는 SHA-2 계열의 해시 함수들(SHA-224, SHA-256, SHA-384, and SHA-512)을 계산한다. 첫번째 인자는 해싱될 문자열이다. 두번째 인자는 기대하는 결과 비트의 길이를 나타내는데, 224, 256, 384, 512 또는 0(256과 동일) 중 하나여야 한다.

매개변수:

string – 암호화할 대상 문자열

반환 형식:

STRING

인자 중 하나 이상이 NULL 이거나 허용된 해시 길이가 아니면 NULL을 반환한다. 정상 범위의 인자를 입력한 경우 원하는 비트 수를 포함하는 해시 값을 반환한다.

```
SELECT SHA2('cubrid', 256);
```

```
sha2('cubrid', 256)
=====
'D14DA17F2C492114F4A57D9F7BED908FD3A351B40CD59F0F79413687E4CA85A5 '
```

```
SELECT SHA2('cubrid', 224);
```

```
sha2('cubrid', 224)
=====
'8E5E18B5B47646C31CCEA98A87B19CBEF084036716FBD13D723AC9B2 '
```

비교 연산식

단순 비교 조건식

비교 조건식은 **SELECT**, **UPDATE**, **DELETE** 문의 **WHERE** 절과 **SELECT** 문의 **HAVING** 절에 포함되는 표현식으로서, 결합되는 연산자의 종류에 따라 단순 비교 조건식, **ANY** / **SOME** / **ALL** 조건식, **BETWEEN** 조건식, **EXISTS** 조건식, **IN** / **NOT IN** 조건식, **LIKE** 조건식, **IS NULL** 조건식이 있다.

먼저, 단순 비교 조건식은 두 개의 비교 가능한 데이터 값을 비교한다. 피연산자로 일반 연산식(expression) 또는 부질의(sub-query)가 지정되며, 피연산자 중 어느 하나가 **NULL** 이면 항상 **NULL** 을 반환한다. 단순 비교 조건식에서 사용할 수 있는 연산자는 아래의 표와 같으며, 보다 자세한 내용은 **비교 연산자** 를 참고한다.

단순 비교 조건식에서 사용할 수 있는 연산자

비교 연산자	설명	조건식	리턴 값
=	왼쪽 및 오른쪽 피연산자의 값이 같다.	1=2	0
<>, !=	왼쪽 및 오른쪽 피연산자의 값이 다르다.	1<>2	1
>	왼쪽 피연산자는 오른쪽 피연산자보다 값이 크다.	1>2	0
<	왼쪽 피연산자는 오른쪽 피연산자보다 값이 작다.	1<2	1
>=	왼쪽 피연산자는 오른쪽 피연산자보다 값이 크거나 같다.	1>=2	0
<=	왼쪽 피연산자는 오른쪽 피연산자보다 값이 작거나 같다.	1<=2	1

ANY/SOME/ALL 수량어와 그룹 조건식

ANY / SOME / ALL 과 같은 수량어를 포함하는 그룹 조건식은 하나의 데이터 값과 리스트에 포함된 값들의 일부 또는 모든 값에 대해서 비교 연산을 수행한다. 즉, **ANY** 또는 **SOME** 이 포함된 그룹 조건식은, 왼쪽의 데이터 값이 오른쪽 피연산자로 지정된 리스트 내의 값 중 최소한 하나에 대해 단순 비교 연산자를 만족할 때 **TRUE** 를 반환한다. 한편, **ALL** 이 포함된 그룹 조건식의 경우, 왼쪽 데이터 값이 오른쪽 리스트 내의 모든 값들에 대해 단순 비교 연산자를 만족할 때 **TRUE** 를 반환한다.

만약, **ANY** 또는 **SOME** 을 포함하는 그룹 조건식에서 **NULL** 을 대상으로 비교 연산을 수행하면 그룹 조건식의 결과로 **UNKNOWN** 또는 **TRUE** 를 반환하고, **ALL** 을 포함하는 그룹 조건식에서 **NULL** 을 대상으로 비교 연산을 수행하면 **UNKNOWN** 또는 **FALSE** 를 반환한다.

```
expression comp_op SOME expression
expression comp_op ANY expression
expression comp_op ALL expression
```

- *comp_op* : 비교 연산자 >, <, =, >=, <= 가 들어갈 수 있다.
- *expression* (왼쪽) : 단일 값을 가지는 칼럼, 경로 표현식(예: *tbl_name.col_name*), 상수 값 또는 단일 값을 생성하는 산술 함수가 될 수 있다.
- *expression* (오른쪽) : 칼럼 이름, 경로 표현식, 상수 값의 리스트(집합), 부질의가 될 수 있다. 리스트는 중괄호({}) 안에 표현된 집합을 의미하며, 부질의가 사용되면 부질의의 수행 결과 전체에 대해서 *expression* (왼쪽)와 비교 연산을 수행한다.

```
--creating a table

CREATE TABLE condition_tbl (id int primary key, name char(10),
dept_name VARCHAR, salary INT);
INSERT INTO condition_tbl VALUES(1, 'Kim', 'devel', 4000000);
INSERT INTO condition_tbl VALUES(2, 'Moy', 'sales', 3000000);
INSERT INTO condition_tbl VALUES(3, 'Jones', 'sales', 5400000);
INSERT INTO condition_tbl VALUES(4, 'Smith', 'devel', 5500000);
INSERT INTO condition_tbl VALUES(5, 'Kim', 'account', 3800000);
INSERT INTO condition_tbl VALUES(6, 'Smith', 'devel', 2400000);
INSERT INTO condition_tbl VALUES(7, 'Brown', 'account', NULL);

--selecting rows where department is sales or devel
SELECT * FROM condition_tbl WHERE dept_name = ANY{'devel','sales'};
```

```

      id  name      dept_name
salary
=====
=
      1  'Kim      '      'devel'
```

```

4000000
      2  'Moy      '      'sales'
3000000
      3  'Jones    '      'sales'
5400000
      4  'Smith    '      'devel'
5500000
      6  'Smith    '      'devel'
2400000

```

```

--selecting rows comparing NULL value in the ALL group conditions
SELECT * FROM condition_tbl WHERE salary > ALL{3000000, 4000000,
NULL};

```

There are no results.

```

--selecting rows comparing NULL value in the ANY group conditions
SELECT * FROM condition_tbl WHERE salary > ANY{3000000, 4000000,
NULL};

```

```

      id  name      dept_name
salary
=====
=
      1  'Kim      '      'devel'
4000000
      3  'Jones    '      'sales'
5400000
      4  'Smith    '      'devel'
5500000
      5  'Kim      '      'account'
3800000

```

```

--selecting rows where salary*0.9 is less than those salary in devel
department
SELECT * FROM condition_tbl WHERE (
  (0.9 * salary) < ALL (SELECT salary FROM condition_tbl
    WHERE dept_name = 'devel')
);

```

```

      id  name      dept_name
salary
=====

```



```
=
        6  'Smith'      'devel'
2400000
```

BETWEEN

BETWEEN 조건식은 왼쪽의 데이터 값이 오른쪽에 지정된 두 데이터 값 사이에 존재하는지 비교한다. 이때, 왼쪽의 데이터 값이 비교 대상 범위의 경계값과 동일한 경우에도 **TRUE** 를 반환한다. 한편, **BETWEEN** 키워드 앞에 **NOT** 이 오면 **BETWEEN** 연산의 결과에 **NOT** 연산을 수행하여 결과를 반환한다.

i **BETWEEN** g **AND** m 은 복합 조건식 $i \geq g$ **AND** $i \leq m$ 과 동일하다.

```
expression [ NOT ] BETWEEN expression AND expression
```

- *expression* : 칼럼 이름, 경로 표현식(예: *tbl_name.col_name*), 상수 값, 산술 표현식, 집계 함수가 될 수 있다. 문자열 표현식인 경우에는 문자의 사전순으로 조건이 평가된다. 표현식 중 하나라도 **NULL** 이 지정되면 **BETWEEN** 조건식의 결과는 **FALSE** 또는 **UNKNOWN** 을 반환한다.

```
--selecting rows where 3000000 <= salary <= 4000000
SELECT * FROM condition_tbl WHERE salary BETWEEN 3000000 AND 4000000;
SELECT * FROM condition_tbl WHERE (salary >= 3000000) AND (salary <=
4000000);
```

salary	id	name	dept_name
=====			
=			
	1	'Kim'	'devel'
4000000	2	'Moy'	'sales'
3000000	5	'Kim'	'account'
3800000			

```
--selecting rows where salary < 3000000 or salary > 4000000
SELECT * FROM condition_tbl WHERE salary NOT BETWEEN 3000000 AND
4000000;
```

salary	id	name	dept_name
=====			

```
=
      3  'Jones      '      'sales'
5400000
      4  'Smith      '      'devel'
5500000
      6  'Smith      '      'devel'
2400000
```

```
--selecting rows where name starts from A to E
SELECT * FROM condition_tbl WHERE name BETWEEN 'A' AND 'E';
```

```

      id  name      dept_name
salary
=====
=
      7  'Brown      '      'account'
NULL
```

EXISTS

EXISTS 조건식은 오른쪽에 지정되는 부질의 실행한 결과가 하나 이상 존재하면 **TRUE** 를 반환하고, 연산 실행 결과가 공집합이면 **FALSE** 를 반환한다.

EXISTS expression

- *expression* : 부질의가 지정되며, 부질의 실행 결과가 존재하는지 비교한다. 만약 부질의가 어떤 결과도 만들지 않는다면 조건식 결과는 **FALSE** 이다.

```
--selecting rows using EXISTS and subquery
SELECT 'raise' FROM db_root WHERE EXISTS(
SELECT * FROM condition_tbl WHERE salary < 2500000);
```

```

'raise'
=====
'raise'
```

```
--selecting rows using NOT EXISTS and subquery
SELECT 'raise' FROM db_root WHERE NOT EXISTS(
SELECT * FROM condition_tbl WHERE salary < 2500000);
```

```
There are no results.
```

IN

IN 조건식은 왼쪽의 단일 데이터 값이 오른쪽에 지정된 리스트 내에 포함되어 있는지 비교한다. 즉, 왼쪽의 단일 데이터 값이 오른쪽에 지정된 표현식의 원소이면 **TRUE** 를 반환한다. **IN** 키워드 앞에 **NOT** 이 있으면 **IN** 연산의 결과에 **NOT** 연산을 수행하여 결과를 반환한다.

```
expression [ NOT ] IN expression
```

- *expression* (left) : 단일 값을 가지는 칼럼, 경로 표현식, 상수 값 또는 단일 값을 생성하는 산술 함수가 될 수 있다.
- *expression* (right) : 칼럼 이름, 경로 표현식(예: *tbl_name.col_name*), 상수 값의 리스트(집합), 부질의가 될 수 있다. 리스트는 소괄호(()) 또는 중괄호({}) 안에 표현된 집합을 의미하며, 부질의가 사용되면 부질의의 수행 결과 전체에 대해서 *expression* (left)와 비교 연산을 수행한다.

```
--selecting rows where department is sales or devel
SELECT * FROM condition_tbl WHERE dept_name IN {'devel','sales'};
SELECT * FROM condition_tbl WHERE dept_name = ANY{'devel','sales'};
```

salary	id	name	dept_name
=====			
=			
	1	'Kim'	'devel'
4000000	2	'Moy'	'sales'
3000000	3	'Jones'	'sales'
5400000	4	'Smith'	'devel'
5500000	6	'Smith'	'devel'
2400000			

```
--selecting rows where department is neither sales nor devel
SELECT * FROM condition_tbl WHERE dept_name NOT IN {'devel','sales'};
```

salary	id	name	dept_name

```
=====
=
          5  'Kim      '          'account'
3800000
          7  'Brown   '          'account'
NULL
```

IS NULL

IS NULL 조건식은 왼쪽에 지정된 표현식의 결과가 **NULL** 인지 비교하여, **NULL** 인 경우 **TRUE** 를 반환하며, 조건절 내에서 사용할 수 있다. **NULL** 키워드 앞에 **NOT** 이 있으면 **IS NULL** 연산의 결과에 **NOT** 연산을 수행하여 결과를 반환한다.

```
expression IS [ NOT ] NULL
```

- *expression* : 단일 값을 가지는 칼럼, 경로 표현식(예: *tbl_name.col_name*), 상수 값 또는 단일 값을 생성하는 산술 함수가 될 수 있다.

```
--selecting rows where salary is NULL
SELECT * FROM condition_tbl WHERE salary IS NULL;
```

```

          id  name          dept_name
salary
=====
=
          7  'Brown   '          'account'
NULL
```

```
--selecting rows where salary is NOT NULL
SELECT * FROM condition_tbl WHERE salary IS NOT NULL;
```

```

          id  name          dept_name
salary
=====
=
          1  'Kim      '          'devel'
4000000
          2  'Moy      '          'sales'
3000000
          3  'Jones    '          'sales'
5400000
          4  'Smith    '          'devel'
5500000
```

```

3800000      5  'Kim      '      'account'
2400000      6  'Smith    '      'devel'

```

```

--simple comparison operation returns NULL when operand is NULL
SELECT * FROM condition_tbl WHERE salary = NULL;

```

There are no results.

LIKE

LIKE 조건식은 문자열 데이터 간의 패턴을 비교하는 연산을 수행하여, 검색어와 일치하는 패턴의 문자열이 검색되면 **TRUE** 를 반환한다. 패턴 비교 대상이 되는 타입은 **CHAR**, **VARCHAR**, **STRING** 이며, **BIT** 타입에 대해서는 **LIKE** 검색을 수행할 수 없다. **LIKE** 키워드 앞에 **NOT** 이 있으면 **LIKE** 연산의 결과에 **NOT** 연산을 수행하여 결과를 반환한다.

LIKE 연산자 오른쪽에 오는 검색어에는 임의의 문자 또는 문자열에 대응되는 와일드 카드(wild card) 문자열을 포함할 수 있으며, **%** (percent)와 **_** (underscore)를 사용할 수 있다. **%** 는 길이가 0 이상인 임의의 문자열에 대응되며, **_** 는 1개의 문자에 대응된다. 또한, 이스케이프 문자(escape character)는 와일드 카드 문자 자체에 대한 검색을 수행할 때 사용되는 문자로서, 사용자에 의해 길이가 1인 다른 문자(**NULL**, 알파벳 또는 숫자)로 지정될 수 있다. 와일드 카드 문자 또는 이스케이프 문자를 포함하는 문자열을 검색어로 사용하는 예제는 아래를 참고한다.

```
expression [ NOT ] LIKE pattern [ ESCAPE char ]
```

- *expression*: 문자열 데이터 타입 칼럼이 지정된다. 패턴 비교는 칼럼 값의 첫 번째 문자부터 시작되며, 대소문자를 구분한다.
- *pattern*: 검색어를 입력하며, 길이가 0 이상인 문자열이 된다. 이때, 검색어 패턴에는 와일드 카드 문자(**%** 또는 **_**)가 포함될 수 있다. 문자열의 길이는 0 이상이다.
- **ESCAPE** *char* : *char* 에 올 수 있는 문자는 **NULL**, 알파벳, 숫자이다. 만약 검색어의 문자열 패턴이 “_” 또는 “%” 자체를 포함하는 경우 이스케이프 문자가 반드시 지정되어야 한다. 예를 들어, 이스케이프 문자를 백슬래시(\)로 지정한 후 ‘10%’인 문자열을 검색하고자 한다면, *pattern*에 ‘10%’을 지정해야 한다. 또한, ‘C:\’인 문자열을 검색하고자 한다면, *pattern*에 ‘C:\’을 지정하면 된다.

CUBRID가 지원하는 문자셋에 관한 상세한 설명은 [문자열 데이터 타입](#) 을 참고한다.

LIKE 조건식의 이스케이프 문자 인식은 **cubrid.conf** 파일의 **no_backslash_escapes** 파라미터와 **require_like_escape_character** 파라미터의 설정에 따라 달라진다. 이에 대한 상세한 설명은 [구문/타입 관련 파라미터](#) 를 참고한다.

참고

- CUBRID 9.0 미만 버전에서는 UTF-8과 같은 멀티바이트 문자셋 환경에서 입력된 데이터에 대해 문자열 비교 연산을 수행하려면, 1바이트 단위로 문자열 비교를 수행하도록 하는 파라미터 (**single_byte_compare** = yes)를 **cubrid.conf** 파일에 추가해야 정상적인 검색 결과를 얻을 수 있다.
- CUBRID 9.0 이상 버전에서는 유니코드 문자셋을 지원하므로 **single_byte_compare** 파라미터를 더 이상 사용하지 않는다.

```
--selection rows where name contains lower case 's', not upper case
SELECT * FROM condition_tbl WHERE name LIKE '%s%';
```

salary	id	name	dept_name
=====			
=			
5400000	3	'Jones'	'sales'

```
--selection rows where second letter is 'O' or 'o'
SELECT * FROM condition_tbl WHERE UPPER(name) LIKE '_O%';
```

salary	id	name	dept_name
=====			
=			
3000000	2	'Moy'	'sales'
5400000	3	'Jones'	'sales'

```
--selection rows where name is 3 characters
SELECT * FROM condition_tbl WHERE name LIKE '___';
```

salary	id	name	dept_name
=====			
=			

```

4000000      1  'Kim      '      'devel'
3000000      2  'Moy      '      'sales'
3800000      5  'Kim      '      'account'

```

CASE

CASE 연산식은 **IF ... THEN ... ELSE** 로직을 SQL 문장으로 표현하며, **WHEN** 에 지정된 비교 연산 결과가 참이면 **THEN** 절의 값을 반환하고 거짓이면 **ELSE** 절에 명시된 값을 반환한다. 만약, **ELSE** 절이 없다면 **NULL** 값을 반환한다.

```

CASE control_expression simple_when_list
[ else_clause ]
END

CASE searched_when_list
[ else_clause ]
END

simple_when :
WHEN expression THEN result

searched_when :
WHEN search_condition THEN result

else_clause :
ELSE result

result :
expression | NULL

```

CASE 조건식은 반드시 키워드 **END** 로 끝나야 하며, *control_expression* 과 데이터 타입과 *simple_when* 절 내의 *expression* 은 비교 가능한 데이터 타입이어야 한다. 또한, **THEN** 과 **ELSE** 절에 지정된 모든 *result* 의 데이터 타입은 서로 같거나, 어느 하나의 공통 데이터 타입으로 변환 가능 (convertible)해야 한다.

CASE 수식이 반환하는 값의 데이터 타입은 다음과 같은 규칙에 따라 결정된다.

- **THEN** 절에 명시된 모든 *result* 의 데이터 타입이 같으면, 해당 타입이 리턴 값의 데이터 타입이 된다.
- 모든 *result* 의 데이터 타입이 같지 않더라도 어느 하나의 공통 데이터 타입으로 변환 가능하면, 해당 타입이 리턴 값의 데이터 타입이 된다.

- *result* 중 어느 하나가 가변 길이 문자열인 경우, 리턴 값의 데이터 타입은 가변 길이 문자열이 된다. 또한, *result* 가 모두 고정 길이 문자열인 경우에는 가장 긴 길이를 가지는 문자열 또는 비트열이 결과로 반환된다.
- *result* 중 어느 하나가 근사치로 표현되는 수치형이면, 근사치로 표현되고 이때 소수점 이하 자릿수는 모든 *result* 의 유효 숫자를 표현할 수 있도록 결정된다.

```
--creating a table
CREATE TABLE case_tbl( a INT);
INSERT INTO case_tbl VALUES (1);
INSERT INTO case_tbl VALUES (2);
INSERT INTO case_tbl VALUES (3);
INSERT INTO case_tbl VALUES (NULL);

--case operation with a search when clause
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 'other'
       END
FROM case_tbl;
```

```
      a  case when a=1 then 'one' when a=2 then 'two' else
'other' end
=====
      1  'one'
      2  'two'
      3  'other'
     NULL 'other'
```

```
--case operation with a simple when clause
SELECT a,
       CASE a WHEN 1 THEN 'one'
            WHEN 2 THEN 'two'
            ELSE 'other'
       END
FROM case_tbl;
```

```
      a  case a when 1 then 'one' when 2 then 'two' else
'other' end
=====
      1  'one'
      2  'two'
      3  'other'
     NULL 'other'
```



```
--result types are converted to a single type containing all of
significant figures
```

```
SELECT a,
       CASE WHEN a=1 THEN 1
            WHEN a=2 THEN 1.2345
            ELSE 1.234567890
       END
FROM case_tbl;
```

```
      a case when a=1 then 1 when a=2 then 1.2345 else
1.234567890 end
=====
      1  1.0000000000
      2  1.2345000000
      3  1.2345678900
  NULL  1.2345678900
```

```
--an error occurs when result types are not convertible
```

```
SELECT a,
       CASE WHEN a=1 THEN 'one'
            WHEN a=2 THEN 'two'
            ELSE 1.2345
       END
FROM case_tbl;
```

```
ERROR: Cannot coerce 'one' to type double.
```

비교 함수

목차

비교 함수

7

- COALESCE 718
- DECODE 719
- GREATEST 721
- IF 722

● IFNULL, NVL	723
● ISNULL	725
● LEAST	726
● NULLIF	726
● NVL2	728

COALESCE

COALESCE(*expression* [, *expression*] ...)

COALESCE 함수는 하나 이상의 연산식 리스트가 인자로 지정되며, 첫 번째 인자가 **NULL** 이 아닌 값이면 해당 값을 결과로 반환하고, **NULL** 이면 두 번째 인자를 반환한다. 만약 인자로 지정된 모든 연산식이 **NULL** 이면 **NULL** 을 결과로 반환한다. 이러한 **COALESCE** 함수는 주로 **NULL** 값을 다른 기본값으로 대체할 때 사용한다.

매개변수:

expression – 하나 이상의 연산식을 지정하며, 서로 비교 가능한 타입이어야 한다.

반환 형식:

*expression*의 타입

COALESCE 함수는 인자의 타입 중 우선순위가 가장 높은 타입으로 모든 인자를 변환하여 연산을 수행한다. 인자 중에 같은 타입으로 변환할 수 없는 타입의 인자가 있으면 모든 인자를 **VARCHAR** 타입으로 변환한다. 아래는 입력 인자의 타입에 따른 변환 우선순위를 나타낸 것이다.

- **CHAR < VARCHAR**
- **BIT < VARBIT**
- **SHORT < INT < BIGINT < NUMERIC < FLOAT < DOUBLE**
- **DATE < TIMESTAMP < DATETIME**

예를 들어 a의 타입이 **INT**, b의 타입이 **BIGINT**, c의 타입이 **SHORT**, d의 타입이 **FLOAT** 이면 **COALESCE** (a, b, c, d)는 **FLOAT** 타입을 반환한다. 만약 a의 타입이 **INTEGER**, b의 타입이 **DOUBLE**, c의 타입이 **FLOAT**, d의 타입이 **TIMESTAMP** 이면 **COALESCE** (a, b, c, d)는 **VARCHAR** 타입을 반환한다.

COALESCE (*a*, *b*)는 다음의 **CASE** 조건식과 같은 의미를 가진다.

```
CASE WHEN a IS NOT NULL
THEN a
ELSE b
END
```

```
SELECT * FROM case_tbl;
```

```

      a
=====
      1
      2
      3
     NULL
```

```
--substituting a default value 10.0000 for a NULL value
SELECT a, COALESCE(a, 10.0000) FROM case_tbl;
```

```

      a  coalesce(a, 10.0000)
=====
      1   1.0000
      2   2.0000
      3   3.0000
     NULL 10.0000
```

DECODE

DECODE(*expression*, *search*, *result* [, *search*, *result*]* [, *default*])

DECODE 함수는 **CASE** 연산식과 마찬가지로 **IF ... THEN ... ELSE** 문과 동일한 기능을 수행한다. 인자로 지정된 *expression* 과 *search* 를 비교하여, 같은 값을 가지는 *search* 에 대응하는 *result* 를 결과로 반환한다. 만약, 같은 값을 가지는 *search* 가 없다면 *default* 값을 반환하고, *default* 값이 생략된 경우에는 **NULL** 을 반환한다. 비교 연산의 대상이 되는 *expression* 과 *search* 는 데이터 타입이 동일하거나 서로 변환 가능해야 하고, 지정된 모든 *result* 값의 유효 숫자를 포함하여 표현할 수 있도록 결과 값의 소수점 아래 자릿수가 결정된다.

매개변수:

- **expression, search** – 비교 가능한 타입의 연산식
- **result** – 매칭되었을 때 반환할 값

- **default** – 매치가 발견되지 않았을 때 반환할 값

반환 형식:

*result*와 *default*의 타입에 따라 결정됨

DECODE(*a*, *b*, *c*, *d*, *e*, *f*)는 다음의 **CASE** 조건식과 같은 의미를 가진다.

```
CASE WHEN a = b THEN c
      WHEN a = d THEN e
      ELSE f
      END
```

```
SELECT * FROM case_tbl;
```

```

      a
=====
      1
      2
      3
     NULL
```

```
--Using DECODE function to compare expression and search values one
by one
SELECT a, DECODE(a, 1, 'one', 2, 'two', 'other') FROM case_tbl;
```

```

      a  decode(a, 1, 'one', 2, 'two', 'other')
=====
      1  'one'
      2  'two'
      3  'other'
     NULL 'other'
```

```
--result types are converted to a single type containing all of
significant figures
SELECT a, DECODE(a, 1, 1, 2, 1.2345, 1.234567890) FROM case_tbl;
```

```

      a  decode(a, 1, 1, 2, 1.2345, 1.234567890)
=====
      1  1.000000000
      2  1.234500000
```

```
3 1.234567890
NULL 1.234567890
```

```
--an error occurs when result types are not convertible
SELECT a, DECODE(a, 1, 'one', 2, 'two', 1.2345) FROM case_tbl;
```

```
ERROR: Cannot coerce 'one' to type double.
```

GREATEST

GREATEST(*expression* [, *expression*] ...)

GREATEST 함수는 인자로 지정된 하나 이상의 연산식을 서로 비교하여 가장 큰 값을 반환한다. 만약, 하나의 연산식만 지정되면 서로 비교할 대상이 없으므로 해당 연산식의 값을 그대로 반환한다.

따라서, 인자로 지정되는 하나 이상의 연산식은 서로 비교 가능한 타입이어야 한다. 지정된 인자의 타입이 동일하면 리턴 값의 타입도 동일하고, 인자의 타입이 다르면 리턴 값의 타입은 변환 가능(convertible)한 공통의 데이터 타입이 된다.

즉, **GREATEST** 함수는 같은 행(row) 내에서 칼럼 1, 칼럼 2, 칼럼 3의 값을 서로 비교하여 최대 값을 반환하며, **MAX()** 함수는 모든 결과 행들의 칼럼 1 값을 서로 비교하여 최대 값을 반환한다.

매개변수:

expression – 하나 이상의 연산식을 지정하며, 서로 비교 가능한 타입이어야 한다. 인자 중 어느 하나가 **NULL** 값이면 **NULL** 을 반환한다.

반환 형식:

*expression*의 타입

다음은 *demodb* 에서 한국이 획득한 각 메달의 수와 최대 메달의 수를 반환하는 예제이다.

```
SELECT gold, silver , bronze, GREATEST (gold, silver, bronze)
FROM participant
WHERE nation_code = 'KOR';
```

	gold	silver	bronze	greatest(gold, silver, bronze)
=====				
==				
	9	12	9	
12	8	10	10	
10	7	15	5	
15	12	5	12	
12	12	10	11	
12				

IF

IF(*expression1*, *expression2*, *expression3*)

IF 함수는 첫 번째 인자로 지정된 연산식의 값이 **TRUE** 이면 *expression2* 를 반환하고, **FALSE** 이거나 **NULL** 이면 *expression3* 를 반환한다. 결과로 반환되는 *expression2* 와 *expression3* 은 데이터 타입이 동일하거나 공통의 타입으로 변환 가능해야 한다. 둘 중 하나가 명확하게 **NULL** 이면, 함수의 결과 타입은 **NULL** 이 아닌 인자의 타입을 따른다.

매개변수:

- **expression1** – 비교 조건식
- **expression2** – *expression1*이 참일 때 반환할 값
- **expression3** – *expression1*이 참이 아닐 때 반환할 값

반환 형식:

expression2 또는 *expression3*의 타입

IF(*a*, *b*, *c*)는 다음의 **CASE** 연산식과 같은 의미를 가진다.

```
CASE WHEN a IS TRUE THEN b
ELSE c
END
```

```
SELECT * FROM case_tbl;
```

```

a
=====
1
2
3
NULL

```

```

--IF function returns the second expression when the first is TRUE
SELECT a, IF(a=1, 'one', 'other') FROM case_tbl;

```

```

a    if(a=1, 'one', 'other')
=====
1    'one'
2    'other'
3    'other'
NULL 'other'

```

```

--If function in WHERE clause
SELECT * FROM case_tbl WHERE IF(a=1, 1, 2) = 1;

```

```

a
=====
1

```

IFNULL, NVL

IFNULL(*expr1*, *expr2*)

NVL(*expr1*, *expr2*)

IFNULL 함수와 **NVL** 함수는 유사하게 동작하며, **NVL** 함수는 컬렉션 타입을 추가로 지원한다. 두 개의 인자가 지정되며, 첫 번째 인자 *expr1* 이 **NULL** 이 아니면 *expr1* 을 반환하고, **NULL** 이면 두 번째 인자인 *expr2* 를 반환한다.

매개변수:

- **expr1** – 조건식
- **expr2** – *expr1*이 **NULL**일 때 반환할 값

반환 형식:

*expr1*과 *expr2*의 타입에 따라 결정됨

IFNULL 함수와 **NVL** 함수는 인자의 타입 중 우선순위가 가장 높은 타입으로 모든 인자를 변환하여 연산을 수행한다. 인자 중에 같은 타입으로 변환할 수 없는 타입의 인자가 있으면 모든 인자를 **VARCHAR** 타입으로 변환한다. 아래는 입력 인자의 타입에 따른 변환 우선순위를 나타낸 것이다.

- **CHAR < VARCHAR**
- **BIT < VARBIT**
- **SHORT < INT < BIGINT < NUMERIC < FLOAT < DOUBLE**
- **DATE < TIMESTAMP < DATETIME**

예를 들어 a의 타입이 **INT**, b의 타입이 **BIGINT** 이면 **IFNULL** (a, b)은 **BIGINT** 타입을 반환한다. 만약 a의 타입이 **INTEGER**, b의 타입이 **TIMESTAMP** 이면 **IFNULL** (a, b)은 **VARCHAR** 타입을 반환한다.

IFNULL(a, b) 또는 **NVL**(a, b)는 다음의 **CASE** 조건식과 같은 의미를 가진다.

```
CASE WHEN a IS NULL THEN b
ELSE a
END
```

```
SELECT * FROM case_tbl;
```

```

      a
=====
      1
      2
      3
    NULL
```

```
--returning a specific value when a is NULL
SELECT a, NVL(a, 10.0000) FROM case_tbl;
```

```

      a  nvl(a, 10.0000)
=====
      1    1.0000
      2    2.0000
      3    3.0000
    NULL   10.0000
```



```
--IFNULL can be used instead of NVL and return values are converted
to the string type
SELECT a, IFNULL(a, 'UNKNOWN') FROM case_tbl;
```

```

      a  ifnull(a, 'UNKNOWN')
=====
      1  '1'
      2  '2'
      3  '3'
  NULL  'UNKNOWN'
```

ISNULL

ISNULL(expression)

ISNULL 함수는 조건절 내에서 사용할 수 있으며, 인자로 지정된 표현식의 결과가 **NULL** 인지 비교하여 **NULL** 이면 1을 반환하고, 아니면 0을 반환한다. 이 함수를 이용하여 어떤 값이 **NULL** 인지 아닌지를 테스트할 수 있으며, **IS NULL** 조건식과 유사하게 동작한다.

매개변수:

expression - 단일 값을 가지는 칼럼, 경로 표현식(예: *tbl_name.col_name*), 상수 값 또는 단일 값을 생성하는 산술 함수를 입력한다.

반환 형식:

INT

```
--Using ISNULL function to select rows with NULL value
SELECT * FROM condition_tbl WHERE ISNULL(salary);
```

```

      id  name  dept_name
salary
=====
=
      7  'Brown'  'account'
  NULL
```

LEAST

LEAST(*expression* [, *expression*] ...)

LEAST 함수는 인자로 지정된 하나 이상의 연산식을 비교하여 가장 작은 값을 반환한다. 만약, 하나의 연산식만 지정되면 서로 비교할 대상이 없으므로 해당 연산식의 값을 그대로 반환한다.

따라서, 인자로 지정되는 하나 이상의 연산식은 서로 비교 가능한 타입이어야 한다. 만약, 지정된 인자의 타입이 동일하면 리턴 값의 타입도 동일하고, 인자의 타입이 다르면 리턴 값의 타입은 변환 가능(convertible)한 공통의 데이터 타입이 된다.

즉, **LEAST** 함수는 같은 행(row) 내에서 칼럼 1, 칼럼 2, 칼럼 3의 값을 서로 비교하여 최소 값을 반환하며, **MIN()** 함수는 모든 결과 행들의 칼럼 1 값을 서로 비교하여 최소 값을 반환한다.

매개변수:

expression – 하나 이상의 연산식을 지정하며, 서로 비교 가능한 타입이어야 한다. 인자 중 어느 하나가 **NULL** 값이면 **NULL** 을 반환한다.

반환 형식:

*expression*의 타입

다음은 *demodb* 에서 한국이 획득한 각 메달의 수와 최소 메달의 수를 반환하는 예제이다.

```
SELECT gold, silver , bronze, LEAST(gold, silver, bronze) FROM
participant
WHERE nation_code = 'KOR';
```

gold	silver	bronze	least(gold, silver, bronze)
9	12	9	9
8	10	10	8
7	15	5	5
12	5	12	5
12	10	11	10

NULLIF

NULLIF(*expr1*, *expr2*)

NULLIF 함수는 인자로 지정된 두 개의 연산식이 동일하면 **NULL** 을 반환하고, 다르면 첫 번째 인자 값을 반환한다.

매개변수:

- **expr1** – 비교할 연산식
- **expr2** – 비교할 연산식

반환 형식:

*expr1*의 타입

NULLIF (*a*, *b*)는 다음의 **CASE** 조건식과 같은 의미를 가진다.

```
CASE
WHEN a = b THEN NULL
ELSE a
END
```

```
SELECT * FROM case_tbl;
```

```
      a
=====
      1
      2
      3
     NULL
```

```
--returning NULL value when a is 1
SELECT a, NULLIF(a, 1) FROM case_tbl;
```

```
      a  nullif(a, 1)
=====
      1      NULL
      2      2
      3      3
     NULL      NULL
```

```
--returning NULL value when arguments are same
SELECT NULLIF (1, 1.000) FROM db_root;
```

```

nullif(1, 1.000)
=====
NULL

```

```

--returning the first value when arguments are not same
SELECT NULLIF ('A', 'a') FROM db_root;

```

```

nullif('A', 'a')
=====
'A'

```

NVL2

NVL2(*expr1*, *expr2*, *expr3*)

NVL2 함수는 세 개의 인자가 지정되며, 첫 번째 연산식(*expr1*)이 **NULL** 이 아니면 두 번째 연산식(*expr2*)을 반환하고, **NULL** 이면 세 번째 연산식(*expr3*)을 반환한다.

매개변수:

- **expr1** – 조건식
- **expr2** – *expr1*이 **NULL**이 아닐 때 반환할 값
- **expr3** – *expr1*이 **NULL**일 때 반환할 값

반환 형식:

expr1, *expr2*, *expr3*의 타입에 따라서 결정됨

NVL2 함수는 인자의 타입 중 우선순위가 가장 높은 타입으로 모든 인자를 변환하여 연산을 수행한다. 인자 중에 같은 타입으로 변환할 수 없는 타입의 인자가 있으면 모든 인자를 **VARCHAR** 타입으로 변환한다. 아래는 입력 인자의 타입에 따른 변환 우선순위를 나타낸 것이다.

- **CHAR < VARCHAR**
- **BIT < VARBIT**
- **SHORT < INT < BIGINT < NUMERIC < FLOAT < DOUBLE**
- **DATE < TIMESTAMP < DATETIME**

예를 들어 a의 타입이 **INT**, b의 타입이 **BIGINT**, c의 타입이 **SHORT** 이면 **NVL2** (a, b, c)는 **BIGINT** 타입을 반환한다. 만약 a의 타입이 **INTEGER**, b의 타입이 **DOUBLE**, c의 타입이 **TIMESTAMP** 이면 **NVL2** (a, b, c)는 **VARCHAR** 타입을 반환한다.

```
SELECT * FROM case_tbl;
```

```

      a
=====
      1
      2
      3
    NULL

```

```
--returning a specific value of INT type
SELECT a, NVL2(a, a+1, 10.5678) FROM case_tbl;
```

```

      a  nvL2(a, a+1, 10.5678)
=====
      1                2
      2                3
      3                4
    NULL              11

```

기타 함수

목차

기타 함수

7

- SLEEP 729
- SYS_GUID 730

SLEEP

SLEEP(*sec*)

명시한 시간동안 멈춰있다가 수행한다.

매개변수:

sec – sleep 시간. 단위는 초이며, double 타입 값을 입력한다.

반환 형식:

INT

```
SELECT SLEEP(3);
```

3초간 멈춰있다가 수행한다.

SYS_GUID

SYS_GUID()

무작위로 고유한 32개 문자의 hexa 문자열(hexadecimal)을 반환한다.

```
SELECT SYS_GUID();
```

```
sys_guid()
=====
'938210043A7B4455927A8697FB2571FF'
```

데이터 조작성

SELECT

SELECT 문은 지정된 테이블에서 원하는 칼럼을 조회한다.

```
SELECT [ <qualifier> ] <select_expressions>
    [{TO | INTO} <variable_comma_list>]
    [FROM <extended_table_specification_comma_list>]
    [WHERE <search_condition>]
    [GROUP BY {col_name | expr} [ASC | DESC], ...[WITH ROLLUP]]
    [HAVING <search_condition> ]
    [USING INDEX { index_name [,index_name, ...] | NONE }]
    [ORDER BY {col_name | expr} [ASC | DESC], ... [NULLS {FIRST |
LAST}]]
```

```

[LIMIT [offset,] row_count]
[FOR UPDATE [OF <spec_name_comma_list>]]

<qualifier> ::= ALL | DISTINCT | DISTINCTROW | UNIQUE

<select_expressions> ::= * | <expression_comma_list> | *,
<expression_comma_list>

<variable_comma_list> ::= [:] identifier, [:] identifier, ...

<extended_table_specification_comma_list> ::=
    <table_specification> [
        {, <table_specification> } ... |
        <join_table_specification> ... |
        <join_table_specification2> ...
    ]

<table_specification> ::=
    <single_table_spec> [<correlation>] |
    <metaclass_specification> [ <correlation> ] |
    <subquery> <correlation> |
    TABLE ( <expression> ) <correlation>

<correlation> ::= [AS] <identifier> [( <identifier_comma_list> )]

<single_table_spec> ::= [ONLY] [schema_name.]table_name |
                        ALL [schema_name.]table_name [ EXCEPT
[schema_name.]table_name ]

<metaclass_specification> ::= CLASS [schema_name.]class_name

<join_table_specification> ::=
{
    [INNER | {LEFT | RIGHT} [OUTER]] JOIN

    } <table_specification> ON <search_condition>

<join_table_specification2> ::=
{
    CROSS JOIN |
    NATURAL [ LEFT | RIGHT ] JOIN
    } <table_specification>

```

- *qualifier*: 한정어. 생략이 가능하며 지정하지 않을 경우에는 **ALL** 로 지정된다.
 - **ALL**: 테이블의 모든 레코드를 조회한다.
 - **DISTINCT**: 중복을 허용하지 않고 유일한 값을 갖는 레코드에 대해서만 조회한다.
DISTINCTROW, **UNIQUE** 와 동일하다.

• `<select_expressions>`

- *: **SELECT** * 구문을 사용하면 **FROM** 절에서 명시한 테이블에 대한 모든 칼럼을 조회할 수 있다.
- `expression_comma_list` : `expression` 은 칼럼 이름이나 경로 표현식(예: `tbl_name.col_name`), 변수, 테이블 이름이 될 수 있으며 산술 연산을 포함하는 일반적인 표현식도 모두 사용될 수 있다. 쉼표(,)는 리스트에서 개별 표현식을 구분하는데 사용된다. 조회하고자 하는 칼럼 또는 연산식에 대해 **AS** 키워드를 사용하여 별칭(alias)를 지정할 수 있으며, 지정된 별칭은 칼럼 이름으로 사용되어 **GROUP BY**, **HAVING**, **ORDER BY** 절 내에서 사용될 수 있다. 칼럼의 위치 인덱스(position)는 칼럼이 명시된 순서대로 부여되며, 시작 값은 1이다.

`expression`에는 **AVG**, **COUNT**, **MAX**, **MIN**, **SUM** 과 같이 조회된 데이터를 조작하는 집계 함수가 사용될 수 있다.

- `schema_name`: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- `table_name`:*: 테이블 이름을 지정한다. *을 사용하면 명시한 테이블의 모든 칼럼을 지정하는 것과 같다.
- `variable_comma_list`: `select_expressions` 이 조회하는 데이터는 하나 이상의 변수에 저장될 수 있다.
- `[:]identifier`: **TO** (또는 **INTO**) 다음에 `:identifier`를 조회하는 데이터를 ‘`identifier`’의 변수에 저장할 수 있다.
- `<single_table_spec>`
 - **ONLY** 키워드 뒤에 수퍼클래스 이름이 명시되면, 해당 수퍼클래스만 선택하고 이를 상속받는 서브클래스는 선택하지 않는다.
 - **ALL** 키워드 뒤에 수퍼클래스 이름이 지정되면, 해당 수퍼클래스 및 이를 상속받는 서브클래스를 모두 선택한다.
 - **EXCEPT** 키워드 뒤에 선택하지 않을 서브클래스 리스트를 명시할 수 있다.

다음은 역대 올림픽이 개최된 국가를 중복 없이 조회한 예제이다. 이 예제는 `demodb` 의 `olympic` 테이블을 대상으로 수행하였다. **DISTINCT** 또는 **UNIQUE** 키워드는 질의 결과가 유일한 값만을 갖도록 만든다. 예를 들어 `host_nation` 값이 ‘Greece’인 `olympic` 인스턴스가 여러 개일 때 질의 결과에는 하나의 값만 나타나도록 할 경우에 사용된다.

```
SELECT DISTINCT host_nation
FROM olympic;
```

```
host_nation
=====
'Australia'
'Belgium'
```



```
'Canada'
'Finland'
'France'
...
```

다음은 조회하고자 하는 칼럼에 칼럼 별칭을 부여하고, **ORDER BY** 절에서 칼럼 별칭을 이용하여 결과 레코드를 정렬하는 예제이다. 이때, **LIMIT** 절을 사용하여 결과 레코드 수를 5개로 제한한다.

```
SELECT host_year as col1, host_nation as col2
FROM olympic
ORDER BY col2 LIMIT 5;
```

```
col1  col2
=====
2000  'Australia'
1956  'Australia'
1920  'Belgium'
1976  'Canada'
1948  'England'
```

```
SELECT CONCAT(host_nation, ', ', host_city) AS host_place
FROM olympic
ORDER BY host_place LIMIT 5;
```

```
host_place
=====
'Australia, Melbourne'
'Australia, Sydney'
'Belgium, Antwerp'
'Canada, Montreal'
'England, London'
```

FROM 절

FROM 절은 질의에서 데이터를 조회하고자 하는 테이블을 지정한다. 어떤 테이블도 참조하지 않는 경우에는 **FROM** 절을 생략할 수도 있다. 조회할 수 있는 경로는 다음과 같다.

- 개별 테이블(single table)
- 부질의(subquery)
- 유도 테이블(derived table)
- 원격 테이블(remote table)

```

SELECT [<qualifier>] <select_expressions>
[
    FROM <table_specification> [ {, <table_specification> |
<join_table_specification> }... ]
]

<select_expressions> ::= * | <expression_comma_list> | *,
<expression_comma_list>

<table_specification> ::=
    <single_table_spec> [<correlation>] |
    <remote_table_spec> [<correlation>] |
    <metaclass_specification> [<correlation>] |
    <subquery> <correlation> |
    TABLE (<expression>) <correlation> |
    DBLINK (<dblink_expr>) <dblink_identifier_col_attrs>

<correlation> ::= [AS] <identifier> [( <identifier_comma_list> )]

<single_table_spec> ::= [ONLY] [schema_name.]table_name |
                        ALL [schema_name.]table_name [EXCEPT
[schema_name.]table_name]

<remote_table_spec> ::= [schema_name.]table_name@server_name

<metaclass_specification> ::= CLASS [schema_name.]class_name

```

- <select_expressions>: 조회하고자 하는 칼럼 또는 연산식을 하나 이상 지정할 수 있으며, 테이블 내 모든 칼럼을 조회할 때에는 * 를 지정한다. 조회하고자 하는 칼럼 또는 연산식에 대해 **AS** 키워드를 사용하여 별칭(alias)을 지정할 수 있으며, 지정된 별칭은 칼럼 이름으로 사용되어 **GROUP BY**, **HAVING**, **ORDER BY** 절 내에서 사용될 수 있다. 칼럼의 위치 인덱스(position)는 칼럼이 명시된 순서대로 부여되며, 시작 값은 1이다.
- <table_specification>: **FROM** 절 뒤에 하나 이상의 테이블 이름이 명시되며, 부질의와 유도 테이블도 지정될 수 있다. 부질의 유도 테이블에 대한 설명은 **부질의 유도 테이블**을 참고한다.

```

--FROM clause can be omitted in the statement
SELECT 1+1 AS sum_value;

```

```

sum_value
=====
2

```

```

SELECT CONCAT('CUBRID', '2008' , 'R3.0') AS db_version;

```

```
db_version
=====
'CUBRID2008R3.0'
```

유도 테이블

질의문에서 **FROM** 절의 테이블 명세 부분에 부질의가 사용될 수 있다. 이런 형태의 부질의는 부질의 결과가 테이블로 취급되는 유도 테이블(derived table)을 만든다.

또한 유도 테이블은 집합 값을 갖는 속성의 개별 원소를 접근하는데 사용된다. 이 경우 집합 값의 한 원소는 유도 테이블에서 하나의 레코드로 생성된다.

부질의 유도 테이블

유도 테이블의 각 레코드는 **FROM** 절에 주어진 부질의의 결과로부터 만들어진다. 부질의로부터 생성되는 유도 테이블은 임의의 개수의 칼럼과 레코드를 가질 수 있다.

```
FROM (subquery) [AS] [derived_table_name [(column_name [{,
column_name } ... ]]]]
```

- *column_name* 파라미터의 개수와 *subquery* 에서 만들어지는 칼럼의 개수는 일치해야 한다.
- *derived_table_name*을 생략할 수 있다.

다음은 한국이 획득한 금메달 개수와 일본이 획득한 은메달 개수를 더한 값을 조회하는 예제이다. 이 예제는 유도 테이블을 이용하여 부질의의 중간 결과를 모으고 하나의 결과로 처리하는 방법을 보여준다. 이 질의는 *nation_code* 칼럼이 'KOR'인 *gold* 값과 *nation_code* 칼럼이 'JPN'인 *silver* 값의 전체 합을 반환한다.

```
SELECT SUM (n)
FROM (SELECT gold FROM participant WHERE nation_code = 'KOR'
      UNION ALL
      SELECT silver FROM participant WHERE nation_code = 'JPN') AS
t(n);
```

부질의 유도 테이블은 외부 질의와 연관되어 있을 때 유용하게 사용할 수 있다. 예를 들어 **WHERE** 절에서 사용된 부질의의 **FROM** 절에 유도 테이블이 사용될 수 있다. 다음은 은메달 및 동메달을 하나 이상 획득한 경우, 해당 은메달과 동메달의 합이 평균보다 많은 수의 금메달을 획득한 *nation_code*, *host_year*, *gold* 필드를 보여주는 질의 예제이다. 이 예제에서는 질의(외부 **SELECT** 절)와 부질의(내부 **SELECT** 절)가 *nation_code* 속성으로 연결되어 있다.

```

SELECT nation_code, host_year, gold
FROM participant p
WHERE gold > (SELECT AVG(s)
              FROM (SELECT silver + bronze
                    FROM participant
                    WHERE nation_code = p.nation_code
                    AND silver > 0
                    AND bronze > 0)
              AS t(s));

```

nation_code	host_year	gold
'JPN'	2004	16
'CHN'	2004	32
'DEN'	1996	4
'ESP'	1992	13

원격 테이블

FROM절에 원격 테이블 (remote table) 을 명시할 수 있으며, 원격 테이블을 명시할 때는 '@'를 사용하여 table_name@server_name와 같은 테이블 확장명을 사용한다. 원격 서버는 CUBRID 뿐만 아니라 타 DBMS(ORACLE, MySQL 또는 MariaDB)을 사용할 수 있다. 원격 테이블은 최적화 단계를 거치면서 DBLINK 구문으로 재 작성되어 실행된다.

```

-- at remote-side, "remote_server"
CREATE TABLE remote_tbl (
  id INT,
  name VARCHAR(32)
);

INSERT INTO remote_tbl VALUES (1, 'Kim');
INSERT INTO remote_tbl VALUES (2, 'Lee');
INSERT INTO remote_tbl VALUES (3, 'Park');

```

```

-- at local-side
SELECT *
FROM remote_tbl@remote_server rem
WHERE id < 3;

```

id	name
=====	

1	Kim
2	Lee

최적화 단계에서 원격테이블에 대해 DBLINK 구문으로 재 작성된 쿼리는 아래와 같다.

```
SELECT *
FROM DBLINK(remote_server, 'SELECT id, name FROM remote_tbl WHERE id
< 3') AS dbl (id INT, name VARCHAR(32));
```

참고

테이블 확장명에 허용되는 객체는 일반 테이블, 동의어(synonym), 그리고 뷰가 있다. 아래의 예는 세가지 형태의 테이블 확장명을 보여주고 있다.

```
-- at remote-side
CREATE TABLE remote_table (
    id INT,
    phone VARCHAR(12)
);

CREATE SYNONYM a_remote_tbl FOR user_a.remote_table
CREATE VIEW v_remote_tbl(r_phone) AS SELECT phone FROM remote_tbl
WHERE id > 10;
```

```
-- at local-side

-- remote-table
SELECT phone FROM user_a.remote_table@server1 WHERE id > 10;

-- remote-synonym
SELECT phone FROM a_remote_tbl@server1 WHERE id > 10;

-- remote-view
SELECT r_phone FROM v_remote_tbl@server1;
```

위 3개의 쿼리는 모두 동일한 결과를 리턴한다.

DBLINK

원격지에 있는 별도의 DBMS에서 질의를 수행하여 그 결과를 얻을 수 있다. 그 결과는 일종의 부질의로 **유도 테이블(derived table)**로 생성된다.

```
FROM DBLINK (<dblink_expr>) [AS] <dblink_identifier_col_attrs>

    <dblink_expr> ::= <dblink_conn>, remote_query_sting
    <dblink_conn> ::= server_name | dblink_conn_string

    <dblink_identifier_col_attrs> ::= dblink_table_alias (
<dblink_column_definition_list> )
    <dblink_column_definition_list> ::= dblink_column_alias
<primitive_type> [{, dblink_column_alias <primitive_type>} ...]
```

- *remote_query_sting*: 원격지 DBMS에 전달할 질의문으로 SELECT 쿼리만 지정 할 수 있다.
- *server_name*: 서버 정의를 사용해서 생성한 서버 이름.
- *dblink_conn_string*: 문자열로 표현된 원격지 접속 정보.
- *dblink_table_alias*: DBLINK를 이용하여 생성하는 유도테이블(derived table) 이름.
- *dblink_column_alias*: DBLINK의 *remote_query_sting* 의 select list에 대응하는 가상의 컬럼명.

참고

DBLINK에서 지원하는 컬럼의 속성은 다음과 같다.

- INT, BIGINT, SHORT, FLOAT, DOUBLE, MONETARY, NUMERIC
- VARCHAR, CHAR
- DATE, TIME, TIMESTAMP, DATETIME
- DATETIMEETZ, DATETIMELTZ, TIMESTAMPTZ, TIMESTAMPLTZ

경고

DBLINK에서는 다음과 같은 속성의 컬럼은 지원하지 않는다.

- COLLECTION TYPE (SET, MULTISSET, SEQUENCE)
- OBJECT
- CLOB / BLOB
- ENUM
- BIT / BIT VARYING

• JSON

i 참고

*dblink_conn_string*은 아래와 같은 구조로 구성된다. 각각의 내용은 **서버 정의문** 구문의 HOST, PORT, DBNAME, USER, PASSWOED, PROPERTIES에 해당하는 정보이다. 각 항목은 ‘:’ 문자로 구분된다.

```
<broker-host>:<port#>:<db_name>:<db_user>:<db_password>:[?<properties>]
```

비밀번호의 노출을 막기 위해서는 *dblink_conn_string*을 이용하기 보다는 *server_name*을 이용하는 것을 권장한다.

```
CREATE SERVER remote_srv1 ( HOST='127.0.0.1', PORT=3300,
DBNAME=demodb, USER=cub, PASSWORD='cub-password');
SELECT * FROM DBLINK (remote_srv1, 'SELECT col1 FROM remote_t') AS
t(col1 int);

SELECT * FROM DBLINK ('127.0.0.1:3300:demodb:cub:cub-
password:', 'SELECT col1, col2 FROM remote_t') AS t(col1 int, col2
varchar(32));
```

위 예시에서 두 SELECT 구문은 동일한 기능을 수행한다.

WHERE 절

질의에서 칼럼은 조건에 따라 처리될 수 있다. **WHERE** 절은 조회하려는 데이터의 조건을 명시한다.

```
WHERE <search_condition>

<search_condition> ::=
    <comparison_predicate>
    <between_predicate>
    <exists_predicate>
    <in_predicate>
    <null_predicate>
    <like_predicate>
    <quantified_predicate>
    <set_predicate>
```

WHERE 절은 *search_condition* 또는 질의에서 조회되는 데이터를 결정하는 조건식을 지정한다. 조건식이 참인 데이터만 질의 결과로 조회된다(**NULL** 값은 알 수 없는 값으로서 질의 결과로 조회되지 않는다).

• *search_condition*: 자세한 내용은 다음의 항목을 참고한다.

- 단순 비교 조건식
- BETWEEN
- EXISTS
- IN
- IS NULL
- LIKE
- ANY/SOME/ALL 수량어와 그룹 조건식

복수의 조건은 논리연산자 **AND**, **OR** 를 사용할 수 있다. **AND** 가 지정된 경우 모든 조건이 참이어야 하고, **OR** 로 지정된 경우에는 하나의 조건만 참이어도 된다. 만약 키워드 **NOT** 이 조건 앞에 붙는다면 조건은 반대의 의미를 갖는다. 논리 연산이 평가되는 순서는 다음 표와 같다.

우선순위	연산자	기능
1	()	괄호 내에 포함된 논리 표현식은 첫 번째로 평가된다.
2	NOT	논리 표현식의 결과를 부정한다.
3	AND	논리 표현식에 포함된 모든 조건이 참이어야 한다.
4	OR	논리 표현식에 포함된 조건 중 하나의 조건은 참이어야 한다.

GROUP BY ... HAVING 절

SELECT 문으로 검색한 결과를 특정 칼럼을 기준으로 그룹화하기 위해 **GROUP BY** 절을 사용하며, 그룹별로 정렬을 수행하거나 집계 함수를 사용하여 그룹별 집계를 구할 때 사용한다. 그룹이란 **GROUP BY** 절에 명시된 칼럼에 대해 동일한 칼럼 값을 가지는 레코드들을 의미한다.

GROUP BY 절 뒤에 **HAVING** 절을 결합하여 그룹 선택을 위한 조건식을 설정할 수 있다. 즉, **GROUP BY** 절로 구성되는 모든 그룹 중 **HAVING** 절에 명시된 조건식을 만족하는 그룹만 조회한다.

SQL 표준에서는 **GROUP BY** 절에서 명시되지 않은 칼럼(hidden column)을 **SELECT** 칼럼 리스트에 명시할 수 없지만, CUBRID는 문법을 확장하여 **GROUP BY** 절에서 명시되지 않은 칼럼도 **SELECT** 칼럼 리스트에 명시할 수 있다. 확장된 문법을 사용하지 않으려면 **only_full_group_by** 파라미터 값을 yes로 설정해야 한다. 이에 대한 자세한 내용은 [구문/타입 관련 파라미터](#) 를 참고한다.

```
SELECT ...
GROUP BY {col_name | expr | position} [ASC | DESC], ...
        [WITH ROLLUP] [HAVING <search_condition>]
```

- *col_name | expr | position*: 하나 이상의 칼럼 이름, 표현식, 별칭 또는 칼럼 위치가 지정될 수 있으며, 각 항목은 쉼표로 구분된다. 이를 기준으로 칼럼들이 정렬된다.
- **[ASC | DESC]**: **GROUP BY** 절 내에 명시된 칼럼 뒤에 **ASC** 또는 **DESC**의 정렬 옵션을 명시할 수 있다. 정렬 옵션이 명시되지 않으면 기본 옵션은 **ASC**가 된다.
- *<search_condition>*: **HAVING** 절에 검색 조건식을 명시한다. **HAVING** 절에서는 **GROUP BY** 절 내에 명시된 칼럼과 별칭, 또는 집계 함수에서 사용되는 칼럼을 참조할 수 있다.

참고

cubrid.conf의 **only_full_group_by** 파라미터의 값이 yes인 경우 **GROUP BY** 절에서 명시되지 않은 칼럼(hidden columns)을 참조할 수도 있는데, 이때 **HAVING** 조건은 질의 결과에 영향을 끼치지 않는다.

- **WITH ROLLUP**: **GROUP BY** 절에 **WITH ROLLUP** 수정자를 명시하면, **GROUP BY** 된 칼럼 각각에 대한 결과 값이 그룹별로 집계되고 나서, 해당 그룹 행의 전체를 집계한 결과 값이 추가로 출력된다. 즉, 그룹별로 집계한 값에 대해 다시 전체 집계를 수행한다. 그룹 대상 칼럼이 두 개 이상일 경우 앞의 그룹을 큰 단위, 뒤의 그룹을 작은 단위로 간주하여 작은 단위 별 전체 집계 행과 큰 단위의 전체 집계 행이 추가된다. 예를 들어 부서별, 사람별 영업 실적의 집계를 하나의 질의문으로 확인할 수 있다.

```
-- creating a new table
CREATE TABLE sales_tbl
(dept_no INT, name VARCHAR(20), sales_month INT, sales_amount INT
DEFAULT 100, PRIMARY KEY (dept_no, name, sales_month));
```

```

INSERT INTO sales_tbl VALUES
(201, 'George' , 1, 450), (201, 'George' , 2, 250), (201, 'Laura' ,
1, 100), (201, 'Laura' , 2, 500),
(301, 'Max' , 1, 300), (301, 'Max' , 2, 300),
(501, 'Stephan', 1, 300), (501, 'Stephan', 2, DEFAULT), (501,
'Chang' , 1, 150), (501, 'Chang' , 2, 150),
(501, 'Sue' , 1, 150), (501, 'Sue' , 2, 200);

-- selecting rows grouped by dept_no
SELECT dept_no, avg(sales_amount)
FROM sales_tbl
GROUP BY dept_no;

```

dept_no	avg(sales_amount)
201	3.2500000000000000e+02
301	3.0000000000000000e+02
501	1.7500000000000000e+02

```

-- conditions in WHERE clause operate first before GROUP BY
SELECT dept_no, avg(sales_amount)
FROM sales_tbl
WHERE sales_amount > 100
GROUP BY dept_no;

```

dept_no	avg(sales_amount)
201	4.0000000000000000e+02
301	3.0000000000000000e+02
501	1.9000000000000000e+02

```

-- conditions in HAVING clause operate last after GROUP BY
SELECT dept_no, avg(sales_amount)
FROM sales_tbl
WHERE sales_amount > 100
GROUP BY dept_no HAVING avg(sales_amount) > 200;

```

dept_no	avg(sales_amount)
201	4.0000000000000000e+02
301	3.0000000000000000e+02

```
-- selecting and sorting rows with using column alias
SELECT dept_no AS a1, avg(sales_amount) AS a2
FROM sales_tbl
WHERE sales_amount > 200 GROUP
BY a1 HAVING a2 > 200
ORDER BY a2;
```

a1	a2
301	3.0000000000000000e+02
501	3.0000000000000000e+02
201	4.0000000000000000e+02

```
-- selecting rows grouped by dept_no, name with WITH ROLLUP modifier
SELECT dept_no AS a1, name AS a2, avg(sales_amount) AS a3
FROM sales_tbl
WHERE sales_amount > 100
GROUP BY a1, a2 WITH ROLLUP;
```

a1	a2	a3
201	'George'	3.5000000000000000e+02
201	'Laura'	5.0000000000000000e+02
201	NULL	4.0000000000000000e+02
301	'Max'	3.0000000000000000e+02
301	NULL	3.0000000000000000e+02
501	'Chang'	1.5000000000000000e+02
501	'Stephan'	3.0000000000000000e+02
501	'Sue'	1.7500000000000000e+02
501	NULL	1.9000000000000000e+02
NULL	NULL	2.7500000000000000e+02

ORDER BY 절

ORDER BY 절은 질의 결과를 오름차순 또는 내림차순으로 정렬하며, **ASC** 또는 **DESC** 와 같은 정렬 옵션을 명시하지 않으면 오름차순으로 정렬한다. **ORDER BY** 절을 지정하지 않으면, 조회되는 레코드의 순서는 질의에 따라 다르다.

```
SELECT ...
ORDER BY {col_name | expr | position} [ASC | DESC], ...] [NULLS
{FIRST | LAST}]
```

- *col_name | expr | position*: 정렬 기준이 되는 칼럼 이름, 표현식, 별칭 또는 칼럼 위치를 지정한다. 하나 이상의 값을 지정할 수 있으며 각 항목은 쉼표로 구분한다. **SELECT** 칼럼 리스트에 명시되지 않은 칼럼도 지정할 수 있다.
- **[ASC | DESC]**: **ASC** 은 오름차순, **DESC** 은 내림차순으로 정렬하며, 정렬 옵션이 명시되지 않으면 오름차순으로 정렬한다.
- **[NULLS {FIRST | LAST}]**: **NULLS FIRST**는 NULL을 앞에 정렬하며, **NULLS LAST**는 NULL을 뒤에 정렬한다. 이 구문이 생략될 경우 **ASC**는 NULL을 앞에 정렬하며, **DESC**는 NULL을 뒤에 정렬한다.

```
-- selecting rows sorted by ORDER BY clause
SELECT *
FROM sales_tbl
ORDER BY dept_no DESC, name ASC;
```

dept_no	name	sales_month	sales_amount
501	'Chang'	1	150
501	'Chang'	2	150
501	'Stephan'	1	300
501	'Stephan'	2	100
501	'Sue'	1	150
501	'Sue'	2	200
301	'Max'	1	300
301	'Max'	2	300
201	'George'	1	450
201	'George'	2	250
201	'Laura'	1	100
201	'Laura'	2	500

```
-- sorting reversely and limiting result rows by LIMIT clause
SELECT dept_no AS a1, avg(sales_amount) AS a2
FROM sales_tbl
GROUP BY a1
ORDER BY a2 DESC
LIMIT 3;
```

a1	a2
201	3.2500000000000000e+02

```
301      3.0000000000000000e+02
501      1.7500000000000000e+02
```

다음은 ORDER BY 절 뒤에 NULLS FIRST, NULLS LAST 구문을 지정하는 예제이다.

```
CREATE TABLE tbl (a INT, b VARCHAR);
```

```
INSERT INTO tbl VALUES
```

```
(1,NULL), (2,NULL), (3,'AB'), (4,NULL), (5,'AB'),
(6,NULL), (7,'ABCD'), (8,NULL), (9,'ABCD'), (10,NULL);
```

```
SELECT * FROM tbl ORDER BY b NULLS FIRST;
```

```

      a  b
=====
      1  NULL
      2  NULL
      4  NULL
      6  NULL
      8  NULL
     10  NULL
      3  'ab'
      5  'ab'
      7  'abcd'
      9  'abcd'
```

```
SELECT * FROM tbl ORDER BY b NULLS LAST;
```

```

      a  b
=====
      3  'ab'
      5  'ab'
      7  'abcd'
      9  'abcd'
      1  NULL
      2  NULL
      4  NULL
      6  NULL
      8  NULL
     10  NULL
```

GROUP BY 별칭(alias)의 해석

```
CREATE TABLE t1(a INT, b INT, c INT);
INSERT INTO t1 VALUES(1,1,1);
INSERT INTO t1 VALUES(2,NULL,2);
INSERT INTO t1 VALUES(2,2,2);

SELECT a, NVL(b,2) AS b
FROM t1
GROUP BY a, b; -- Q1
```

위의 SELECT 질의를 수행할 때 “GROUP BY a, b”는

- 9.2 이하 버전에서 “GROUP BY a, NVL(b, 2)”(별칭 이름 b)로 해석되며, 아래 Q2와 동일한 결과를 출력한다.

```
SELECT a, NVL(b,2) AS bxxx
FROM t1
GROUP BY a, bxxx; -- Q2
```

a	b
1	1
2	2

- 9.3 이상 버전에서 “GROUP BY a, b”(칼럼 이름 b)로 해석되며, 아래 Q3와 동일한 결과를 출력한다.

```
SELECT a, NVL(b,2) AS bxxx
FROM t1
GROUP BY a, b; -- Q3
```

a	b
1	1
2	2
2	2

LIMIT 절

LIMIT 절은 출력되는 레코드의 개수를 제한할 때 사용한다. **LIMIT** 절은 prepared statement에 포함하여 사용할 수 있으며, 인자로 바인드 파라미터를 사용할 수 있다.

LIMIT 절을 포함하는 질의에서는 **WHERE** 절에 **INST_NUM ()**, **ROWNUM** 을 포함할 수 없으며, **HAVING GROUPBY_NUM ()**과 함께 사용할 수 없다.

```
LIMIT {[offset,] row_count | row_count [OFFSET offset]}
```

```
<offset> ::= <limit_expression>
<row_count> ::= <limit_expression>
```

```
<limit_expression> ::= <limit_term> | <limit_expression> +
<limit_term> | <limit_expression> - <limit_term>
<limit_term> ::= <limit_factor> | <limit_term> * <limit_factor> |
<limit_term> / <limit_factor>
<limit_factor> ::= <unsigned_int> | <input_hostvar> | (
<limit_expression> )
```

- *offset*: 출력할 레코드의 시작 행의 오프셋을 지정한다. 결과 셋에 있는 시작 행의 오프셋은 0이다. 생략할 수 있으며 기본값은 0 이다. 부호 없는 정수, 호스트 변수 또는 간단한 표현식 중 하나일 수 있다.
- *row_count*: 출력하고자 하는 레코드 개수를 명시한다. 부호 없는 정수, 호스트 변수 또는 간단한 표현식 중 하나일 수 있다.

```
-- LIMIT clause can be used in prepared statement
PREPARE stmt FROM 'SELECT * FROM sales_tbl LIMIT ?, ?';
EXECUTE stmt USING 0, 10;
```

```
-- selecting rows with LIMIT clause
SELECT *
FROM sales_tbl
WHERE sales_amount > 100
LIMIT 5;
```

dept_no	name	sales_month	sales_amount
201	'George'	1	450
201	'George'	2	250
201	'Laura'	2	500
301	'Max'	1	300
301	'Max'	2	300

```
-- LIMIT clause can be used in subquery
SELECT t1.*
FROM (SELECT * FROM sales_tbl AS t2 WHERE sales_amount > 100 LIMIT
5) AS t1
LIMIT 1,3;

-- above query and below query shows the same result
SELECT t1.*
FROM (SELECT * FROM sales_tbl AS t2 WHERE sales_amount > 100 LIMIT
5) AS t1
LIMIT 3 OFFSET 1;
```

dept_no	name	sales_month	sales_amount
201	'George'	2	250
201	'Laura'	2	500
301	'Max'	1	300

```
-- LIMIT clause allows simple expressions for both offset and
row_count
SELECT *
FROM sales_tbl
WHERE sales_amount > 100
LIMIT ? * ?, (? * ?) + ?;
```

조인 질의

조인은 두 개 이상의 테이블 또는 뷰(view)에 대해 행(row)을 결합하는 질의이다. 조인 질의에서 두 개 이상의 테이블에 공통인 칼럼을 비교하는 조건을 조인 조건이라고 하며, 조인된 각 테이블로부터 행을 가져와 지정된 조인 조건을 만족하는 경우에만 결과 행을 결합한다.

조인 질의에서 동등 연산자(=)를 이용한 조인 조건을 포함하는 조인 질의를 동등 조인(equi-join)이라 하고, 조인 조건이 없는 조인 질의를 카티션 곱(cartesian products)이라 한다. 또한, 하나의 테이블을 조인하는 경우를 자체 조인(self join)이라 하는데, 자체 조인에서는 **FROM** 절에 같은 테이블이 두 번 사용되므로 테이블 별칭(alias)을 사용하여 칼럼을 구분한다.

조인된 테이블에 대해 조인 조건을 만족하는 행만 결과를 출력하는 경우를 내부 조인(inner join) 또는 간단 조인(simple join)이라고 하는 반면, 조인된 테이블에 대해 조인 조건을 만족하는 행은 물론 조인 조건을 만족하지 못하는 행도 포함하여 출력하는 경우를 외부 조인(outer join)이라 한다.

외부 조인은 왼쪽 테이블의 모든 행이 결과로 출력되는(조건과 일치하지 않는 오른쪽 테이블의 칼럼들은 NULL로 출력됨) 왼쪽 외부 조인과(left outer join)과 오른쪽 테이블의 모든 행이 결과로 출력되는(조건과 일치하지 않는 왼쪽 테이블의 칼럼들은 NULL로 출력됨) 오른쪽 외부 조인(right outer

join)이 있으며, 양쪽의 행이 모두 출력되는 완전 외부 조인(full outer join)이 있다. 외부 조인 질의 결과에서 한쪽 테이블에 대해 대응되는 칼럼 값이 없는 경우, 이는 모두 **NULL**을 반환된다.

```
FROM <table_specification> [{, <table_specification>
    | { <join_table_specification> |
<join_table_specification2> } ...]

<table_specification> ::=
    <single_table_spec> [<correlation>] |
    <metaclass_specification> [<correlation>] |
    <subquery> <correlation> |
    TABLE (<expression>) <correlation>

<join_table_specification> ::=
    {
        [INNER | {LEFT | RIGHT} [OUTER]] JOIN

    } <table_specification> ON <search_condition>

<join_table_specification2> ::=
    {
        CROSS JOIN |
        NATURAL [LEFT | RIGHT] JOIN
    } <table_specification>
```

• <join_table_specification>

- **[INNER] JOIN**: 내부 조인에 사용되며 조인 조건이 반드시 필요하다.
- **{LEFT | RIGHT} [OUTER] JOIN**: **LEFT** 는 왼쪽 외부 조인을 수행하는 질의를 만드는데 사용되고, **RIGHT** 는 오른쪽 외부 조인을 수행하는 질의를 만드는데 사용된다.

• <join_table_specification2>

- **CROSS JOIN**: 교차 조인에 사용되며, 조인 조건을 사용하지 않는다.
- **NATURAL [LEFT | RIGHT] JOIN**: 자연 조인에 사용되며, 조인 조건을 사용하지 않는다. 같은 이름의 칼럼끼리 동등 조건을 가지는 것과 같이 동작한다.

내부 조인

내부 조인은 조인을 위한 조건이 반드시 필요하다. **INNER JOIN** 키워드는 생략할 수 있으며, 생략하면 테이블 사이를 쉼표(,)로 구분하고, **ON** 조인 조건을 **WHERE** 조건으로 대체할 수 있다.

다음은 내부 조인을 이용하여 1950년 이후에 열린 올림픽 중에서 신기록이 세워진 올림픽의 개최연도와 개최국가를 조회하는 예제이다. 다음 질의는 *history* 테이블의 *host_year* 가 1950보다 큰 범위에서 값이 존재하는 레코드를 가져온다. 다음 두 개의 질의는 같은 결과를 출력한다.

```
SELECT DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o ON h.host_year = o.host_year AND
o.host_year > 1950;
```

```
SELECT DISTINCT h.host_year, o.host_nation
FROM history h, olympic o
WHERE h.host_year = o.host_year AND o.host_year > 1950;
```

```
host_year  host_nation
=====
1968      'Mexico'
1980      'U.S.S.R.'
1984      'United States of America'
1988      'Korea'
1992      'Spain'
1996      'United States of America'
2000      'Australia'
2004      'Greece'
```

외부 조인

CUBRID는 외부 조인 중 왼쪽 외부 조인과 오른쪽 외부 조인만 지원하며, 완전 외부 조인(full outer join)을 지원하지 않는다. 또한, 외부 조인에서 조인 조건에 부질의와 하위 칼럼을 포함하는 경로 표현식을 사용할 수 없다.

외부 조인의 경우 조인 조건은 내부 조인의 경우와는 다른 방법으로 지정된다. 내부 조인의 조인 조건은 **WHERE** 절에서도 표현될 수 있지만, 외부 조인의 경우에는 조인 조건이 **FROM** 절 내의 **ON** 키워드 뒤에 나타난다. 다른 검색 조건은 **WHERE** 절이나 **ON** 절에서 사용할 수 있지만 검색 조건이 **WHERE** 절에 있을 때와 **ON** 절에 있을 때 질의 결과가 달라질 수 있다.

FROM 절에 명시된 순서대로 테이블 실행 순서가 고정되므로, 외부 조인을 사용하는 경우 테이블 순서에 주의하여 질의문을 작성한다. 외부 조인 연산자 **(+)**를 **WHERE** 절에 명시하여 Oracle 스타일의 조인 질의문도 작성 가능하나, 실행 결과나 실행 계획이 원하지 않는 방향으로 유도될 수 있으므로 **{LEFT | RIGHT} [OUTER] JOIN**을 이용한 표준 구문을 사용할 것을 권장한다.

다음은 오른쪽 외부 조인을 이용하여 1950년 이후에 열린 올림픽에서 신기록이 세워진 올림픽의 개최국가와 개최연도를 조회하되, 신기록이 세워지지 않은 올림픽에 대한 정보도 포함하는 예제이다. 이 예제는 오른쪽 외부 조인이므로, *olympic* 테이블의 *host_nation* 의 모든 레코드를 포함하고, 값이 존재하지 않는 *history* 테이블의 *host_year*에 대해서는 칼럼 값으로 **NULL**을 반환한다.

```
SELECT DISTINCT h.host_year, o.host_year, o.host_nation
FROM history h RIGHT OUTER JOIN olympic o ON h.host_year =
o.host_year
WHERE o.host_year > 1950;
```

host_year	host_year	host_nation
NULL	1952	'Finland'
NULL	1956	'Australia'
NULL	1960	'Italy'
NULL	1964	'Japan'
NULL	1972	'Germany'
NULL	1976	'Canada'
1968	1968	'Mexico'
1980	1980	'USSR'
1984	1984	'USA'
1988	1988	'Korea'
1992	1992	'Spain'
1996	1996	'USA'
2000	2000	'Australia'
2004	2004	'Greece'

다음은 왼쪽 외부 조인을 이용하여 위와 동일한 결과를 출력하는 예제이다. **FROM** 절에서 두 테이블의 순서를 바꾸어 명시한 후, 왼쪽 외부 조인을 수행한다.

```
SELECT DISTINCT h.host_year, o.host_year, o.host_nation
FROM olympic o LEFT OUTER JOIN history h ON h.host_year = o.host_year
WHERE o.host_year > 1950;
```

host_year	host_year	host_nation
NULL	1952	'Finland'
NULL	1956	'Australia'
NULL	1960	'Italy'
NULL	1964	'Japan'
NULL	1972	'Germany'
NULL	1976	'Canada'
1968	1968	'Mexico'
1980	1980	'USSR'
1984	1984	'USA'
1988	1988	'Korea'
1992	1992	'Spain'
1996	1996	'USA'
2000	2000	'Australia'
2004	2004	'Greece'

다음은 **WHERE** 절에서 (+)를 사용해서 외부 조인 질의를 작성한 예이며, 위와 같은 결과를 출력한다. 단, (+) 연산자를 이용한 Oracle 스타일의 외부 조인 질의문은 ISO/ANSI 표준이 아니며 모호한 상황을 만들어 낼 수 있으므로 가능하면 표준 구문인 **LEFT OUTER JOIN**(또는 **RIGHT OUTER JOIN**)을 사용할 것을 권장한다.

```
SELECT DISTINCT h.host_year, o.host_year, o.host_nation
FROM history h, olympic o
WHERE o.host_year = h.host_year(+) AND o.host_year > 1950;
```

host_year	host_year	host_nation
NULL	1952	'Finland'
NULL	1956	'Australia'
NULL	1960	'Italy'
NULL	1964	'Japan'
NULL	1972	'Germany'
NULL	1976	'Canada'
1968	1968	'Mexico'
1980	1980	'USSR'
1984	1984	'USA'
1988	1988	'Korea'
1992	1992	'Spain'
1996	1996	'USA'
2000	2000	'Australia'
2004	2004	'Greece'

이상의 예에서 $h.host\_year = o.host\_year$ 는 외부 조인 조건이고 $o.host\_year > 1950$ 은 검색 조건이다. 만약 검색 조건이 **WHERE** 절이 아닌 **ON** 절에서 조인 조건으로 사용될 경우 질의의 의미와 결과는 달라진다. 다음 질의는 $o.host\_year$ 가 1950보다 크지 않은 값도 질의 결과에 포함된다.

```
SELECT DISTINCT h.host_year, o.host_year, o.host_nation
FROM olympic o LEFT OUTER JOIN history h ON h.host_year =
o.host_year AND o.host_year > 1950;
```

host_year	host_year	host_nation
NULL	1896	'Greece'
NULL	1900	'France'
NULL	1904	'USA'
NULL	1908	'United Kingdom'
NULL	1912	'Sweden'
NULL	1920	'Belgium'
NULL	1924	'France'
NULL	1928	'Netherlands'
NULL	1932	'USA'
NULL	1936	'Germany'
NULL	1948	'England'
NULL	1952	'Finland'
NULL	1956	'Australia'
NULL	1960	'Italy'
NULL	1964	'Japan'

```

NULL      1972  'Germany'
NULL      1976  'Canada'
1968      1968  'Mexico'
1980      1980  'USSR'
1984      1984  'USA'
1988      1988  'Korea'
1992      1992  'Spain'
1996      1996  'USA'
2000      2000  'Australia'
2004      2004  'Greece'

```

위의 예에서 **LEFT OUTER JOIN**은 왼쪽 테이블의 행이 조건에 부합하지 않더라도 모든 행을 결과 행에 결합해야 하므로, 왼쪽 테이블의 칼럼 조건인 “AND o.host_year > 1950”는 이므로 무시된다. 그러나 “WHERE o.host_year > 1950”는 조인이 완료된 이후에 적용된다. **OUTER JOIN**에서는 **ON** 절 뒤의 조건과 **WHERE** 절 뒤의 조건이 다르게 적용될 수 있음에 주의해야 한다.

교차 조인

교차 조인은 아무런 조건 없이 두 개의 테이블을 결합한 것, 즉 카티션 곱(cartesian product)이다. 교차 조인에서 **CROSS JOIN** 키워드는 생략할 수 있으며, 생략하려면 테이블 사이를 쉼표(,)로 구분한다.

다음은 내부 조인을 이용하여 1950년 이후에 열린 올림픽 중에서 신기록이 세워진 올림픽의 개최연도와 개최국가를 조회하는 예제이다. 다음 질의는 *history* 테이블의 *host_year* 가 1950보다 큰 범위에서 값이 존재하는 레코드를 가져온다.

다음은 교차 조인을 작성한 예이다. 다음 두 개의 질의는 같은 결과를 출력한다.

```

SELECT DISTINCT h.host_year, o.host_nation
FROM history h CROSS JOIN olympic o;

```

```

SELECT DISTINCT h.host_year, o.host_nation
FROM history h, olympic o;

```

```

      host_year  host_nation
=====
      1968      'Australia'
      1968      'Belgium'
      1968      'Canada'
      1968      'England'
      1968      'Finland'
      1968      'France'
      1968      'Germany'
...
      2004      'Spain'
      2004      'Sweden'
      2004      'USA'
      2004      'USSR'

```

```
2004 'United Kingdom'
```

```
144 rows selected. (1.283548 sec) Committed.
```

자연 조인

각 테이블에서 조인할 칼럼 이름이 같은 경우 즉, 해당 칼럼끼리 동등 조건(=)을 부여하고자 하는 경우 내부/외부 조인을 대체하는 자연 조인(natural join)을 사용할 수 있다.

```
CREATE TABLE t1 (a int, b1 int);
CREATE TABLE t2 (a int, b2 int);

INSERT INTO t1 values(1,1);
INSERT INTO t1 values(3,3);
INSERT INTO t2 values(1,1);
INSERT INTO t2 values(2,2);
```

다음은 **NATURAL JOIN**을 수행하는 예이다.

```
SELECT /** RECOMPILE*/ *
FROM t1 NATURAL JOIN t2;
```

위의 질의를 수행하는 것은 아래의 질의를 수행하는 것과 동일하며, 같은 결과를 출력한다.

```
SELECT /** RECOMPILE*/ *
FROM t1 INNER JOIN t2 ON t1.a=t2.a;
```

a	b1	a	b2
1	1	1	1

다음은 **NATURAL LEFT JOIN**을 수행하는 예이다.

```
SELECT /** RECOMPILE*/ *
FROM t1 NATURAL LEFT JOIN t2;
```

위의 질의를 수행하는 것은 아래의 질의를 수행하는 것과 동일하며, 같은 결과를 출력한다.

```
SELECT /** RECOMPILE*/ *
FROM t1 LEFT JOIN t2 ON t1.a=t2.a;
```

a	b1	a	b2
1	1	1	1
3	3	NULL	NULL

다음은 **NATURAL RIGHT JOIN**을 수행하는 예이다.

```
SELECT /** RECOMPILE*/ *
FROM t1 NATURAL RIGHT JOIN t2;
```

위의 질의는 아래의 질의를 수행하는 것과 동일하며, 같은 결과를 출력한다.

```
SELECT /** RECOMPILE*/ *
FROM t1 RIGHT JOIN t2 ON t1.a=t2.a;
```

a	b1	a	b2
1	1	1	1
NULL	NULL	2	2

부질의

부질의는 질의 내에서 **SELECT** 절이나 **WHERE** 절 등 표현식이 가능한 모든 곳에서 사용할 수 있다. 부질의가 표현식으로 사용될 경우에는 반드시 단일 칼럼을 반환해야 하지만, 표현식이 아닌 경우에는 하나 이상의 행이 반환될 수 있다. 부질의가 사용되는 경우에 따라 단일 행 부질의와 다중 행 부질의로 나눌 수 있다.

단일 행 부질의

단일 행 부질의는 하나의 칼럼을 갖는 하나의 행을 만든다. 부질의에 의해 행이 반환되지 않을 경우에 부질의 표현식은 **NULL** 을 가진다. 만약 부질의가 두 개 이상의 행을 반환하도록 만들어진 경우에는 에러가 발생한다.

다음은 역대 기록 테이블을 조회하는데, 신기록을 수립한 올림픽이 개최된 국가도 함께 조회하는 예제이다. 이 예제는 표현식으로 사용된 단일 행 부질을 보여준다. 이 예에서 부질의는 *olympic* 테이블에서 *host_year* 칼럼 값이 *history* 테이블의 *host_year* 칼럼 값과 같은 행에 대해 *host_nation* 값을 반환한다. 조건에 일치되는 값이 없을 경우 부질의 결과는 **NULL** 이 표시된다.

```
SELECT h.host_year, (SELECT host_nation FROM olympic o WHERE
o.host_year=h.host_year) AS host_nation,
```

```

        h.event_code, h.score, h.unit
FROM history h;

```

```

        host_year  host_nation      event_code
score            unit
=====
=====
        2004      'Greece'          20283
'07:53.0'         'time'
        2004      'Greece'          20283
'07:53.0'         'time'
        2004      'Greece'          20281
'03:57.0'         'time'
        2004      'Greece'          20281
'03:57.0'         'time'
        2004      'Greece'          20281
'03:57.0'         'time'
        2004      'Greece'          20281
'03:57.0'         'time'
        2004      'Greece'          20326
'210'             'kg'
        2000      'Australia'        20328
'225'             'kg'
        2004      'Greece'          20331
'237.5'           'kg'
...

```

다중 행 부질의

다중 행 부질의는 지정된 칼럼을 갖는 하나 이상의 행을 반환한다. 다중 행 부질의 결과는 적절한 키워드를 사용하여 **SET**, **MULTISET**, **LIST (= SEQUENCE)**를 만드는데 사용될 수 있다.

다음은 국가 테이블에서 국가 이름과 수도 이름을 조회하되, 올림픽을 개최한 국가는 개최도시를 **LIST** 로 묶어 함께 조회하는 예제이다. 이 예제 같은 경우는 부질의 결과를 이용하여 *olympic* 테이블의 *host_city* 칼럼 값으로 **LIST** 로 만든다. 이 질의는 *nation* 테이블에 대해 *name*, *capital* 값과 *host_nation* 값을 포함하는 *olympic* 테이블의 *host_city* 값에 대한 집합을 반환한다. 질의 결과에서 *name* 값이 공집합인 경우는 제외되고, *name* 과 같은 값을 갖는 *olympic* 테이블이 존재하지 않는 경우에는 공집합이 반환된다.

```

SELECT name, capital, list(SELECT host_city FROM olympic WHERE
host_nation = name) AS host_cities
FROM nation;

```

```

        name            capital      host_cities
=====
        'Somalia'       'Mogadishu'    {}

```



```
'Sri Lanka'          'Sri Jayewardenepura Kotte' {}
'Sao Tome & Principe' 'Sao Tome'          {}
...
'U.S.S.R.'           'Moscow'           {'Moscow'}
'Uruguay'            'Montevideo'        {}
'United States of America' 'Washington.D.C'    {'Atlanta ',
'St. Louis', 'Los Angeles', 'Los Angeles'}
'Uzbekistan'         'Tashkent'          {}
'Vanuatu'            'Port Vila'         {}
```

이런 형태의 다중 행 부질의 표현식은 컬렉션 타입의 값을 갖는 표현식이 허용되는 모든 곳에서 사용할 수 있다. 단, 클래스 속성 정의에서 **DEFAULT** 명세 부분과 같이 컬렉션 타입의 상수 값이 요구되는 곳에는 사용될 수 없다.

부질의 내에서 **ORDER BY** 절을 명시적으로 사용하지 않는 경우 다중 행 부질의 결과의 순서는 지정되지 않으므로, **LIST** (= **SEQUENCE**)를 생성하는 다중 행 부질의는 **ORDER BY** 절을 사용하여 결과의 순서를 지정해야 한다.

VALUES

VALUES 절은 표현식에 명시된 행 값들을 출력한다. 대부분 상수 테이블을 생성할 때 사용하지만, **VALUES** 절 자체만으로도 사용될 수 있다. **VALUES** 절에 한 개 이상의 행이 지정되면 모든 행은 같은 개수의 원소를 가져야 한다.

```
VALUES (expression[, ...])([, ...])
```

· *expression*: 괄호로 감싸인 표현식은 테이블에서의 하나의 행을 나타낸다.

VALUES 절은 상수 값으로 구성된 **UNION ALL** 질의문을 단순하게 표현하는 방법으로 볼 수 있다. 예를 들면 다음과 같은 질의문을 실행할 수 있다.

```
VALUES (1 AS col1, 'first' AS col2), (2, 'second'), (3, 'third'),
(4, 'fourth');
```

위 질의문은 다음과 같은 결과를 출력한다.

```
SELECT 1 AS col1, 'first' AS col2
UNION ALL
SELECT 2, 'second'
UNION ALL
SELECT 3, 'third'
UNION ALL
SELECT 4, 'fourth';
```

다음은 **INSERT** 문 안에서 여러 행을 갖는 **VALUES** 절을 사용하는 예이다.

```
INSERT INTO athlete (code, name, gender, nation_code, event)
VALUES ('21111', 'Jang Mi-Ran ', 'F', 'KOR', 'Weight-lifting'),
       ('21112', 'Son Yeon-Jae ', 'F', 'KOR', 'Rhythmic gymnastics');
```

다음은 **FROM** 절에서 부질의(subquery)로 사용하는 예이다.

```
SELECT a.*
FROM athlete a, (VALUES ('Jang Mi-Ran', 'F'), ('Son Yeon-Jae', 'F'))
AS t(name, gender)
WHERE a.name=t.name AND a.gender=t.gender;
```

code	name	gender	nation_code	event
21111	'Jang Mi-Ran'	'F'	'KOR'	'Weight-lifting'
21112	'Son Yeon-Jae'	'F'	'KOR'	'Rhythmic gymnastics'

FOR UPDATE

FOR UPDATE 절은 **UPDATE/DELETE** 문을 수행하기 위해 **SELECT** 문에서 반환되는 행들에 잠금을 부여하기 위해 사용될 수 있다.

```
SELECT ... [FOR UPDATE [OF <spec_name_comma_list>]]
        <spec_name_comma_list> ::= <spec_name> [, <spec_name>, ... ]
        <spec_name> ::= [schema_name.]table_name |
        [schema_name.]view_name
```

· <spec_name_comma_list>: **FROM** 절에서 참조하는 테이블/뷰들의 목록

<spec_name_comma_list>에 참조된 테이블/뷰만 잠긴다. <spec_name_comma_list>가 누락되었지만 **FOR UPDATE** 가 있는 경우 **SELECT** 질의문의 **FROM** 절에 있는 모든 테이블/뷰가 참조된다고 가정한다. 행은 **X_LOCK** 을 사용하여 잠근다.

참고

제약 사항

- 부질의 안에서 **FOR UPDATE** 절을 사용할 수 없다. 단, **FOR UPDATE** 절이 부질의를 참조할 수는 있다.

- **GROUP BY**, **DISTINCT** 또는 집계 함수를 가진 질의문에서 사용할 수 없다.
- **UNION** 을 참조할 수 없다.

다음은 **SELECT ... FOR UPDATE** 문을 사용하는 예이다.

```
CREATE TABLE t1(i INT);
INSERT INTO t1 VALUES (1), (2), (3), (4), (5);

CREATE TABLE t2(i INT);
INSERT INTO t2 VALUES (1), (2), (3), (4), (5);
CREATE INDEX idx_t2_i ON t2(i);

CREATE VIEW v12 AS SELECT t1.i AS i1, t2.i AS i2 FROM t1 INNER JOIN
t2 ON t1.i=t2.i;

SELECT * FROM t1 ORDER BY 1 FOR UPDATE;
SELECT * FROM t1 ORDER BY 1 FOR UPDATE OF t1;
SELECT * FROM t1 INNER JOIN t2 ON t1.i=t2.i ORDER BY 1 FOR UPDATE OF
t1, t2;

SELECT * FROM t1 INNER JOIN (SELECT * FROM t2 WHERE t2.i > 0) r ON
t1.i=r.i WHERE t1.i > 0 ORDER BY 1 FOR UPDATE;

SELECT * FROM v12 ORDER BY 1 FOR UPDATE;
SELECT * FROM t1, (SELECT * FROM v12, t2 WHERE t2.i > 0 AND
t2.i=v12.i1) r WHERE t1.i > 0 AND t1.i=r.i ORDER BY 1 FOR UPDATE OF
r;
```

계층적 질의

계층적 질의란 테이블에 포함된 행(row)간에 수직적 계층 관계가 성립되는 데이터에 대하여 계층 관계에 따라 각 행을 출력하는 질의이다. **START WITH ... CONNECT BY** 절은 **SELECT** 구문과 결합하여 사용된다.

CONNECT BY ... START WITH 로 두 절의 순서를 바꿔서 사용할 수도 있다.

```
SELECT column_list
FROM table_joins | tables
[WHERE join_conditions and/or filtering_conditions]
[hierarchical_clause]

hierarchical_clause :
    [START WITH condition] CONNECT BY [NOCYCLE] condition
    | CONNECT BY [NOCYCLE] condition [START WITH condition]
```

START WITH 절

START WITH 절은 계층 관계가 시작되는 루트 행(root row)을 지정하기 위한 것으로, **START WITH** 절 다음에 계층 관계를 검색하기 위한 조건식을 포함한다. 만약, **START WITH** 절 다음에 위치하는 조건식이 생략되면 대상 테이블 내에 존재하는 모든 행을 루트 행으로 간주하여 계층 관계를 검색할 것이다.

참고

START WITH 절이 생략되거나, **START WITH** 조건식을 만족하는 결과 행이 존재하지 않는 경우, 테이블 내의 모든 행을 루트 행으로 간주하여 각 루트 행에 속하는 하위 자식 행들 간 계층 관계를 검색하므로 결과 행들 중 일부는 중복되어 출력될 수 있다.

CONNECT BY 절

- **PRIOR** : **CONNECT BY** 조건식은 한 쌍의 행에 대한 상-하 계층 관계(부모-자식 관계)를 정의하기 위한 것으로, 조건식 내에서 하나는 부모(parent)로 지정되고, 다른 하나는 자식(child)으로 지정된다. 이처럼 행 간의 부모-자식 간 계층 관계를 정의하기 위하여 **CONNECT BY** 조건식 내에 **PRIOR** 연산자를 이용하여 부모 행의 칼럼 값을 지정한다. 즉, 부모 행의 칼럼 값과 같은 칼럼 값을 가지는 모든 행은 자식 행이 된다.
- **NOCYCLE** : **CONNECT BY** 절의 조건식에 따른 계층 질의 결과는 루프를 포함할 수 있으며, 이것은 계층 트리를 생성할 때 무한 루프를 발생시키는 원인이 될 수 있다. 따라서, CUBRID는 루프를 발견하면 기본적으로 오류를 반환하고, 특수 연산자인 **NOCYCLE** 이 **CONNECT BY** 절에 명시된 경우에는 오류를 발생시키지 않고 해당 루프에 의해 검색된 결과를 출력한다.

만약, **CONNECT BY** 절에서 **NOCYCLE** 이 명시되지 않은 계층 질의문을 수행 중에 루프가 감지되는 경우, CUBRID는 오류를 반환하고 해당 질의문을 취소한다. 반면, **NOCYCLE** 이 명시된 계층 질의문에서 루프가 감지되는 경우, CUBRID는 오류를 반환하지는 않지만 루프가 감지된 행에 대해 **CONNECT_BY_ISCYCLE** 값을 1로 설정하고, 더 이상 계층 트리의 검색을 확장하지 않을 것이다.

다음은 계층 질의문을 수행하는 예제이다.

```
-- Creating tree table and then inserting data
CREATE TABLE tree(ID INT, MgrID INT, Name VARCHAR(32), BirthYear
INT);

INSERT INTO tree VALUES (1,NULL,'Kim', 1963);
INSERT INTO tree VALUES (2,NULL,'Moy', 1958);
INSERT INTO tree VALUES (3,1,'Jonas', 1976);
INSERT INTO tree VALUES (4,1,'Smith', 1974);
INSERT INTO tree VALUES (5,2,'Verma', 1973);
INSERT INTO tree VALUES (6,2,'Foster', 1972);
INSERT INTO tree VALUES (7,6,'Brown', 1981);
```

```
-- Executing a hierarchical query with CONNECT BY clause
SELECT id, mgrid, name
FROM tree
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name
1	NULL	'Kim'
2	NULL	'Moy'
3	1	'Jonas'
3	1	'Jonas'
4	1	'Smith'
4	1	'Smith'
5	2	'Verma'
5	2	'Verma'
6	2	'Foster'
6	2	'Foster'
7	6	'Brown'
7	6	'Brown'
7	6	'Brown'

```
-- Executing a hierarchical query with START WITH clause
SELECT id, mgrid, name
FROM tree
START WITH mgrid IS NULL
CONNECT BY prior id=mgrid
ORDER BY id;
```

id	mgrid	name
1	NULL	'Kim'
2	NULL	'Moy'
3	1	'Jonas'
4	1	'Smith'
5	2	'Verma'
6	2	'Foster'
7	6	'Brown'

계층 질의 실행

조인 테이블에 대한 계층 질의

SELECT 문에서 대상 테이블이 조인된 경우, **WHERE** 절에는 검색 조건식 외에 테이블 조인 조건을 포함할 수 있다. 이때, CUBRID는 제일 먼저 **WHERE** 절의 조인 조건을 적용하여 테이블 조인 연산을 수행한 후, **CONNECT BY** 절의 조건식을 적용하고, 마지막으로 **WHERE** 절 내의 나머지 검색 조건식을 적용하여 연산을 처리한다.

WHERE 절 내에 조인 조건식과 검색 조건식을 함께 명시하는 경우, 내부적으로 조인 조건식이 검색 조건식으로 분류되어 의도하지 않게 연산 순서가 달라질 수 있으므로, **WHERE** 절보다는 **FROM** 절 내에 테이블 조인 조건을 명시하는 것을 권장한다.

계층 질의 결과

조인 테이블에 대한 계층 질의 결과는 **START WITH** 절의 조건식에 따라 루트 행으로부터 출력된다. 만약 **START WITH** 절이 생략되면 조인된 테이블의 모든 행들을 루트 행으로 간주하여 계층 관계를 출력한다. 이를 위해 CUBRID는 하나의 루트 행에 대하여 모든 자식 행을 검색한 후, 각 자식 행 하위에 속하는 모든 자식 행을 재귀적으로 검색한다. 이러한 검색은 더 이상의 자식 행이 발견되지 않을 때까지 반복된다.

또한, 계층 질의문은 **CONNECT BY** 절의 조건식을 먼저 적용하여 결과 행들을 검색한 후, **WHERE** 절에 명시된 검색 조건식을 적용하여 최종 결과 행들을 출력한다.

다음은 두 개의 조인된 테이블에 대하여 계층 질의문을 수행하는 예제이다.

```
-- Creating tree2 table and then inserting data
CREATE TABLE tree2(id int, treeid int, job varchar(32));

INSERT INTO tree2 VALUES(1,1,'Partner');
INSERT INTO tree2 VALUES(2,2,'Partner');
INSERT INTO tree2 VALUES(3,3,'Developer');
INSERT INTO tree2 VALUES(4,4,'Developer');
INSERT INTO tree2 VALUES(5,5,'Sales Exec. ');
INSERT INTO tree2 VALUES(6,6,'Sales Exec. ');
INSERT INTO tree2 VALUES(7,7,'Assistant');
INSERT INTO tree2 VALUES(8,null,'Secretary');

-- Executing a hierarchical query onto table joins
SELECT t.id,t.name,t2.job,level
FROM tree t INNER JOIN tree2 t2 ON t.id=t2.treeid
START WITH t.mgrid is null
CONNECT BY prior t.id=t.mgrid
ORDER BY t.id;
```

id	name	job	level
=====			
1	'Kim'	'Partner'	1
2	'Moy'	'Partner'	1
3	'Jonas'	'Developer'	2
4	'Smith'	'Developer'	2
5	'Verma'	'Sales Exec.'	2
6	'Foster'	'Sales Exec.'	2
7	'Brown'	'Assistant'	3

계층 질의문에서의 데이터 정렬

ORDER SIBLINGS BY 절은 계층 질의 결과 값들의 계층 정보를 유지하면서 특정 칼럼을 기준으로 오름차순 또는 내림차순으로 데이터를 정렬하며, 동일한 부모를 가진 자식 행들을 정렬할 수 있다. 계층적 질의문에서 데이터의 계층적 순서를 파악하기 위해 사용한다.

```
ORDER SIBLINGS BY col_1 [ASC|DESC] [, col_2 [ASC|DESC] [...[, col_n
[ASC|DESC]]...]]
```

다음은 상사와 그의 부하 직원을 출력하되, 출생 연도가 앞서는 사람부터 출력하는 예제이다.

계층 질의 결과는 기본적으로 **ORDER SIBLINGS BY** 절에 명시된 칼럼 리스트에 따라 정렬된 부모와 그 부모의 자식 노드들이 연속으로 출력된다. 부모가 같은 형제 노드는 명시된 정렬 순서에 따라 정렬되어 출력된다.

```
-- Outputting a parent node and its child nodes, which sibling nodes
that share the same parent are sorted in the order of birthyear.
SELECT id, mgrid, name, birthyear, level
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER SIBLINGS BY birthyear;
```

id	mgrid	name	birthyear	level
=====				
2	NULL	'Moy'	1958	1
6	2	'Foster'	1972	2
7	6	'Brown'	1981	3
5	2	'Verma'	1973	2
1	NULL	'Kim'	1963	1
4	1	'Smith'	1974	2
3	1	'Jonas'	1976	2

다음은 상사와 그의 부하 직원을 출력하되, 같은 레벨 간에는 우선 입사한 순서로 정렬시키는 예제이다. *id* 는 입사한 순서로 부여된다. *id* 는 직원의 입사번호이며, *mgrid* 는 상사의 입사번호이다.

```
-- Outputting a parent node and its child nodes, which sibling nodes
that share the same parent are sorted in the order of id.
SELECT id, mgrid, name, LEVEL
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER SIBLINGS BY id;
```

id	mgrid	name	level
1	NULL	'Kim'	1
3	1	'Jonas'	2
4	1	'Smith'	2
2	NULL	'Moy'	1
5	2	'Verma'	2
6	2	'Foster'	2
7	6	'Brown'	3

계층 질의 의사 칼럼

LEVEL

LEVEL 은 계층 질의 결과 행의 깊이 레벨(depth)을 나타내는 의사 칼럼(pseudocolumn)이다. 루트 노드의 **LEVEL** 은 1이며, 하위 자식 노드의 **LEVEL** 은 2가 된다.

LEVEL 의사 칼럼은 **SELECT** 문 내의 **WHERE** 절, **ORDER BY** 절, **GROUP BY ... HAVING** 절, **CONNECT BY** 절에서 사용 가능하며, 집계 함수를 이용하는 구문에서도 사용 가능하다.

다음은 노드의 레벨을 확인하기 위하여 **LEVEL** 값을 조회하는 예제이다.

```
-- Checking the LEVEL value
SELECT id, mgrid, name, LEVEL
FROM tree
WHERE LEVEL=2
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name	level
3	1	'Jonas'	2

4	1	'Smith'	2
5	2	'Verma'	2
6	2	'Foster'	2

다음은 **CONNECT BY** 절 뒤에 **LEVEL** 조건을 추가한 예제이다.

```
SELECT LEVEL FROM db_root CONNECT BY LEVEL <= 10;
```

```

level
=====
      1
      2
      3
      4
      5
      6
      7
      8
      9
     10

```

단, “CONNECT BY expr(LEVEL) < expr”과 같은 형태, 예를 들어 “CONNECT BY LEVEL +1 < 5”와 같은 형태는 지원하지 않는다.

CONNECT_BY_ISLEAF

CONNECT_BY_ISLEAF 는 계층 질의 결과 행이 리프 노드(leaf node : 하위에 자식 노드를 가지지 않는 단말 노드)인지 가리키는 의사 칼럼이다. 계층 구조 하에서 현재 행이 리프 노드이면 1을 반환하고, 그렇지 않으면 0을 반환한다.

다음은 리프 노드를 확인하기 위하여 **CONNECT_BY_ISLEAF** 값을 조회하는 예제이다.

```

-- CONNECT_BY_ISLEAF의 값을 확인하기
SELECT id, mgrid, name, CONNECT_BY_ISLEAF
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;

```

id	mgrid	name	connect_by_isleaf
1	NULL	'Kim'	0
2	NULL	'Moy'	0
3	1	'Jonas'	1
4	1	'Smith'	1

5	2	'Verma'	1
6	2	'Foster'	0
7	6	'Brown'	1

CONNECT_BY_ISCYCLE

CONNECT_BY_ISCYCLE 은 계층 질의 결과 행이 루프를 발생시키는 행인지를 가리키는 의사 칼럼이다. 즉, 현재 행의 자식이 동시에 조상이 되어 루프를 발생시키는 경우 1을 반환하고, 그렇지 않으면 0을 반환한다.

CONNECT_BY_ISCYCLE 의사 칼럼은 **SELECT** 문 내의 **WHERE** 절, **ORDER BY** 절, **GROUP BY ... HAVING** 절에서 사용할 수 있으며, 집계 함수를 이용하는 구문에서도 사용 가능하다.

참고

CONNECT_BY_ISCYCLE 은 **CONNECT BY** 절에 **NOCYCLE** 키워드가 명시되는 경우에만 사용할 수 있다.

다음은 루프를 발생시키는 행을 확인하기 위해 **CONNECT_BY_ISCYCLE** 값을 조회하는 예제이다.

```
-- tree_cycle 테이블을 만들고 데이터를 삽입하기
CREATE TABLE tree_cycle(ID INT, MgrID INT, Name VARCHAR(32));

INSERT INTO tree_cycle VALUES (1,NULL,'Kim');
INSERT INTO tree_cycle VALUES (2,11,'Moy');
INSERT INTO tree_cycle VALUES (3,1,'Jonas');
INSERT INTO tree_cycle VALUES (4,1,'Smith');
INSERT INTO tree_cycle VALUES (5,3,'Verma');
INSERT INTO tree_cycle VALUES (6,3,'Foster');
INSERT INTO tree_cycle VALUES (7,4,'Brown');
INSERT INTO tree_cycle VALUES (8,4,'Lin');
INSERT INTO tree_cycle VALUES (9,2,'Edwin');
INSERT INTO tree_cycle VALUES (10,9,'Audrey');
INSERT INTO tree_cycle VALUES (11,10,'Stone');

-- CONNECT_BY_ISCYCLE의 값을 확인하기
SELECT id, mgrid, name, CONNECT_BY_ISCYCLE
FROM tree_cycle
START WITH name IN ('Kim', 'Moy')
CONNECT BY NOCYCLE PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name	connect_by_iscycle
=====			

1	NULL	'Kim'	0
2	11	'Moy'	0
3	1	'Jonas'	0
4	1	'Smith'	0
5	3	'Verma'	0
6	3	'Foster'	0
7	4	'Brown'	0
8	4	'Lin'	0
9	2	'Edwin'	0
10	9	'Audrey'	0
11	10	'Stone'	1

계층 질의 연산자

CONNECT_BY_ROOT

CONNECT_BY_ROOT 은 칼럼 값으로 루트 행의 값을 반환한다. 이 연산자는 **SELECT** 문 내의 **WHERE** 절 및 **ORDER BY** 절에서 사용할 수 있지만, **WHERE** 절은 반드시 **START WITH / CONNECT BY** 절보다 앞에 위치해야 하고, **ORDER BY** 는 계층절 다음에 위치하여야 한다.

다음은 계층 질의 결과 행에 대하여 루트 행의 *id* 값을 조회하는 예제이다.

```
-- 각 행마다 루트 행의 id 값을 확인하기
SELECT id, mgrid, name, CONNECT_BY_ROOT id
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name	connect_by_root id
1	NULL	'Kim'	1
2	NULL	'Moy'	2
3	1	'Jonas'	1
4	1	'Smith'	1
5	2	'Verma'	2
6	2	'Foster'	2
7	6	'Brown'	2

```
-- 올바른 예
SELECT id, CONNECT_BY_ROOT name
FROM tree
WHERE CONNECT_BY_ROOT name = 'Kim'
START WITH mgrid IS NULL
CONNECT BY PRIOR id = mgrid
```

```
-- 잘못된 예
SELECT id, CONNECT_BY_ROOT name
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id = mgrid
WHERE CONNECT_BY_ROOT name = 'Kim'; -- 문법 오류
```

PRIOR

PRIOR 연산자는 칼럼 값으로 부모 행의 값을 반환하고, 루트 행에 대해서는 **NULL** 을 반환한다. 이 연산자는 **SELECT** 문 내의 **WHERE** 절, **ORDER BY** 절 및 **CONNECT BY** 절에서 사용할 수 있다.

다음은 계층 질의 결과 행에 대하여 부모 행의 id 값을 조회하는 예제이다.

```
-- 각 행마다 부모 행의 id 값을 확인하기
SELECT id, mgrid, name, PRIOR id as "prior_id"
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name	prior_id
1	NULL	'Kim'	NULL
2	NULL	'Moy'	NULL
3	1	'Jonas'	1
4	1	'Smith'	1
5	2	'Verma'	2
6	2	'Foster'	2
7	6	'Brown'	6

계층 질의 함수

SYS_CONNECT_BY_PATH

SYS_CONNECT_BY_PATH 함수는 루트 행으로부터 해당 행까지의 상-하 관계의 path를 문자열로 반환하는 함수이다. 이때, 함수의 인자로 지정되는 칼럼과 구분자는 문자형 타입이어야 하며, 각 path는 지정된 구분자에 의해 구분되어 연쇄적으로 출력된다. 이 함수는 **SELECT** 문 내의 **WHERE** 절과 **ORDER BY** 절에서 사용할 수 있다.

```
SYS_CONNECT_BY_PATH (column_name, separator_char)
```

다음은 루트 행으로부터 해당 행의 path를 확인하는 예제이다.

```
-- 구분자를 이용하여 루트 행으로부터 해당 행까지 path를 확인하기
SELECT id, mgrid, name, SYS_CONNECT_BY_PATH(name, '/') as [hierarchy]
FROM tree
START WITH mgrid IS NULL
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

id	mgrid	name	hierarchy
1	NULL	'Kim'	'/Kim'
2	NULL	'Moy'	'/Moy'
3	1	'Jonas'	'/Kim/Jonas'
4	1	'Smith'	'/Kim/Smith'
5	2	'Verma'	'/Moy/Verma'
6	2	'Foster'	'/Moy/Foster'
7	6	'Brown'	'/Moy/Foster/Brown'

계층 질의문 예

SELECT 문에 **CONNECT BY** 절을 명시하여 계층 질의문을 작성하는 예이다.

재귀적 참조 관계를 가지는 테이블을 생성했으며, 이 테이블은 *ID* 와 *ParentID* 라는 두 개의 칼럼으로 구성되고, *ID* 와 *ParentID* 는 각각 기본 키와 외래 키로 정의된다고 가정한다. 이때, 루트 노드의 *ParentID* 값은 **NULL** 이 된다.

테이블이 생성되었다면, 아래와 같이 **UNION ALL** 을 이용하여 계층 구조를 가지는 전체 데이터와 **LEVEL** 값을 조회할 수 있다.

```
CREATE TABLE tree_table (ID int PRIMARY KEY, ParentID int, name
VARCHAR(128));
```

```
INSERT INTO tree_table VALUES (1, NULL, 'Kim');
INSERT INTO tree_table VALUES (2, 1, 'Moy');
INSERT INTO tree_table VALUES (3, 1, 'Jonas');
INSERT INTO tree_table VALUES (4, 1, 'Smith');
INSERT INTO tree_table VALUES (5, 3, 'Verma');
INSERT INTO tree_table VALUES (6, 3, 'Foster');
INSERT INTO tree_table VALUES (7, 4, 'Brown');
INSERT INTO tree_table VALUES (8, 4, 'Lin');
INSERT INTO tree_table VALUES (9, 2, 'Edwin');
INSERT INTO tree_table VALUES (10, 9, 'Audrey');
INSERT INTO tree_table VALUES (11, 10, 'Stone');
```

```
SELECT L1.ID, L1.ParentID, L1.name, 1 AS [Level]
FROM tree_table AS L1
```

```

WHERE L1.ParentID IS NULL
UNION ALL
SELECT L2.ID, L2.ParentID, L2.name, 2 AS [Level]
FROM tree_table AS L1
INNER JOIN tree_table AS L2 ON L1.ID=L2.ParentID
WHERE L1.ParentID IS NULL
UNION ALL
SELECT L3.ID, L3.ParentID, L3.name, 3 AS [Level]
FROM tree_table AS L1
INNER JOIN tree_table AS L2 ON L1.ID=L2.ParentID
INNER JOIN tree_table AS L3 ON L2.ID=L3.ParentID
WHERE L1.ParentID IS NULL
UNION ALL
SELECT L4.ID, L4.ParentID, L4.name, 4 AS [Level]
FROM tree_table AS L1
INNER JOIN tree_table AS L2 ON L1.ID=L2.ParentID
INNER JOIN tree_table AS L3 ON L2.ID=L3.ParentID
INNER JOIN tree_table AS L4 ON L3.ID=L4.ParentID
WHERE L1.ParentID IS NULL;

```

ID	ParentID	name	Level
1	NULL	'Kim'	1
2	1	'Moy'	2
3	1	'Jonas'	2
4	1	'Smith'	2
9	2	'Edwin'	3
5	3	'Verma'	3
6	3	'Foster'	3
7	4	'Brown'	3
8	4	'Lin'	3
10	9	'Audrey'	4

계층 관계를 가지는 데이터의 레벨이 얼마나 될지 예측할 수 없으므로, 위 질의문은 새로운 행이 검색되지 않을 때까지 루프를 도는 저장 프로시저(stored procedure) 문으로 재작성할 수 있다.

그러나 루프를 도는 동안 각 단계마다 계층 트리를 확인해야 하므로, 아래와 같이 **SELECT** 문에 **CONNECT BY** 절을 명시하여 계층 질의문을 재작성할 수 있다. 다음의 질의문을 실행하면, 계층 관계를 가지는 데이터 전체와 각 행의 레벨이 출력된다.

```

SELECT ID, ParentID, name, Level
FROM tree_table
START WITH ParentID IS NULL
CONNECT BY ParentID=PRIOR ID;

```

ID	ParentID	name	level
=====			

1	NULL	'Kim'	1
2	1	'Moy'	2
9	2	'Edwin'	3
10	9	'Audrey'	4
11	10	'Stone'	5
3	1	'Jonas'	2
5	3	'Verma'	3
6	3	'Foster'	3
4	1	'Smith'	2
7	4	'Brown'	3
8	4	'Lin'	3

루프로 인한 오류를 발생시키지 않으려면 다음과 같이 **NOCYCLE**을 명시할 수 있다. 아래의 질의 수행 시 루프가 발생하지 않으므로, 결과는 위와 동일하다.

```
SELECT ID, ParentID, name, Level
FROM tree_table
START WITH ParentID IS NULL
CONNECT BY NOCYCLE ParentID=PRIOR ID;
```

계층 질의에 대한 루프 탐색 과정 중에 동일한 행이 발견되면, CUBRID는 그 질의를 루프가 있는 것으로 판단한다. 다음은 루프가 존재하는 예로, **NOCYCLE**을 명시하여 루프가 존재하는 경우 추가 탐색을 종료하도록 했다.

```
CREATE TABLE tbl(seq INT, id VARCHAR(10), parent VARCHAR(10));

INSERT INTO tbl VALUES (1, 'a', null);
INSERT INTO tbl VALUES (2, 'b', 'a');
INSERT INTO tbl VALUES (3, 'b', 'c');
INSERT INTO tbl VALUES (4, 'c', 'b');
INSERT INTO tbl VALUES (5, 'c', 'b');

SELECT seq, id, parent, LEVEL,
       CONNECT_BY_ISCYCLE AS iscycle,
       CAST(SYS_CONNECT_BY_PATH(id, '/') AS VARCHAR(10)) AS idpath
FROM tbl
START WITH PARENT is NULL
CONNECT BY NOCYCLE PARENT = PRIOR id;
```

seq	id	parent	level	iscycle	idpath
1	'a'	NULL	1	0	'/a'
2	'b'	'a'	2	0	'/a/b'
4	'c'	'b'	3	0	'/a/b/c'
3	'b'	'c'	4	1	'/a/b/c/b'
5	'c'	'b'	5	1	'/a/b/c/b/c'

5	'c'	'b'	3	0	'/a/b/c'
3	'b'	'c'	4	1	'/a/b/c/b'
4	'c'	'b'	5	1	'/a/b/c/b/c'

다음은 계층 질의를 사용하여 2013년 3월(201303)의 날짜들을 출력하는 예제이다.

```
SELECT TO_CHAR(base_month + lvl -1, 'YYYYMMDD') h_date
FROM (
  SELECT LEVEL lvl, base_month
  FROM (
    SELECT TO_DATE('201303', 'YYYYMM') base_month FROM
db_root
  )
  CONNECT BY LEVEL <= LAST_DAY(base_month) - base_month + 1
);
```

```
h_date
=====
'20130301'
'20130302'
'20130303'
'20130304'
'20130305'
'20130306'
'20130307'
'20130308'
'20130309'
'20130310'
'20130311'
'20130312'
'20130313'
'20130314'
'20130315'
'20130316'
'20130317'
'20130318'
'20130319'
'20130320'
'20130321'
'20130322'
'20130323'
'20130324'
'20130325'
'20130326'
'20130327'
'20130328'
'20130329'
'20130330'
'20130331'
```



```
31 rows selected. (0.066175 sec) Committed.
```

계층 질의문의 성능

단일 테이블 계층 질의

단일 테이블 계층 질의는 **CONNECT BY** 질의 자식에 해당하는 컬럼에 인덱스를 추가하면 성능이 향상된다. 아래는 *mgrid* 컬럼에 인덱스를 추가한 예제이다.

```
-- Creating tree table and then inserting data
CREATE TABLE tree(ID INT, MgrID INT, Name VARCHAR(32), BirthYear
INT);

INSERT INTO tree VALUES (1,NULL,'Kim', 1963);
INSERT INTO tree VALUES (2,NULL,'Moy', 1958);
INSERT INTO tree VALUES (3,1,'Jonas', 1976);
INSERT INTO tree VALUES (4,1,'Smith', 1974);
INSERT INTO tree VALUES (5,2,'Verma', 1973);
INSERT INTO tree VALUES (6,2,'Foster', 1972);
INSERT INTO tree VALUES (7,6,'Brown', 1981);

-- Copy data
INSERT INTO tree select * from tree;
INSERT INTO tree select * from tree;
INSERT INTO tree select * from tree;
INSERT INTO tree select * from tree;
INSERT INTO tree select * from tree;

-- Executing a single table hierarchical query
SELECT count(*)
FROM tree
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

```
Trace Statistics:
  SELECT (time: 1686, fetch: 155971, fetch_time: 11, ioread: 0)
    SCAN (table: dba.tree), (heap time: 1, fetch: 1, ioread: 0,
readrows: 224, rows: 224)
    CONNECTBY (time: 1685, fetch: 155970, fetch_time: 0, ioread: 0)
      SCAN (table: dba.tree), (heap time: 1607, fetch: 38112,
ioread: 0, readrows: 8537088, rows: 37888)
        ORDERBY (time: 11, sort: true, page: 105, ioread: 0)
```

```
-- Create index
CREATE INDEX idx ON tree(mgrid);

-- Executing a single table hierarchical query with index
SELECT count(*)
FROM tree
CONNECT BY PRIOR id=mgrid
ORDER BY id;
```

```
Trace Statistics:
  SELECT (time: 128, fetch: 195206, fetch_time: 9, ioread: 0)
    SCAN (table: dba.tree), (heap time: 0, fetch: 1, ioread: 0,
readrows: 224, rows: 224)
      CONNECTBY (time: 128, fetch: 195203, fetch_time: 0, ioread: 0)
        SCAN (index: dba.tree.idx), (btree time: 35, fetch: 77344,
ioread: 0, readkeys: 1120, filteredkeys: 0, rows: 37888) (lookup
time: 10, rows: 37888)
          ORDERBY (time: 11, sort: true, page: 105, ioread: 0)
```

조인 테이블 계층 질의

조인 테이블에 대한 계층 질의는 테이블 조인 연산을 수행한 후, **CONNECT BY** 절의 조건식을 적용할 때 해시 리스트 스캔을 사용하여 최적화 한다. 아래는 해시 리스트 스캔과 일반 스캔을 사용할 때의 예시이다.

```
-- Creating tree2 table and then inserting data
CREATE TABLE tree2(id int, treeid int, job varchar(32));

INSERT INTO tree2 VALUES(1,1,'Partner');
INSERT INTO tree2 VALUES(2,2,'Partner');
INSERT INTO tree2 VALUES(3,3,'Developer');
INSERT INTO tree2 VALUES(4,4,'Developer');
INSERT INTO tree2 VALUES(5,5,'Sales Exec. ');
INSERT INTO tree2 VALUES(6,6,'Sales Exec. ');
INSERT INTO tree2 VALUES(7,7,'Assistant');
INSERT INTO tree2 VALUES(8,null,'Secretary');

-- Copy data
INSERT INTO tree2 select * from tree2;

-- Executing a hierarchical query onto table joins
SELECT count(*)
FROM tree t INNER JOIN tree2 t2 ON t.id=t2.treeid
START WITH t.mgrid is null
CONNECT BY prior t.id=t.mgrid
ORDER BY t.id;
```

```
Trace Statistics:
  SELECT (time: 663, fetch: 877290, fetch_time: 54, ioread: 0)
    SCAN (table: dba.tree), (heap time: 0, fetch: 225, ioread: 0,
readrows: 224, rows: 224)
    SCAN (table: dba.tree2), (heap time: 0, fetch: 224, ioread: 0,
readrows: 3584, rows: 448)
  CONNECTBY (time: 662, fetch: 876841, fetch_time: 0, ioread: 0)
    SCAN (hash temp(m), build time: 0, time: 53, fetch: 0, ioread:
0, readrows: 278976, rows: 278528)
    ORDERBY (time: 92, sort: true, page: 1560, ioread: 0)
```

```
-- Executing a hierarchical query onto table joins without hash list
scan
SELECT /*+ NO_HASH_LIST_SCAN */ count(*)
FROM tree t INNER JOIN tree2 t2 ON t.id=t2.treeid
START WITH t.mgrid is null
CONNECT BY prior t.id=t.mgrid
ORDER BY t.id;
```

```
Trace Statistics:
  SELECT (time: 8977, fetch: 877335, fetch_time: 57, ioread: 0)
    SCAN (table: dba.tree), (heap time: 0, fetch: 225, ioread: 0,
readrows: 224, rows: 224)
    SCAN (table: dba.tree2), (heap time: 0, fetch: 224, ioread: 0,
readrows: 3584, rows: 448)
  CONNECTBY (time: 8977, fetch: 876886, fetch_time: 0, ioread: 0)
    SCAN (temp time: 8353, fetch: 0, ioread: 0, readrows:
124837888, rows: 278528)
    ORDERBY (time: 91, sort: true, page: 1560, ioread: 0)
```

참고

조인 테이블 계층형 질의에 대한 해시 리스트 스캔 최적화는 CUBRID 11.0 부터 지원된다.

INSERT

INSERT 문을 사용하여 데이터베이스에 존재하는 테이블에 새로운 레코드를 삽입할 수 있다. CUBRID 는 **INSERT ... VALUES** 문, **INSERT ... SET** 문, **INSERT ... SELECT** 문을 지원한다.

INSERT ... VALUES 문과 **INSERT ... SET** 문은 명시적으로 지정된 값을 기반으로 새로운 레코드를 삽입 하며, **INSERT ... SELECT** 문은 다른 테이블에서 조회한 결과 레코드를 삽입할 수 있다. 단일 **INSERT** 문

을 이용하여 여러 행을 삽입하기 위해서는 **INSERT ... VALUES** 문 또는 **INSERT ... SELECT** 문을 사용한다.

```
<INSERT ... VALUES statement>
INSERT [INTO] <table_specification> [(column_name, ...)]
    {VALUES | VALUE}({expr | DEFAULT}, ...)[,({expr |
DEFAULT}, ...),...]
    [ON DUPLICATE KEY UPDATE column_name = expr, ... ]
INSERT [INTO] <table_specification> DEFAULT [ VALUES ]

<INSERT ... SET statement>
INSERT [INTO] <table_specification>
    SET column_name = {expr | DEFAULT}[, column_name = {expr |
DEFAULT},...]
    [ON DUPLICATE KEY UPDATE column_name = expr, ... ]

<INSERT ... SELECT statement>
INSERT [INTO] <table_specification> [(column_name, ...)]
    SELECT...
    [ON DUPLICATE KEY UPDATE column_name = expr, ... ]

<table_specification> ::= [schema_name.]table_name |
<remote_table_spec>
<remote_table_spec> ::=
[schema_name.]table_name@[schema_name.]server_name
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 새로운 레코드를 삽입할 대상 테이블 이름을 지정한다.
- *server_name*: dblink로 연결할 원격 서버 이름을 지정한다.
- *column_name*: 값을 삽입할 칼럼 이름을 지정한다. 이 값을 생략하면, 테이블에 정의된 모든 칼럼이 명시된 것으로 간주되므로 모든 칼럼에 대한 값을 **VALUES** 뒤에 명시해야 한다. 테이블에 정의된 칼럼 중 일부 칼럼만 명시하면 나머지 칼럼에는 **DEFAULT** 로 정의된 값이 할당되며, 정의된 기본값이 없는 경우 **NULL** 값이 할당된다.
- *expr | DEFAULT*: **VALUES** 뒤에는 칼럼에 대응하는 칼럼 값을 명시하며, 표현식 또는 **DEFAULT** 키워드를 값으로 지정할 수 있다. 명시된 칼럼 리스트의 순서와 개수는 칼럼 값 리스트와 대응되어야 하며, 하나의 레코드에 대해 칼럼 값 리스트는 괄호로 처리된다.
- **DEFAULT**: 기본값을 칼럼 값으로 명시하기 위하여 **DEFAULT** 키워드를 사용할 수 있다. **VALUES** 키워드 뒤의 칼럼 값 리스트 내에 **DEFAULT** 를 명시하면 해당 칼럼에 기본값을 저장하고, **VALUES** 키워드 앞에 **DEFAULT** 를 명시하면 테이블 내 모든 칼럼에 대해 기본값을 저장한다. 기본값이 정의되지 않은 칼럼에 대해서는 **NULL** 을 저장한다.

- **ON DUPLICATE KEY UPDATE: PRIMARY KEY** 또는 **UNIQUE** 속성이 정의된 칼럼에 중복 값이 삽입되어 제약 조건 위반이 발생하면, **ON DUPLICATE KEY UPDATE** 절에 명시된 액션을 수행하면서 제약 조건 위반을 발생시킨 값을 특정 값으로 변경한다.

```
CREATE TABLE a_tbl1(
    id INT UNIQUE,
    name VARCHAR,
    phone VARCHAR DEFAULT '000-0000'
);

--insert default values with DEFAULT keyword before VALUES
INSERT INTO a_tbl1 DEFAULT VALUES;

--insert multiple rows
INSERT INTO a_tbl1 VALUES (1,'aaa', DEFAULT),(2,'bbb', DEFAULT);

--insert a single row specifying column values for all
INSERT INTO a_tbl1 VALUES (3,'ccc', '333-3333');

--insert two rows specifying column values for only
INSERT INTO a_tbl1(id) VALUES (4), (5);

--insert a single row with SET clauses
INSERT INTO a_tbl1 SET id=6, name='eee';
INSERT INTO a_tbl1 SET id=7, phone='777-7777';

SELECT * FROM a_tbl1;
```

id	name	phone
NULL	NULL	'000-0000'
1	'aaa'	'000-0000'
2	'bbb'	'000-0000'
3	'ccc'	'333-3333'
4	NULL	'000-0000'
5	NULL	'000-0000'
6	'eee'	'000-0000'
7	NULL	'777-7777'

```
INSERT INTO a_tbl1 SET id=6, phone='000-0000'
ON DUPLICATE KEY UPDATE phone='666-6666';
SELECT * FROM a_tbl1 WHERE id=6;
```

id	name	phone
6	'eee'	'666-6666'

```
INSERT INTO a_tbl1 SELECT * FROM a_tbl1 WHERE id=7 ON DUPLICATE KEY
UPDATE name='ggg';
SELECT * FROM a_tbl1 WHERE id=7;
```

id	name	phone
7	'ggg'	'777-7777'

INSERT ... SET 문에서 할당 표현식에 대한 평가는 왼쪽에서 오른쪽으로 수행된다. 칼럼 값이 정해지지 않았으면 기본값을 할당하고, 기본값이 없으면 **NULL**을 할당한다.

```
CREATE TABLE tbl (a INT, b INT, c INT);
INSERT INTO tbl SET a=1, b=a+1, c=b+2;
SELECT * FROM tbl;
```

a	b	c
1	2	4

위의 예에서 칼럼 b의 값을 할당할 때, a의 값이 1이므로 b는 2, c는 4가 된다.

```
CREATE TABLE tbl2 (a INT, b INT, c INT);
INSERT INTO tbl2 SET a=b+1, b=1, c=b+2;
```

위의 예에서 칼럼 a의 값을 할당할 때, b의 값이 아직 정해지지 않았으며 b의 기본값이 없으므로 a의 값은 **NULL**이 된다.

```
SELECT * FROM tbl2;
```

a	b	c
NULL	1	3

```
CREATE TABLE tbl3 (a INT, b INT default 10, c INT);
INSERT INTO tbl3 SET a=b+1, b=1, c=b+2;
```

위의 예에서 칼럼 a의 값을 할당할 때, b의 값이 아직 정해지지 않았으며 b의 기본값이 10이므로 a의 값은 11이 된다.

```
SELECT * FROM tbl3;
```

a	b	c
11	1	3

INSERT ... SELECT 문

INSERT 문에 **SELECT** 질의를 사용하면 하나 이상의 테이블로부터 특정 검색 조건을 만족하는 질의 결과를 대상 테이블에 삽입할 수 있다.

```
INSERT [INTO] [schema_name.]table_name [(column_name, ...)]
  SELECT...
  [ON DUPLICATE KEY UPDATE column_name = expr, ... ]
```

SELECT 문은 **VALUES** 키워드 대신 사용하거나 **VALUES** 뒤의 칼럼 값 리스트 내에 부질의로서 포함될 수 있다. **VALUES** 키워드를 대신하여 **SELECT** 문을 명시하면, 질의 결과로 얻은 다수의 레코드를 한 번에 대상 테이블 칼럼에 삽입할 수 있다. 그러나, **SELECT** 문을 칼럼 값 리스트 내에 부질의로 사용하면 질의 결과 레코드가 하나여야 한다.

```
--creating an empty table which schema replicated from a_tbl1
CREATE TABLE a_tbl2 LIKE a_tbl1;

--inserting multiple rows from SELECT query results
INSERT INTO a_tbl2 SELECT * FROM a_tbl1 WHERE id IS NOT NULL;

--inserting column value with SELECT subquery specified in the value
list
INSERT INTO a_tbl2 VALUES(8, SELECT name FROM a_tbl1 WHERE name
<'bbb', DEFAULT);

SELECT * FROM a_tbl2;
```

id	name	phone
1	'aaa'	'000-0000'
2	'bbb'	'000-0000'
3	'ccc'	'333-3333'
4	NULL	'000-0000'
5	NULL	'000-0000'
6	'eee'	'000-0000'
7	NULL	'777-7777'
8	'aaa'	'000-0000'

INSERT INTO <remote-table-spec>... SELECT 문

INSERT 문에 SELECT 절의에서 사용한 동일 원격 서버의 원격 테이블을 사용하면 하나 이상의 원격 테이블로부터 특정 검색 조건에 만족하는 질의 결과를 원격 테이블에 삽입할 수 있다. 단, SELECT 문에 명시된 테이블 중 INSERT 구문의 원격 테이블과 다른 서버의 테이블 (로컬 테이블 또는 다른 원격 서버의 테이블)을 사용시 에러가 발생한다.

```
--at remote-side

--creating an empty table which schema replicated from a_tbl1
CREATE TABLE a_tbl2 LIKE a_tbl1;

--at local-side

--inserting multiple rows from SELECT query results
INSERT INTO a_tbl2@server1 SELECT * FROM a_tbl1@server1 WHERE id IS NOT NULL;

--inserting column value with SELECT subquery specified in the value list
INSERT INTO a_tbl2@server1 VALUES(8, SELECT name FROM a_tbl1@server1 WHERE name <'bbb', DEFAULT);

SELECT * FROM a_tbl2@server1;
```

	id	name	phone
=====			
	1	'aaa'	'000-0000'
	2	'bbb'	'000-0000'
	3	'ccc'	'333-3333'
	4	NULL	'000-0000'
	5	NULL	'000-0000'
	6	'eee'	'000-0000'
	7	NULL	'777-7777'
	8	'aaa'	'000-0000'

원격 테이블의 데이터는 로컬 테이블로 삽입 가능하나 로컬 테이블의 데이터를 원격 테이블에 삽입하는 질의는 사용할 수 없으므로 사용시 주의해야 한다. 아래의 질의는 에러가 발생한다.

```
--inserting multiple rows from SELECT query results
INSERT INTO a_tbl2@server1 SELECT * FROM a_tbl1 WHERE id IS NOT NULL;

dblink: local mixed remote DML is not allowed
```

또한 INSERT 구문과 SELECT 구문에서 사용한 원격 테이블의 서버가 다른 경우 질의 수행이 허용되지 않는다. 아래의 질의는 에러가 발생한다.


```
--inserting multiple rows from SELECT query results
INSERT INTO a_tbl2@server1 SELECT * FROM a_tbl1@server2 WHERE id IS NOT NULL;
```

dblink: multi-remote DML **is not** allowed

ON DUPLICATE KEY UPDATE 절

INSERT 문에 **ON DUPLICATE KEY UPDATE** 절을 명시하여 **UNIQUE** 인덱스 또는 **PRIMARY KEY** 제약 조건이 설정된 칼럼에 중복된 값이 삽입되는 상황에서 에러를 출력하지 않고 새로운 값으로 갱신할 수 있다.

참고

- **PRIMARY KEY**와 **UNIQUE** 또는 다수의 **UNIQUE**가 한 테이블에 같이 존재하는 경우, 둘 중 하나에 의해 제약 조건 위반이 발생할 수 있으므로 **ON DUPLICATE KEY UPDATE** 절의 사용을 권장하지 않는다.
- **INSERT**에 실패하여 **UPDATE**가 실행되더라도 한 번 증가한 **AUTO_INCREMENT** 값은 예전 값으로 롤백되지 않는다.

```
<INSERT ... VALUES statement>
<INSERT ... SET statement>
<INSERT ... SELECT statement>
  INSERT ...
  [ON DUPLICATE KEY UPDATE column_name = expr, ... ]
```

- *column_name = expr*: **ON DUPLICATE KEY UPDATE** 뒤에 칼럼 값을 변경하고자 하는 칼럼 이름을 명시하고, 등호 부호를 이용하여 새로운 칼럼 값을 명시한다.

```
--creating a new table having the same schema as a_tbl1
CREATE TABLE a_tbl3 LIKE a_tbl1;
INSERT INTO a_tbl3 SELECT * FROM a_tbl1 WHERE id IS NOT NULL and
name IS NOT NULL;
SELECT * FROM a_tbl3;
```

id	name	phone
1	'aaa'	'000-0000'
2	'bbb'	'000-0000'

```

3  'ccc'          '333-3333'
6  'eee'          '000-0000'

```

```

--insert duplicated value violating UNIQUE constraint
INSERT INTO a_tbl3 VALUES(2, 'bbb', '222-2222');

```

```

ERROR: Operation would have caused one or more unique constraint
violations.

```

ON DUPLICATE KEY UPDATE에서 “affected rows” 값은 새로운 행이 삽입되었을 경우에는 1이고, 존재하는 행이 업데이트되었을 경우에는 2이다.

```

--insert duplicated value with specifying ON DUPLICATE KEY UPDATE
clause
INSERT INTO a_tbl3 VALUES(2, 'ggg', '222-2222')
ON DUPLICATE KEY UPDATE name='ggg', phone = '222-2222';

SELECT * FROM a_tbl3 WHERE id=2;

```

```

      id  name          phone
=====
      2  'ggg'          '222-2222'

2 rows affected.

```

UPDATE

UPDATE 문을 사용하면 대상 테이블 또는 뷰에 저장된 레코드의 칼럼 값을 새로운 값으로 업데이트할 수 있다. **SET** 절에는 업데이트할 칼럼 이름과 새로운 값을 명시하며, **WHERE** 절에는 업데이트할 레코드를 추출하기 위한 조건을 명시한다. 하나의 **UPDATE** 문으로 하나 이상의 테이블 또는 뷰를 업데이트할 수 있다.

```

<UPDATE single table>
UPDATE [schema_name.]table_name | <remote_table_spec> | view_name
SET column_name = {<expr> | DEFAULT} [, column_name = {<expr> |
DEFAULT} ...]
    [WHERE <search_condition>]
    [ORDER BY {col_name | <expr>}]
    [LIMIT row_count]

<remote_table_spec> ::=
[schema_name.]table_name@[schema.name.]server_name [correlation>]

```

```
<UPDATE multiple tables>
UPDATE <table_specifications> SET column_name = {<expr> | DEFAULT}
[, column_name = {<expr> | DEFAULT} ...]
[WHERE <search_condition>]
```

- *<table_specifications>*: **SELECT** 문의 **FROM** 절과 같은 형태의 구문을 지정할 수 있으며, 하나 이상의 테이블을 지정할 수 있다.
- *server_name*: 현재 서버가 아닌 dblink로 연결된 원격 서버의 테이블을 지정할 때 사용한다.
- *column_name*: 업데이트할 칼럼 이름을 지정한다. 하나 이상의 테이블에 대한 칼럼들을 지정할 수 있다.
- *<expr> | DEFAULT*: 해당 칼럼의 새로운 값을 지정하며, 표현식 또는 **DEFAULT** 키워드를 값으로 지정할 수 있다. 단일 결과 레코드를 반환하는 **SELECT** 질의를 지정할 수도 있다.
- *<search_condition>*: **WHERE** 절에 조건식을 명시하면, 조건식을 만족하는 레코드에 대해서만 칼럼 값을 업데이트한다.
- *col_name | <expr>*: 업데이트할 순서의 기준이 되는 칼럼을 지정한다.
- *row_count*: **LIMIT** 절 이후 갱신할 레코드 수를 지정한다. 부호 없는 정수, 호스트 변수 또는 간단한 표현식 중 하나일 수 있다.

*<table_specifications>*에 하나의 테이블이 지정된 경우에만 다음이 허용된다:

- **ORDER BY** 절을 지정할 수 있다. **ORDER BY** 절을 명시하면 해당 칼럼의 순서로 레코드를 업데이트할 수 있다. 이것은 트리거의 실행 순서나 잠금 순서를 유지하고자 할 때 유용하다.
- **LIMIT** 절을 지정할 수 있다. **LIMIT** 절을 명시하면 업데이트할 레코드 수를 제한할 수 있다.
- **SET** 절의 *<expr>*에 분석 함수를 사용할 수 있다. 그러나 *<expr>*에 **SELECT** 질의를 지정한 경우, *<table_specifications>*에 지정된 테이블 수와 관계없이 **SELECT** 질의에서 분석 함수를 사용할 수 있다.

참고

CUBRID 9.0 미만 버전에서는 *<table_specifications>*에 하나의 테이블만 지정할 수 있다.

참고

CUBRID 10.0 버전부터는 **JOIN** 구문을 포함하는 뷰에 대한 업데이트가 가능하다.

예시 1. ORDER BY 절을 이용한 단일 테이블 업데이트

```

DROP TABLE IF EXISTS t1;

CREATE TABLE t1 (id INT);
INSERT INTO t1 VALUES (11), (1), (12), (13), (2), (3), (14), (4);

CREATE TRIGGER trigger1 BEFORE UPDATE ON t1 IF NEW.id < 10 EXECUTE
PRINT 'trigger1 executed';
CREATE TRIGGER trigger2 BEFORE UPDATE ON t1 IF NEW.id > 10 EXECUTE
PRINT 'trigger2 executed';

UPDATE t1 SET id = id + 1;

trigger2 executed
trigger1 executed
trigger2 executed
trigger2 executed
trigger1 executed
trigger1 executed
trigger2 executed
trigger1 executed

```

ORDER BY 절을 이용하면 레코드가 업데이트되는 순서를 변경할 수 있다.

```

TRUNCATE TABLE t1;
INSERT INTO t1 VALUES (11), (1), (12), (13), (2), (3), (14), (4);

UPDATE t1 SET id = id + 1 ORDER BY id;

trigger1 executed
trigger1 executed
trigger1 executed
trigger1 executed
trigger2 executed
trigger2 executed
trigger2 executed
trigger2 executed

```

예시 2. LIMIT 절을 활용한 단일 테이블 업데이트

이 예시는 LIMIT 3 절을 이용하여 *name IS NULL*인 레코드 중 최대 3개만 업데이트한다.

```

DROP TABLE IF EXISTS t1;

CREATE TABLE t1 (id int, name varchar, phone varchar);
INSERT INTO t1 VALUES (NULL, NULL, '000-0000');
INSERT INTO t1 VALUES (1, 'aaa', '111-1111');

```

```
INSERT INTO t1 VALUES ( 2, 'bbb', '222-2222');
INSERT INTO t1 VALUES ( 3, 'ccc', '333-3333');
INSERT INTO t1 VALUES ( 4, NULL, '000-0000');
INSERT INTO t1 VALUES ( 5, NULL, '000-0000');
INSERT INTO t1 VALUES ( 6, 'ddd', '000-0000');
INSERT INTO t1 VALUES ( 7, NULL, '777-7777');
```

```
SELECT * FROM t1 WHERE name IS NULL;
```

id	name	phone
NULL	NULL	'000-0000'
4	NULL	'000-0000'
5	NULL	'000-0000'
7	NULL	'777-7777'

```
UPDATE t1 SET name='update', phone='999-9999' WHERE name IS NULL
LIMIT 3;
```

```
SELECT * FROM t1;
```

id	name	phone
NULL	'update'	'999-9999'
1	'aaa'	'111-1111'
2	'bbb'	'222-2222'
3	'ccc'	'333-3333'
4	'update'	'999-9999'
5	'update'	'999-9999'
6	'ddd'	'000-0000'
7	NULL	'777-7777'

예시 3. 조인을 활용한 업데이트

UPDATE 문에서 테이블 **A**의 레코드가 테이블 **B**의 여러 레코드와 조인될 때, **A**의 레코드는 **B**에서 처음으로 일치하는 레코드의 값만 사용하여 업데이트된다.

이 예시에서, *t1* 테이블의 *id*가 3인 레코드는 *t2* 테이블에서 *rate_id*가 3인 두 개의 레코드와 조인된다. 그러나 *t1* 테이블의 *charge* 칼럼은 *t2* 테이블에서 첫 번째로 일치하는 레코드의 *rate* 칼럼 값만 사용하여 업데이트된다.

```
DROP TABLE IF EXISTS t1, t2;
```

```
CREATE TABLE t1 (id INT PRIMARY KEY, charge DOUBLE);
INSERT INTO t1 VALUES (1, 100.0);
INSERT INTO t1 VALUES (2, 100.0);
INSERT INTO t1 VALUES (3, 100.0);
```

```
CREATE TABLE t2 (rate_id INT, rate DOUBLE);
INSERT INTO t2 VALUES (1, 0.1);
```

```

INSERT INTO t2 VALUES (2, 0.2);
INSERT INTO t2 VALUES (3, 0.3);
INSERT INTO t2 VALUES (3, 0.5);

UPDATE t1 a INNER JOIN t2 b ON a.id = b.rate_id
SET a.charge = a.charge * (1 + b.rate);

SELECT id, TO_CHAR(charge) AS charge FROM t1;

```

id	charge
1	'110'
2	'120'
3	'150'

조인 구문에 대한 자세한 설명은 [조인 질의](#)를 참고한다.

예시 4. 뷰 업데이트

```

DROP TABLE IF EXISTS t1, t2;
DROP VIEW v1;

CREATE TABLE t1 (id INT, val INT) DONT_REUSE_OID;
INSERT INTO t1 VALUES (1, 1);
INSERT INTO t1 VALUES (2, 2);
INSERT INTO t1 VALUES (3, 3);
INSERT INTO t1 VALUES (4, 4);
INSERT INTO t1 VALUES (5, 5);

CREATE TABLE t2 (id INT, val INT) DONT_REUSE_OID;
INSERT INTO t2 VALUES (1, 1);
INSERT INTO t2 VALUES (2, 2);
INSERT INTO t2 VALUES (3, 3);
INSERT INTO t2 VALUES (4, 4);
INSERT INTO t2 VALUES (6, 6);

CREATE VIEW v1 AS
  SELECT b.id, b.val FROM t2 b LEFT JOIN t1 a ON b.id = a.id WHERE
  b.id <= 3;

UPDATE v1 SET val = -1;

SELECT * from v1;

```

id	val
1	-1
2	-1
3	-1

```
SELECT * from t2;
```

id	val
1	-1
2	-1
3	-1
4	4
6	6

⚠ 경고

REUSE OID 테이블이 포함된 뷰의 레코드는 업데이트할 수 없다.

REUSE_OID와 **DONT_REUSE_OID**에 대한 자세한 내용은 **REUSE_OID** (**DONT_REUSE_OID**)를 참고한다.

예시 5. update_use_attribute_references 파라미터를 활용한 업데이트

이 예시의 결과는 **update_use_attribute_references** 파라미터의 값에 따라 달라진다.

- 이 파라미터의 값이 yes인 경우, $c1 = 10$ 의 영향을 받아 $c2$ 의 값이 10으로 갱신된다.
- 이 파라미터의 값이 no인 경우, $c2$ 의 값은 $c1 = 10$ 의 영향을 받지 않고, 해당 레코드에 저장된 $c1$ 값에 따라 1로 갱신된다.

```
DROP TABLE IF EXISTS t1;
```

```
CREATE TABLE t1 (c1 INT, c2 INT);
INSERT INTO t1 values (1, NULL);
```

```
SET SYSTEM PARAMETERS 'update_use_attribute_references=yes';
```

```
UPDATE t1 SET c1 = 10, c2 = c1;
```

```
SELECT * FROM t1;
```

c1	c2
10	10

```
TRUNCATE TABLE t1;
```

```
INSERT INTO t1 values (1, NULL);
```

```
SET SYSTEM PARAMETERS 'update_use_attribute_references=no';
```

```
UPDATE t1 SET c1 = 10, c2 = c1;
```

```
SELECT * FROM t1;
```

c1	c2
10	1

예시 6. 분석 함수를 활용한 단일 테이블 업데이트

UPDATE 문에서 단일 테이블만 지정된 경우, **SET** 절에서 분석 함수를 사용할 수 있다.

```
DROP TABLE IF EXISTS t1, t2;
```

```
CREATE TABLE t1 (id INT);
```

```
INSERT INTO t1 VALUES (1), (2), (3), (4), (5);
```

```
CREATE TABLE t2 (id INT, val INT, update_val DOUBLE, join_update_val DOUBLE);
```

```
INSERT INTO t2 (id, val) SELECT a.id, b.id FROM t1 a, t1 b WHERE b.id <= a.id;
```

```
UPDATE t2 SET update_val = AVG (val) OVER (PARTITION BY id);
```

```
SELECT DISTINCT id, TO_CHAR (update_val) AS update_val FROM t2;
```

id	update_val
1	'1'
2	'1.5'
3	'2'
4	'2.5'
5	'3'

예시 7. 분석 함수를 활용한 다중 테이블 업데이트

예시 6. 분석 함수를 활용한 단일 테이블 업데이트에 이어서, **UPDATE** 문에서 여러 개의 테이블이 지정된 경우, **SET** 절에서는 분석 함수를 사용할 수 없다. 하지만, **SET** 절에 **SELECT** 절의를 지정하면, 지정된 테이블 수에 관계없이 **SELECT** 절의 내에서 분석 함수를 사용할 수 있다.

```
UPDATE t1 a, t2 b SET b.join_update_val = AVG (b.val) OVER (PARTITION BY b.id) WHERE a.id = b.id;
```

```
ERROR: before ' where a.id = b.id; '
Nested analytic functions are not allowed.
```



```
UPDATE t2 c SET c.join_update_val = (SELECT AVG (b.val) OVER
(PARTITION BY b.id) FROM t1 a, t2 b WHERE a.id = b.id AND a.id =
c.id LIMIT 1);
```

```
SELECT DISTINCT id, TO_CHAR (join_update_val) AS join_update_val
FROM t2;
```

id	join_update_val
1	'1'
2	'1.5'
3	'2'
4	'2.5'
5	'3'

예시 8. 원격 테이블 업데이트

테이블 확장명을 사용하면 로컬 서버뿐만 아니라 원격 서버의 테이블도 업데이트할 수 있다.

```
/**
 * on the remote server (e.g., 192.168.6.21)
 */
-- DROP SERVER origin;
CREATE SERVER origin (HOST='localhost', PORT=33000, DBNAME=demodb,
USER=public);
```

```
DROP TABLE IF EXISTS t1;
```

```
CREATE TABLE t1 (id int, name varchar, phone varchar);
INSERT INTO t1@origin VALUES (NULL, NULL, '000-0000');
INSERT INTO t1@origin VALUES ( 1, 'aaa', '111-1111');
INSERT INTO t1@origin VALUES ( 2, 'bbb', '222-2222');
INSERT INTO t1@origin VALUES ( 3, 'ccc', '333-3333');
INSERT INTO t1@origin VALUES ( 4, NULL, '000-0000');
INSERT INTO t1@origin VALUES ( 5, NULL, '000-0000');
INSERT INTO t1@origin VALUES ( 6, 'ddd', '000-0000');
INSERT INTO t1@origin VALUES ( 7, NULL, '777-7777');
```

```
SELECT * FROM t1;
```

id	name	phone
NULL	NULL	'000-0000'
1	'aaa'	'111-1111'
2	'bbb'	'222-2222'
3	'ccc'	'333-3333'
4	NULL	'000-0000'
5	NULL	'000-0000'

```

6 'ddd' '000-0000'
7 NULL '777-7777'

```

```

/**
 * on the local server (e.g., 192.168.6.22)
 */

-- DROP SERVER remote;
CREATE SERVER remote (HOST='192.168.6.21', PORT=33000,
DBNAME=demodb, USER=public);

SELECT * FROM t1@remote WHERE name IS NULL;

      id  name                phone
=====
      NULL NULL                '000-0000'
      4  NULL                '000-0000'
      5  NULL                '000-0000'
      7  NULL                '777-7777'

UPDATE t1@remote SET name='update', phone='999-9999' WHERE name IS
NULL LIMIT 3;

SELECT * FROM t1@remote;

      id  name                phone
=====
      NULL 'update'                '999-9999'
      1  'aaa'                '111-1111'
      2  'bbb'                '222-2222'
      3  'ccc'                '333-3333'
      4  'update'              '999-9999'
      5  'update'              '999-9999'
      6  'ddd'                '000-0000'
      7  NULL                '777-7777'

```

예시 9. 원격 테이블과 조인하여 로컬 테이블 업데이트

```

/**
 * on the remote server (e.g., 192.168.6.21)
 */

DROP TABLE IF EXISTS t2;

CREATE TABLE t2 (rate_id INT, rate DOUBLE);

```

```

/**
 * on the local server (e.g., 192.168.6.22)

```

```

*/

-- DROP SERVER remote;
CREATE SERVER remote (HOST='192.168.6.21', PORT=33000,
DBNAME=demodb, USER=public);

DROP TABLE IF EXISTS t1;

CREATE TABLE t1 (id INT PRIMARY KEY, charge DOUBLE);
INSERT INTO t1 VALUES (1, 100.0);
INSERT INTO t1 VALUES (2, 100.0);
INSERT INTO t1 VALUES (3, 100.0);

INSERT INTO t2@remote VALUES (1, 0.1);
INSERT INTO t2@remote VALUES (2, 0.2);
INSERT INTO t2@remote VALUES (3, 0.3);
INSERT INTO t2@remote VALUES (3, 0.5);

UPDATE t1 a INNER JOIN t2@remote b ON a.id = b.rate_id
SET a.charge = a.charge * (1 + b.rate);

SELECT id, TO_CHAR (charge) AS charge FROM t1;

```

id	charge
1	'110'
2	'120'
3	'150'

⚠ 경고

UPDATE 문에서 로컬 테이블과 원격 테이블을 조인할 때, 원격 테이블의 레코드는 업데이트할 수 없다.

```

UPDATE t1 a INNER JOIN t2@remote b ON a.id = b.rate_id
SET b.rate = b.rate + 0.1;

```

```

ERROR: before ' a INNER JOIN t2@remote b ON a.id = b.rate_id
SET b.rate = b....'
dblink: local mixed remote DML is not allowed

```

REPLACE

REPLACE 문은 **INSERT** 문과 유사하지만, **PRIMARY KEY** 또는 **UNIQUE** 제약 조건이 정의된 칼럼에 중복된 값을 삽입하면 에러 출력 없이 기존 레코드를 삭제한 후 새로운 레코드를 삽입한다. **REPLACE** 문은 삽입 또는 삭제 후 삽입을 수행하므로, **REPLACE** 문을 사용하기 위해서는 테이블에 대한 **INSERT**와 **DELETE** 권한을 동시에 가지고 있어야 한다. 권한 설정에 관한 자세한 내용은 **GRANT** 절을 참고하면 된다.

```
<REPLACE ... VALUES statement>
REPLACE [INTO] <table_specification> [(column_name, ...)]
    {VALUES | VALUE}({expr | DEFAULT}, ...)[,({expr |
DEFAULT}, ...),...]

<REPLACE ... SET statement>
REPLACE [INTO] <table_specification>
    SET column_name = {expr | DEFAULT}[, column_name = {expr |
DEFAULT},...]

<REPLACE ... SELECT statement>
REPLACE [INTO] <table_specification> [column_name, ...)]
    SELECT...

<table_specification> ::= [schema_name.]table_name
    |
[schema_name.]table_name@[schema_name.]server_name
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 새로운 레코드를 삽입할 대상 테이블 이름을 지정한다.
- *server_name*: 대상 테이블의 원격지 서버 이름을 지정한다. 로컬 테이블의 경우 *server_name*을 지정하지 않는다.
- *column_name*: 값을 삽입할 칼럼 이름을 지정한다. 이 값을 생략하면, 테이블에 정의된 모든 칼럼이 명시된 것으로 간주되므로 모든 칼럼에 대한 값을 **VALUES** 뒤에 명시해야 한다. 테이블에 정의된 칼럼 중 일부 칼럼만 명시하면 나머지 칼럼에는 **DEFAULT**로 정의된 값이 할당되며, 정의된 기본 값이 없는 경우 **NULL** 값이 할당된다.
- *expr* | **DEFAULT**: **VALUES** 뒤에는 칼럼에 대응하는 칼럼 값을 명시하며, 표현식 또는 **DEFAULT** 키워드를 값으로 지정할 수 있다. 명시된 칼럼 리스트의 순서와 개수는 칼럼 값 리스트와 대응되어야 하며, 하나의 레코드에 대해 칼럼 값 리스트는 괄호로 처리된다.

REPLACE 문은 새로운 레코드에 의한 **PRIMARY KEY** 또는 **UNIQUE** 인덱스 칼럼 값의 중복을 판단하므로, **PRIMARY KEY** 또는 **UNIQUE** 인덱스가 정의되지 않은 테이블에 대해서는 **INSERT** 문을 사용하는 것이 성능 상 유리하다.

```
--creating a new table having the same schema as a_tbl1
CREATE TABLE a_tbl4 LIKE a_tbl1;
INSERT INTO a_tbl4 SELECT * FROM a_tbl1 WHERE id IS NOT NULL and
name IS NOT NULL;

SELECT * FROM a_tbl4;
```

id	name	phone
1	'aaa'	'000-0000'
2	'bbb'	'000-0000'
3	'ccc'	'333-3333'
6	'eee'	'000-0000'

```
--insert duplicated value violating UNIQUE constraint
REPLACE INTO a_tbl4 VALUES(1, 'aaa', '111-1111'),(2, 'bbb',
'222-2222');
REPLACE INTO a_tbl4 SET id=6, name='fff', phone=DEFAULT;

SELECT * FROM a_tbl4;
```

id	name	phone
3	'ccc'	'333-3333'
1	'aaa'	'111-1111'
2	'bbb'	'222-2222'
6	'fff'	'000-0000'

DELETE

DELETE 문을 사용하여 테이블 내에 레코드를 삭제할 수 있으며, **WHERE 절**과 결합하여 삭제 조건을 명시할 수 있다. 하나의 **DELETE** 문으로 하나 이상의 테이블을 삭제할 수 있다.

```
<DELETE single table>
DELETE [FROM] [schema_name.]table_name[@[schema_name.]server_name]
[<correlation>] WHERE <search_condition> ] [LIMIT row_count]

<DELETE multiple tables FROM ...>
DELETE [schema_name.]table_name[@[schema_name.]server_name] |
<correlation> [{,
[schema_name.]table_name[@[schema_name.]server_name] |
<correlation>}]
FROM <table_specifications> [ WHERE <search_condition> ]
```

```
<DELETE FROM multiple tables USING ...>
DELETE FROM [schema_name.]table_name[@[schema_name.]server_name] |
<correlation> [{,
[schema_name.]table_name[@[schema_name.]server_name] |
<correlation>}]
USING <table_specifications> [ WHERE <search_condition> ]
```

- *<table_specifications>*: **SELECT** 문의 **FROM** 절과 같은 형태의 구문을 지정할 수 있으며, 하나 이상의 테이블을 지정할 수 있다. 11.3 버전부터는 로컬 테이블 뿐만 아니라 원격 테이블도 지정할 수 있다.
- *server_name*: 현재 서버가 아닌 dblink로 연결된 원격 서버의 테이블을 지정할 때 사용한다.
- *correlation*: 지정한 로컬 테이블의 별칭 혹은 원격 테이블의 별칭이다
- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 삭제할 데이터가 포함되어 있는 테이블의 이름을 지정한다. 테이블의 개수가 한 개 일 경우 앞의 **FROM** 키워드를 생략할 수 있다.
- *search_condition*: **WHERE** 절을 이용하여 *search_condition*을 만족하는 데이터만 삭제한다. 생략할 경우 지정된 테이블의 모든 데이터를 삭제한다.
- *row_count*: **LIMIT** 절 에서 삭제할 레코드 수를 지정한다. 부호 없는 정수, 호스트 변수 또는 간단한 표현식 중 하나일 수 있다.

삭제할 테이블이 한 개인 경우에 한하여, **LIMIT** 절을 지정할 수 있다. **LIMIT** 절을 명시하면 삭제할 레코드 수를 한정할 수 있다. **WHERE** 절을 만족하는 레코드 개수가 *row_count*를 초과하면 *row_count* 개의 레코드만 삭제된다.

참고

- 여러 개의 테이블이 있는(multiple table) **DELETE** 문에서는 *<table_specifications>* 내에서만 테이블 별칭(alias)을 정의할 수 있고, *<table_specifications>* 밖에서는 *<table_specifications>* 내에서 정의한 테이블 별칭만 사용할 수 있다.
- CUBRID 9.0 미만 버전에서는 *<table_specifications>*에 한 개의 테이블만 입력할 수 있다.

```
CREATE TABLE a_tbl(
    id INT NOT NULL,
    phone VARCHAR(10));
INSERT INTO a_tbl VALUES(1, '111-1111'), (2, '222-2222'), (3,
'333-3333'), (4, NULL), (5, NULL);

--delete one record only from a_tbl
```

```
DELETE FROM a_tbl WHERE phone IS NULL LIMIT 1;
SELECT * FROM a_tbl;
```

```

      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      5  NULL

```

```
--delete all records from a_tbl
DELETE FROM a_tbl;
```

아래 테이블들은 **DELETE JOIN**을 설명하기 위해 생성한 것이다.

```

CREATE TABLE a_tbl(
  id INT NOT NULL,
  phone VARCHAR(10));
CREATE TABLE b_tbl(
  id INT NOT NULL,
  phone VARCHAR(10));
CREATE TABLE c_tbl(
  id INT NOT NULL,
  phone VARCHAR(10));

INSERT INTO a_tbl VALUES(1,'111-1111'), (2,'222-2222'), (3,
'333-3333'), (4, NULL), (5, NULL);
INSERT INTO b_tbl VALUES(1,'111-1111'), (2,'222-2222'), (3,
'333-3333'), (4, NULL);
INSERT INTO c_tbl VALUES(1,'111-1111'), (2,'222-2222'), (10,
'333-3333'), (11, NULL), (12, NULL);

```

다음 질의들은 여러 개의 테이블들을 조인한 후 삭제를 수행하며, 모두 같은 결과를 보여준다.

```

-- Below four queries show the same result.
-- <DELETE multiple tables FROM ...>

DELETE a, b FROM a_tbl a, b_tbl b, c_tbl c
WHERE a.id=b.id AND b.id=c.id;

DELETE a, b FROM a_tbl a INNER JOIN b_tbl b ON a.id=b.id
INNER JOIN c_tbl c ON b.id=c.id;

-- <DELETE FROM multiple tables USING ...>

DELETE FROM a, b USING a_tbl a, b_tbl b, c_tbl c

```

```
WHERE a.id=b.id AND b.id=c.id;
```

```
DELETE FROM a, b USING a_tbl a INNER JOIN b_tbl b ON a.id=b.id
INNER JOIN c_tbl c ON b.id=c.id;
```

조인 구문에 대한 자세한 설명은 [조인 질의](#)를 참고한다.

확장 테이블명을 사용해서 원격 테이블 데이터를 삭제할 수도 있다. 아래의 질의는 원격 테이블 데이터를 삭제하기 위한 질의이다.

```
--at remote srv1
CREATE TABLE a_tbl(
    id INT NOT NULL,
    phone VARCHAR(10));
INSERT INTO a_tbl VALUES(1,'111-1111'), (2,'222-2222'), (3,
'333-3333'), (4, NULL), (5, NULL);

--at local
--delete one record only from remote a_tbl
DELETE FROM a_tbl@srv1 WHERE phone IS NULL LIMIT 1;
SELECT * FROM a_tbl@srv1;
      id  phone
=====
      1  '111-1111'
      2  '222-2222'
      3  '333-3333'
      5  NULL

--delete all records from remote a_tbl
DELETE FROM a_tbl@srv1;
```

아래 테이블들은 원격 DELETE JOIN을 설명하기 위해 생성한 것이다.

```
--at remote srv1
CREATE TABLE a_tbl(
    id INT NOT NULL,
    phone VARCHAR(10));
CREATE TABLE b_tbl(
    id INT NOT NULL,
    phone VARCHAR(10));
CREATE TABLE c_tbl(
    id INT NOT NULL,
    phone VARCHAR(10));

--at local
INSERT INTO a_tbl@srv1 VALUES(1,'111-1111'), (2,'222-2222'), (3,
'333-3333'), (4, NULL), (5, NULL);
INSERT INTO b_tbl@srv1 VALUES(1,'111-1111'), (2,'222-2222'), (3,
'333-3333'), (4, NULL);
```



```
INSERT INTO c_tbl@srv1 VALUES(1,'111-1111'), (2,'222-2222'), (10,
'333-3333'), (11, NULL), (12, NULL);
```

다음 질의들은 여러 개의 테이블들을 조인한 후 삭제를 수행하며, 모두 같은 결과를 보여준다.

```
-- Below four queries show the same result.
-- <DELETE multiple tables FROM ...>

DELETE a, b FROM a_tbl@srv1 a, b_tbl@srv1 b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;

DELETE a, b FROM a_tbl@srv1 a INNER JOIN b_tbl@srv1 b ON a.id=b.id
INNER JOIN c_tbl@srv1 c ON b.id=c.id;

-- <DELETE FROM multiple tables USING ...>

DELETE FROM a, b USING a_tbl@srv1 a, b_tbl@srv1 b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;

DELETE FROM a, b USING a_tbl@srv1 a INNER JOIN b_tbl@srv1 b ON
a.id=b.id
INNER JOIN c_tbl@srv1 c ON b.id=c.id;
```

다음 질의들은 원격 테이블을 포함한 테이블들(로컬 2개, 원격 1개)을 조인한 후 삭제를 수행하며, 모두 같은 결과를 보여준다.

```
-- Below four queries show the same result.
-- <DELETE multiple tables FROM ...>

DELETE a, b FROM a_tbl a, b_tbl b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;

DELETE a, b FROM a_tbl a INNER JOIN b_tbl b ON a.id=b.id
INNER JOIN c_tbl@srv1 c ON b.id=c.id;

-- <DELETE FROM multiple tables USING ...>

DELETE FROM a, b USING a_tbl a, b_tbl b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;

DELETE FROM a, b USING a_tbl a INNER JOIN b_tbl b ON a.id=b.id
INNER JOIN c_tbl@srv1 c ON b.id=c.id;
```

주의 사항

아래와 같이 로컬 테이블과 원격 테이블이 포함되어 있고, 원격 테이블이 삭제되는 DELETE ... JOIN 쿼리는 허용하지 않는다. 로컬 데이터를 원격으로 보내지 않기 때문이다.

```
DELETE c FROM a_tbl a, b_tbl b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;

DELETE a, b, c FROM a_tbl a, b_tbl b, c_tbl@srv1 c
WHERE a.id=b.id AND b.id=c.id;
```

MERGE

MERGE 문은 하나의 원본으로부터 행들을 선택하여 하나의 테이블 또는 뷰로 갱신이나 삽입을 수행하기 위해 사용되며, 대상 테이블 또는 뷰에 갱신할지 또는 삽입할지 결정하는 조건을 지정할 수 있다. **MERGE** 문은 결정적 문장(deterministic statement)으로, 하나의 문장 내에서 대상 테이블의 같은 행을 여러 번 갱신할 수 없다.

```
MERGE [<merge_hint>] INTO <target> [[AS] <alias>]
USING <source> [[AS] <alias>]
ON <join_condition>
[ <merge_update_clause> ]
[ <merge_insert_clause> ]

<merge_update_clause> ::=
WHEN MATCHED THEN UPDATE
SET <col = expr> [, <col = expr>, ...] [WHERE <update_condition>]
[DELETE WHERE <delete_condition>]

<merge_insert_clause> ::=
WHEN NOT MATCHED THEN INSERT [( <attributes_list> )] VALUES
( <expr_list> ) [WHERE <insert_condition>]

<merge_hint> ::=
/** [ USE_UPDATE_IDX ( <update_index_list> ) ] [ USE_INSERT_IDX
( <insert_index_list> ) ] */
```

- **<target>**: 갱신하거나 삽입할 대상 테이블. 여러 개의 테이블 또는 뷰가 될 수 있으며 **table-name@server-name**처럼 원격 테이블도 지정할 수 있다.
- **<source>**: 데이터를 가져올 원본 테이블. 하나의 테이블 또는 뷰가 될 수 있으며, 부질의(subquery)도 가능하다. **target**이 원격 테이블인 경우, **source**도 **target**과 동일한 서버로 지정된 원격 테이블이거나 뷰이어야 한다. 혹은, **target**에 지정된 서버가 포함된 부질의이어야 한다.
- **<join_condition>**: 갱신할 조건을 명시한다.
- **<merge_update_clause>**: **<join_condition>** 조건이 TRUE이면 대상 테이블의 새로운 칼럼 값들을 지정한다.
 - **UPDATE** 절이 실행되면, **<target>**에 정의된 **UPDATE** 트리거가 활성화된다.

- **<col>**: 업데이트할 칼럼은 반드시 대상 테이블의 칼럼이어야 한다.
- **<expr>**: **<source>**와 **<target>**의 칼럼들을 포함하는 표현식도 가능하다. 또는 **DEFAULT** 키워드도 될 수 있다. **<expr>**은 집계 함수를 포함할 수 없다.
- **<update_condition>**: 특정 조건이 TRUE일 때만 **UPDATE** 연산을 수행한다. 조건은 원본 테이블과 대상 테이블 둘 다 참조할 수 있다. 이 조건을 만족하지 않는 행들은 업데이트하지 않는다.
- **<delete_condition>**: 갱신 이후 삭제할 대상을 지정한다. 이때 **DELETE** 조건은 갱신된 결과 값을 가지고 수행된다. **DELETE** 절이 실행되면 **<target>**에 정의된 **DELETE** 트리거가 활성화된다.
- **ON** 조건 절에서 참조된 칼럼은 업데이트할 수 없다.
- 뷰를 업데이트할 때에는 **DEFAULT** 를 명시할 수 없다.
- **<merge_insert_clause>**: **<join_condition>** 조건이 FALSE이면 대상 테이블의 칼럼으로 값을 삽입한다.
 - **INSERT** 절이 실행되면, **<target>**에 정의된 **INSERT** 트리거들이 활성화된다.
 - **<insert_condition>**: 지정한 조건이 TRUE일 때 삽입 연산을 실행한다. **<source>**의 칼럼만 조건에 포함할 수 있다.
 - **<attribute_list>**: **<target>**에 삽입될 칼럼들이다.
 - **<expr_list>**: 상수 필터 조건은 모든 원본 테이블의 행들을 대상 테이블에 삽입하는 데 사용될 수 있다. 상수 필터 조건의 예로 ON (1=1)과 같은 것이 있다.
 - **<merge_update_clause>**만 지정하거나 **<merge_update_clause>**와 함께 지정할 수 있다. 둘 다 명시한다면 순서는 바뀌어도 된다.
- **<merge_hint>**: **MERGE** 문의 인덱스 힌트
 - **USE_UPDATE_IDX** (**<update_index_list>**): **MERGE** 문의 **UPDATE** 절에서 사용되는 인덱스 힌트. **update_index_list**에 **UPDATE** 절을 수행할 때 사용할 인덱스 이름을 나열한다. **<join_condition>**과 **<update_condition>**에 해당 힌트가 적용된다.
 - **USE_INSERT_IDX** (**<insert_index_list>**): **MERGE** 문의 **INSERT** 절에서 사용되는 인덱스 힌트. **insert_index_list**에 **INSERT** 절을 수행할 때 사용할 인덱스 이름을 나열한다. **<join_condition>**에 해당 힌트가 적용된다.

MERGE 문을 실행하기 위해서는 원본 테이블에 대해 **SELECT** 권한을 가져야 하며, 대상 테이블에 대해 **UPDATE** 절이 포함되어 있으면 **UPDATE** 권한, **DELETE** 절이 포함되어 있으면 **DELETE** 권한, **INSERT** 절이 포함되어 있으면 **INSERT** 권한을 가져야 한다.

다음은 **source_table**의 값을 **target_table**에 합치는 예이다.

```
-- source_table
CREATE TABLE source_table (a INT, b INT, c INT);
INSERT INTO source_table VALUES (1, 1, 1);
INSERT INTO source_table VALUES (1, 3, 2);
INSERT INTO source_table VALUES (2, 4, 5);
INSERT INTO source_table VALUES (3, 1, 3);

-- target_table
CREATE TABLE target_table (a INT, b INT, c INT);
INSERT INTO target_table VALUES (1, 1, 4);
INSERT INTO target_table VALUES (1, 2, 5);
INSERT INTO target_table VALUES (1, 3, 2);
INSERT INTO target_table VALUES (3, 1, 6);
INSERT INTO target_table VALUES (5, 5, 2);

MERGE INTO target_table tt USING source_table st
ON (st.a=tt.a AND st.b=tt.b)
WHEN MATCHED THEN UPDATE SET tt.c=st.c
    DELETE WHERE tt.c = 1
WHEN NOT MATCHED THEN INSERT VALUES (st.a, st.b, st.c);

-- the result of above query
SELECT * FROM target_table;
```

a	b	c
1	2	5
1	3	2
3	1	3
5	5	2
2	4	5

위의 예에서 *source_table*의 칼럼 a, b와 *target_table*의 칼럼 a, b의 값이 같은 경우 *target_table*의 칼럼 c를 *source_table*의 칼럼 c값으로 갱신하고, 그렇지 않은 경우 *source_table*의 레코드 값을 *target_table*에 삽입하는 예이다. 단, 갱신된 레코드에서 *target_table*의 칼럼 c의 값이 1이면 해당 레코드는 삭제된다.

다음은 학생들에게 줄 보너스 점수 테이블(*bonus*)의 레코드를 정리할 때 **MERGE** 문을 이용하는 예제이다.

```
CREATE TABLE bonus (std_id INT, addscore INT);
CREATE INDEX i_bonus_std_id ON bonus (std_id);

INSERT INTO bonus VALUES (1,10);
INSERT INTO bonus VALUES (2,10);
INSERT INTO bonus VALUES (3,10);
INSERT INTO bonus VALUES (4,10);
```

```

INSERT INTO bonus VALUES (5,10);
INSERT INTO bonus VALUES (6,10);
INSERT INTO bonus VALUES (7,10);
INSERT INTO bonus VALUES (8,10);
INSERT INTO bonus VALUES (9,10);
INSERT INTO bonus VALUES (10,10);

CREATE TABLE std (std_id INT, score INT);
CREATE INDEX i_std_std_id ON std (std_id);
CREATE INDEX i_std_std_id_score ON std (std_id, score);

INSERT INTO std VALUES (1,60);
INSERT INTO std VALUES (2,70);
INSERT INTO std VALUES (3,80);
INSERT INTO std VALUES (4,35);
INSERT INTO std VALUES (5,55);
INSERT INTO std VALUES (6,30);
INSERT INTO std VALUES (7,65);
INSERT INTO std VALUES (8,65);
INSERT INTO std VALUES (9,70);
INSERT INTO std VALUES (10,22);
INSERT INTO std VALUES (11,67);
INSERT INTO std VALUES (12,20);
INSERT INTO std VALUES (13,45);
INSERT INTO std VALUES (14,30);

MERGE INTO bonus t USING (SELECT * FROM std WHERE score < 40) s
ON t.std_id = s.std_id
WHEN MATCHED THEN
UPDATE SET t.addscore = t.addscore + s.score * 0.1
WHEN NOT MATCHED THEN
INSERT (t.std_id, t.addscore) VALUES (s.std_id, 10 + s.score * 0.1)
WHERE s.score <= 30;

SELECT * FROM bonus ORDER BY 1;

```

std_id	addscore
1	10
2	10
3	10
4	14
5	10
6	13
7	10
8	10
9	10
10	12
12	12
14	13

위의 예에서 원본 테이블은 *score*가 40 미만인 *std* 테이블의 레코드 집합이고, 대상 테이블은 *bonus*이다. **UPDATE** 절에서 점수(*std.score*)가 40점 미만인 학생 번호(*std_id*)는 4, 6, 10, 12, 14이고 이들 중 보너스 테이블(*bonus*)에 있는 4, 6, 10번에게는 기존 보너스 점수(*bonus.addscore*)에 자신의 점수의 10%를 추가로 부여한다. **INSERT** 절에서는 보너스 테이블에 없는 12, 14번에게 10점과 자신의 점수의 10%를 추가로 부여한다.

다음은 **MERGE** 문에 인덱스 힌트를 사용하는 예이다.

```
CREATE TABLE target (i INT, j INT);
CREATE TABLE source (i INT, j INT);

INSERT INTO target VALUES (1,1),(2,2),(3,3);
INSERT INTO source VALUES (1,11),(2,22),(4,44),(5,55),(7,77),(8,88);

CREATE INDEX i_t_i ON target(i);
CREATE INDEX i_t_ij ON target(i,j);
CREATE INDEX i_s_i ON source(i);
CREATE INDEX i_s_ij ON source(i,j);

MERGE /*+ RECOMPILE USE_UPDATE_IDX(i_s_ij) USE_INSERT_IDX(i_t_ij,
i_t_i) */
INTO target t USING source s ON t.i=s.i
WHEN MATCHED THEN UPDATE SET t.j=s.j WHERE s.i <> 1
WHEN NOT MATCHED THEN INSERT VALUES (i,j);
```

MERGE 문을 실행하기 위해서는 원본 테이블에 대해 SELECT 권한을 가져야 하며, 대상 테이블에 대해 UPDATE 절이 포함되어 있으면 UPDATE 권한, DELETE 절이 포함되어 있으면 DELETE 권한, INSERT 절이 포함되어 있으면 INSERT 권한을 가져야 한다.

다음은 원격 서버의 *source_table* 값을 원격 서버의 *target_table*에 합치는 예이다.

```
-- at remote server
-- source_table
CREATE TABLE source_table (a INT, b INT, c INT);
CREATE TABLE target_table (a INT, b INT, c INT);

-- at local server
-- source table
INSERT INTO source_table@srv1 VALUES (1, 1, 1);
INSERT INTO source_table@srv1 VALUES (1, 3, 2);
INSERT INTO source_table@srv1 VALUES (2, 4, 5);
INSERT INTO source_table@srv1 VALUES (3, 1, 3);

-- target_table
INSERT INTO target_table@srv1 VALUES (1, 1, 4);
INSERT INTO target_table@srv1 VALUES (1, 2, 5);
INSERT INTO target_table@srv1 VALUES (1, 3, 2);
INSERT INTO target_table@srv1 VALUES (3, 1, 6);
```

```

INSERT INTO target_table@srv1 VALUES (5, 5, 2);

MERGE INTO target_table@srv1 tt USING source_table@srv1 st
ON (st.a=tt.a AND st.b=tt.b)
WHEN MATCHED THEN UPDATE SET tt.c=st.c
DELETE WHERE tt.c = 1
WHEN NOT MATCHED THEN INSERT VALUES (st.a, st.b, st.c);

-- the result of above query
SELECT * FROM target_table@srv1;

```

a	b	c
1	2	5
1	3	2
3	1	3
5	5	2
2	4	5

다음은 원격 서버의 source_table 값을 로컬 서버의 target_table에 합치는 예이다.

```

-- at remote server
-- source_table
CREATE TABLE source_table (a INT, b INT, c INT);

-- at local server
CREATE TABLE target_table (a INT, b INT, c INT);

-- source table insert
INSERT INTO source_table@srv1 VALUES (1, 1, 1);
INSERT INTO source_table@srv1 VALUES (1, 3, 2);
INSERT INTO source_table@srv1 VALUES (2, 4, 5);
INSERT INTO source_table@srv1 VALUES (3, 1, 3);

-- target_table insert
INSERT INTO target_table VALUES (1, 1, 4);
INSERT INTO target_table VALUES (1, 2, 5);
INSERT INTO target_table VALUES (1, 3, 2);
INSERT INTO target_table VALUES (3, 1, 6);
INSERT INTO target_table VALUES (5, 5, 2);

MERGE INTO target_table tt USING source_table@srv1 st
ON (st.a=tt.a AND st.b=tt.b)
WHEN MATCHED THEN UPDATE SET tt.c=st.c
DELETE WHERE tt.c = 1
WHEN NOT MATCHED THEN INSERT VALUES (st.a, st.b, st.c);

-- the result of above query
SELECT * FROM target_table;

```

a	b	c
=====	=====	=====
1	2	5
1	3	2
3	1	3
5	5	2
2	4	5

⚠ 경고

아래의 예처럼 target이 remote이고 source가 local인 경우는 에러가 발생한다.

```

MERGE INTO target_table@srv1 tt USING source_table st
ON (st.a=tt.a AND st.b=tt.b)
WHEN MATCHED THEN UPDATE SET tt.c=st.c
    DELETE WHERE tt.c = 1
WHEN NOT MATCHED THEN INSERT VALUES (st.a, st.b, st.c);

```

또한 아래의 예처럼 target과 source가 각각 다른 서버인 경우도 에러가 발생한다.

```

MERGE INTO target_table@srv1 tt USING source_table@srv2 st
ON (st.a=tt.a AND st.b=tt.b)
WHEN MATCHED THEN UPDATE SET tt.c=st.c
    DELETE WHERE tt.c = 1
WHEN NOT MATCHED THEN INSERT VALUES (st.a, st.b, st.c);

```

TRUNCATE

TRUNCATE 문은 명시된 테이블의 모든 레코드들을 삭제한다.

WHERE 조건이 없는 **DELETE FROM** [schema_name.]table_name 문을 사용하는 것에 비해 일반적으로 다음의 이점을 지닌다.

- 인덱스와 제약사항 사전 처리, 레코드들 일괄 삭제로 성능면에서 훨씬 뛰어나다.
- 레코드들을 개별 삭제하는 것이 아니기 때문에 각 레코드에 대한 VACUUM 비용이 없다.
- 로그양이 훨씬 적기 때문에 HA 복제, 복구 그리고 롤백(rollback) 등에서 더 좋은 성능을 보인다.

i 참고

- **TRUNCATE** 문을 사용해서 삭제하면 **DELETE** 트리거가 활성화되지 않는다.

- **TRUNCATE** 문은 해당 테이블의 **AUTO INCREMENT** 칼럼을 초기화하여, 다시 데이터가 입력되면 **AUTO INCREMENT** 칼럼의 초기값부터 생성된다.
- **TRUNCATE** 문을 수행하려면 해당 테이블에 **ALTER, INDEX, DELETE** 권한이 필요하다. 권한을 부여하는 방법은 **GRANT** 를 참고한다.
- 테이블이 **DONT_REUSE_OID** 옵션(**DONT_REUSE_OID**)으로 생성된 경우 다른 테이블의 컬럼이 해당 테이블을 참조할 수 있다. 이렇게 외부 참조가 있을 경우 일괄 삭제가 불가능하며 내부적으로 **DELETE FROM** 연산을 통해 **TRUNCATE**가 수행된다.

```
TRUNCATE [ TABLE ] [schema_name.]table_name [ CASCADE ]
```

- *schema_name*: 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 삭제할 데이터가 포함되어 있는 테이블의 이름을 지정한다.
- **CASCADE**: 외래 키 (Foreign Key) 참조 관계에 있는 모든 테이블들의 데이터를 삭제한다. 대상 테이블에 **PRIMARY KEY** 제약 조건이 정의되어 있고, 이 **PRIMARY KEY** 를 하나 이상의 외래 키가 참조하고 있어야 한다. 이 때 외래 키의 **ON DELETE** 동작은 **CASCADE** 로 정의되어 있어야 한다. 외래 키 참조가 있는 경우 **CASCADE** 옵션을 사용하지 않으면, **TRUNCATE** 는 실패한다. 또한, 여러 테이블이 외래 키 참조로 연결되어 있고, 하나라도 **ON DELETE** 동작이 **CASCADE** 가 아닌 경우 역시 실패한다. 외래 키 제약 조건에 대한 자세한 내용은 **FOREIGN KEY 제약** 을 참고한다.

```
CREATE TABLE a_tbl(A INT AUTO_INCREMENT(3,10) PRIMARY KEY);
INSERT INTO a_tbl VALUES (NULL),(NULL),(NULL);
SELECT * FROM a_tbl;
```

```

      a
=====
      3
     13
     23
```

```
--AUTO_INCREMENT column value increases from the initial value after
truncating the table
TRUNCATE TABLE a_tbl;
INSERT INTO a_tbl VALUES (NULL);
SELECT * FROM a_tbl;
```

```

a
=====
3

```

다음은 **CASCADE** 옵션을 사용하는 예이다. 일부 레코드의 외래 키 컬럼이 NULL로 어떠한 레코드를 가리키지 않더라도 CASCADE로 전파된 TRUNCATE는 모든 레코드를 제거한다.

```

-- a_tbl <- b_tbl <- c_tbl
--      <- d_tbl
CREATE TABLE a_tbl(a1 INT PRIMARY KEY);
CREATE TABLE b_tbl(b1 INT PRIMARY KEY, b2 INT FOREIGN KEY REFERENCES
a_tbl (a1) ON DELETE CASCADE);
CREATE TABLE c_tbl(c1 INT PRIMARY KEY, c2 INT FOREIGN KEY REFERENCES
b_tbl (b1) ON DELETE CASCADE);
CREATE TABLE d_tbl(d1 INT PRIMARY KEY, d2 INT FOREIGN KEY REFERENCES
a_tbl (a1) ON DELETE CASCADE);
INSERT INTO a_tbl VALUES (1);
INSERT INTO b_tbl VALUES (2, 1);
INSERT INTO c_tbl VALUES (3, 2);
INSERT INTO d_tbl VALUES (5, 1);
INSERT INTO d_tbl VALUES (4, NULL); -- not refer to any record in
a_tbl, but will be deleted.
SELECT * FROM a_tbl;
SELECT * FROM b_tbl;
SELECT * FROM c_tbl;
SELECT * FROM d_tbl;

TRUNCATE a_tbl CASCADE;
SELECT * FROM a_tbl, b_tbl, c_tbl, d_tbl;

```

```

a1
=====
1

b1          b2
=====
2          1

c1          c2
=====
3          2

d1          d2
=====
5          1
4          NULL

```

```
-- after TRUNCATE a_tbl CASCADE;
There are no results.
```

PREPARED STATEMENT

prepared statement 기능은 보통 JDBC, PHP, ODBC 등의 인터페이스 함수를 통해서 사용할 수 있는데, SQL 레벨에서도 직접 수행할 수 있다. prepared statement 사용을 위해 다음의 SQL 문을 제공한다.

- 실행하고자 하는 SQL 문을 준비한다.

```
PREPARE stmt_name FROM preparable_stmt
```

- prepared statement를 실행한다.

```
EXECUTE stmt_name [USING value [, value] ...]
```

- prepared statement를 해제한다.

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

참고

- SQL 수준의 PREPARE 문은 CSQL 인터프리터에서만 사용할 것을 권장한다. 응용 프로그램에서 사용하는 경우 정상 동작을 보장하지 않는다.
- SQL 수준의 PREPARE 문은 DB 연결 당 개수가 최대 20개로 제한된다. SQL 수준의 PREPARE 문은 DB 서버의 메모리 자원을 사용하므로 DB 서버 메모리의 남용을 방지하기 위해 제한된다.
- 인터페이스 함수의 prepared statement는 브로커 파라미터인 `MAX_PREPARED_STMT_COUNT`를 통해 DB 연결 당 prepared statement 개수가 제한된다.

PREPARE 문

PREPARE 문은 **FROM** 절의 *preparable_stmt*에 지정된 질의문을 준비하고, 이후에 해당 SQL 문을 참조할 때 사용될 이름을 *stmt_name*에 할당한다. 예제는 **EXECUTE** 문 을 참고한다.

```
PREPARE stmt_name FROM preparable_stmt
```

- *stmt_name* : prepared statement의 이름을 할당한다. 해당 클라이언트 세션에 이미 동일한 *stmt_name* 을 가지는 SQL 문이 존재하면, 기존 prepared statement을 해제한 후 새로운 SQL 문을 준비한다. 주어진 SQL 문의 오류로 인해 **PREPARE** 문이 정상 수행되지 않는 경우, 해당 SQL 문에 할당된 *stmt_name* 은 존재하지 않는 것으로 처리된다.
- *preparable_stmt* : 반드시 단일 SQL 문이어야 하며, 여러 개의 SQL 문을 지정할 수 없다. *preparable_stmt* 인자에 바인드 파라미터(?)를 사용할 수 있으며, 이를 따옴표로 감싸지 않아야 한다.

참고

PREPARE 문은 응용 프로그램이 서버에 연결하면서 시작되며 응용 프로그램이 연결을 종료하거나 세션 기간이 만료되기 전까지 유지된다. 세션 기간은 시스템 파라미터의 **session_state_timeout** 파라미터로 설정할 수 있으며, 기본값은 **21600** 초(=6시간)이다.

세션에 의해 관리되는 데이터는 **PREPARE** 문 외에 사용자 정의 변수, 가장 마지막에 삽입한 ID(**LAST_INSERT_ID**), 가장 마지막에 실행한 문장에 의해 영향 받은 레코드의 개수(**ROW_COUNT**)를 포함한다.

EXECUTE 문

EXECUTE 문은 prepared statement을 실행하며, prepared statement에 바인드 파라미터(?)를 포함하면 **USING** 절 뒤에 데이터 값을 바인딩할 수 있다. **USING** 절에서는 세션 변수뿐만 아니라 리터럴, 입력 파라미터와 같은 값도 지정할 수 있다.

```
EXECUTE stmt_name [USING value [, value] ...]
```

- *stmt_name* : 실행하고자 하는 prepared statement에 부여된 이름을 지정한다. *stmt_name* 이 유효하지 않거나 prepared statement가 존재하지 않는 경우 에러가 출력된다.
- *value* : 바인드 파라미터가 prepared statement에 있는 경우 바인딩할 데이터를 입력한다. 바인드 파라미터와 데이터의 개수 및 순서가 대응되어야 한다. 그렇지 않으면 에러가 출력된다.

```
PREPARE st FROM 'SELECT 1 + ?';
EXECUTE st USING 4;
```

```

1+ ?:0
=====
5

```

```

PREPARE st FROM 'SELECT 1 + ?';
SET @a=3;
EXECUTE st USING @a;

```

```

1+ ?:0
=====
4

```

```

PREPARE st FROM 'SELECT ? + ?';
EXECUTE st USING 1,3;

```

```

?:0 + ?:1
=====
4

```

```

PREPARE st FROM 'SELECT ? + ?';
EXECUTE st USING 'a','b';

```

```

?:0 + ?:1
=====
'ab'

```

```

PREPARE st FROM 'SELECT FLOOR(?)';
EXECUTE st USING '3.2';

```

```

floor( ?:0 )
=====
3.0000000000000000e+000

```

DEALLOCATE PREPARE 문, DROP PREPARE 문

DEALLOCATE PREPARE 문과 **DROP PREPARE** 문은 동일하며, prepared statement를 해제한다. **DEALLOCATE PREPARE** 문 또는 **DROP PREPARE** 문을 수행하지 않더라도 클라이언트 세션이 종료되면, 서버에 의해 모든 prepared statement가 자동 해제된다.

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

- *stmt_name* : 해제하고자 하는 prepared statement에 부여된 이름을 지정한다. *stmt_name* 이 유효하지 않거나 prepared statement가 존재하지 않으면 에러가 출력된다.

```
DEALLOCATE PREPARE stmt1;
```

DO

DO 문은 지정된 연산식을 실행하지만 결과 값을 리턴하지 않는다. 지정된 연산식이 문법에 맞게 쓰여지지 않으면 에러를 반환하므로, 연산식의 문법이 올바른지 여부를 확인하는 데 사용할 수 있다.

DO 문은 데이터베이스 서버에서 연산 결과 또는 에러를 반환하지 않기 때문에, 일반적으로 **SELECT** 문보다 수행 속도가 빠르다.

```
DO expression
```

- *expression* : 임의의 연산식을 지정한다.

```
DO 1+1;
DO SYSDATE + 1;
DO (SELECT count(*) FROM athlete);
```

CTE

CTE(Common Table Expressions)는 질의문과 관련된 임시 테이블(결과 목록)이다. 질의문 내에서 CTE를 여러 번 참조할 수 있으며 질의문 범위 내에서만 표시된다. CTE를 사용하면 질의문 로직을 보다 효과적으로 분리하여 수행 성능을 개선할 수 있으며 계층 질의문을 생성할 때 **CONNECT BY** 질의문 또는 복잡한 질의 대신 재귀적 CTE를 사용할 수 있다.

CTE는 **WITH** 절로 시작한다. 부질의 목록과 부질의를 사용하는 최종 질의가 있어야 한다. 각 부질의(테이블 표현식)는 이름과 질의 정의를 포함한다. 테이블 표현식은 이전에 동일한 질의문에 정의된 다른 테이블 표현식을 참조할 수 있다. 구문은 다음과 같다.

```
WITH
  [RECURSIVE <recursive_cte_name> [ (<recursive_column_names>) ] AS
  <recursive_sub-query>]
  <cte_name1> [ (<cte1_column_names>) ] AS <sub-query1>
  <cte_name2> [ (<cte2_column_names>) ] AS <sub-query2>
```

```
...
<final_query>
```

- *recursive_cte_name*, *cte_name1*, *cte_name2* : 테이블 표현식(부질의)의 식별자
- *recursive_column_names*, *cte1_column_names*, *cte2_column_names* : 각 테이블 표현식 결과 컬럼에 대한 식별자
- *sub-query1*, *sub-query2* : 각 테이블 표현식을 정의하는 부질의
- *final_query* : 이전에 정의된 테이블 표현식을 사용하는 질의. 일반적으로 **FROM** 절은 CTE 식별자를 포함한다.

가장 단순한 사용법은 테이블 표현식의 결과 목록을 결합하는 것이다.

```
CREATE TABLE products (id INTEGER PRIMARY KEY, parent_id INTEGER,
item VARCHAR(100), price INTEGER);
INSERT INTO products VALUES (1, -1, 'Drone', 2000);
INSERT INTO products VALUES (2, 1, 'Blade', 10);
INSERT INTO products VALUES (3, 1, 'Brushless motor', 20);
INSERT INTO products VALUES (4, 1, 'Frame', 50);
INSERT INTO products VALUES (5, -1, 'Car', 20000);
INSERT INTO products VALUES (6, 5, 'Wheel', 100);
INSERT INTO products VALUES (7, 5, 'Engine', 4000);
INSERT INTO products VALUES (8, 5, 'Frame', 4700);

WITH
  of_drones AS (SELECT item, 'drones' FROM products WHERE parent_id =
1),
  of_cars AS (SELECT item, 'cars' FROM products WHERE parent_id = 5)
SELECT * FROM of_drones UNION ALL SELECT * FROM of_cars ORDER BY 1;
```

item	'drones'
=====	
'Blade'	'drones'
'Brushless motor'	'drones'
'Car'	'cars'
'Drone'	'drones'
'Engine'	'cars'
'Frame'	'drones'
'Frame'	'cars'
'Wheel'	'cars'

한 CTE의 부질의가 다른 CTE의 부질의에 참조될 수 있다(참조되는 CTE가 미리 정의되어 있어야 함):

```
WITH
  of_drones AS (SELECT item FROM products WHERE parent_id = 1),
```

```
filter_common_with_cars AS (SELECT * FROM of_drones INTERSECT
SELECT item FROM products WHERE parent_id = 5)
SELECT * FROM filter_common_with_cars ORDER BY 1;
```

```
item
=====
'Frame'
```

다음과 같은 경우 오류가 발생한다.:

- 둘 이상의 CTE에서 동일한 식별자명 사용.
- 중첩된 **WITH** 절 사용.

```
WITH
my_cte AS (SELECT item FROM products WHERE parent_id = 1),
my_cte AS (SELECT * FROM my_cte INTERSECT SELECT item FROM products
WHERE parent_id = 5)
SELECT * FROM my_cte ORDER BY 1;
```

```
before '
    SELECT * FROM my_cte ORDER BY 1;
'
CTE name ambiguity, there are more than one CTEs with the same name:
'my_cte'.
```

```
WITH
of_drones AS (SELECT item FROM products WHERE parent_id = 1),
of_cars1 AS (WITH
    of_cars2 AS (SELECT item FROM products WHERE
parent_id = 5)
    SELECT * FROM of_cars2
)
SELECT * FROM of_drones, of_cars1 ORDER BY 1;
```

```
before '
    SELECT * FROM of_drones, of_cars1 ORDER BY 1;
'
Nested WITH clauses are not supported.
```

CTE 컬럼명

각 CTE 결과의 컬럼명은 CTE 이름 다음에 지정할 수 있다. CTE 컬럼 목록의 요소 수는 CTE 부질의 컬럼 수와 일치해야 한다.

WITH

```
of_drones (product_name, product_type, price) AS (SELECT item,
'drones', price FROM products WHERE parent_id = 1),
of_cars (product_name, product_type, price) AS (SELECT item,
'cars', price FROM products WHERE parent_id = 5)
SELECT * FROM of_drones UNION ALL SELECT * FROM of_cars ORDER BY
product_type, price;
```

WITH

```
of_drones (product_name, product_type, price) AS (SELECT item,
'drones' as type, MAX(price) FROM products WHERE parent_id = 1 GROUP
BY type),
of_cars (product_name, product_type, price) AS (SELECT item,
'cars' as type, MAX (price) FROM products WHERE parent_id = 5 GROUP
BY type)
SELECT * FROM of_drones UNION ALL SELECT * FROM of_cars ORDER BY
product_type, price;
```

product_name	product_type	price
'Wheel'	'cars'	100
'Engine'	'cars'	4000
'Frame'	'cars'	4700
'Blade'	'drones'	10
'Brushless motor'	'drones'	20
'Frame'	'drones'	50

product_name	product_type	price
'Wheel'	'cars'	4700
'Blade'	'drones'	50

CTE에 컬럼명이 없으면 CTE의 첫 번째 내부 Select 문에서 컬럼명을 가져온다. 원본 구문에 따라 표현식 결과 컬럼명이 결정된다.

WITH

```
of_drones AS (SELECT item, 'drones', MAX(price) FROM products WHERE
parent_id = 1 GROUP BY 2),
of_cars AS (SELECT item, 'cars', MAX (price) FROM products WHERE
parent_id = 5 GROUP BY 2)
SELECT * FROM of_drones UNION ALL SELECT * FROM of_cars ORDER BY 1;
```

item	'drones'	max(products.price)
'Blade'	'drones'	50
'Wheel'	'cars'	4700

재귀절

RECURSIVE 키워드를 사용하여 반복되는 질의를 구성할 수 있다(테이블 표현식 부질의 정의 자체 이름 포함). 재귀 테이블 표현식은 비재귀적 부분과 재귀적 부분(CTE 이름으로 부질의 참조)으로 구성된다. **UNION ALL** 질의 연산자를 사용하여 재귀적 부분과 비재귀적 부분을 결합해야 한다. 무한 반복하지 않도록 재귀적 부분을 정의해야 한다. 또한 재귀적 부분에 집계 함수를 포함하는 경우 집계 함수가 항상 튜플을 반환하고 재귀 반복이 계속되므로 **GROUP BY** 절도 포함해야 한다. **WHERE** 절의 조건을 더 이상 만족하지 않고 현재 수행된 반복의 결과가 없을 경우 재귀 반복이 중단된다.

```
WITH
  RECURSIVE cars (id, parent_id, item, price) AS (
    SELECT id, parent_id, item, price
      FROM products WHERE item LIKE 'Car%'
    UNION ALL
    SELECT p.id, p.parent_id, p.item, p.price
      FROM products p
    INNER JOIN cars rec_cars ON p.parent_id =
rec_cars.id)
SELECT item, price FROM cars ORDER BY 1;
```

item	price
'Car'	20000
'Engine'	4000
'Frame'	4700
'Wheel'	100

재귀적 CTE는 무한 루프에 빠질 수 있다. 이런 경우를 피하기 위해서 시스템 파라미터 **cte_max_recursions** 를 원하는 임계치로 설정해야 한다. 이 파라미터의 기본값은 2,000번 재귀 반복이며, 최대값은 1,000,000 최소값은 2이다.

```
SET SYSTEM PARAMETERS 'cte_max_recursions=2';
WITH
  RECURSIVE cars (id, parent_id, item, price) AS (
    SELECT id, parent_id, item, price
      FROM products WHERE item LIKE 'Car%'
    UNION ALL
    SELECT p.id, p.parent_id, p.item, p.price
      FROM products p
    INNER JOIN cars rec_cars ON p.parent_id =
rec_cars.id)
SELECT item, price FROM cars ORDER BY 1;
```

In the command **from** line 9,
Maximum recursions 2 reached executing CTE.

⚠ 경고

- CTE 부질의 복잡도에 따라, 많은 량의 데이터가 생산되며, 심지어 **cte_max_recursions** 의 기본값만으로도 디스크 공간 부족을 발생할 수 있다.

재귀적 CTE의 실행 알고리즘은 다음과 같이 요약될 수 있다:

- CTE의 비재귀적 부분을 수행하고 결과를 최종 결과 셋에 추가
- 비재귀적 부분에서 얻은 결과 셋을 사용하여 재귀적 부분을 수행하고, 결과를 최종 결과 셋에 추가한 후, 결과 셋 내에서 현재 반복의 시작과 끝을 기억한다
- 이전 반복의 결과 셋을 사용하여 비재귀적 부분의 수행을 반복하고 해당 결과를 최종 결과 셋에 추가
- 재귀 반복에서 결과가 생성되지 않으면 중지
- 설정된 최대 반복 횟수에 도달하는 경우에도 중지

재귀적 CTE를 **FROM** 절에서 바로 참조해야 한다. 부질의에서 참조하면 오류가 발생한다.

```
WITH
  RECURSIVE cte1(x) AS SELECT c FROM t1 UNION ALL SELECT * FROM
  (SELECT cte1.x + 1 FROM cte1 WHERE cte1.x < 5)
  SELECT * FROM cte1;
```

```
before '
SELECT * FROM cte1;
'

Recursive CTE 'cte1' must be referenced directly in its recursive
query.
```

DML과 CREATE에서 CTE의 사용

SELECT 문에 대한 사용 외에도 CTE는 다른 문장에도 사용될 수 있다. CTE가 **CREATE TABLE** *table_name* **AS SELECT** 에 사용될 수 있다:

```
CREATE TABLE inc AS
  WITH RECURSIVE cte (n) AS (
    SELECT 1
    UNION ALL
    SELECT n + 1
    FROM cte
    WHERE n < 3)
```

```
SELECT n FROM cte;

SELECT * FROM inc;
```

```
      n
=====
      1
      2
      3
```

또한, **INSERT/REPLACE INTO** *table_name* **SELECT** 도 CTE 사용이 가능하다:

```
INSERT INTO inc
  WITH RECURSIVE cte (n) AS (
    SELECT 1
    UNION ALL
    SELECT n + 1
    FROM cte
    WHERE n < 3)
  SELECT * FROM cte;

REPLACE INTO inc
  WITH cte AS (SELECT * FROM inc)
  SELECT * FROM cte;
```

또한 **UPDATE** 의 부질의에서도 사용 가능하다:

```
CREATE TABLE green_products (producer_id INTEGER, sales_n INTEGER,
product VARCHAR, product_type INTEGER, price INTEGER);
INSERT INTO green_products VALUES (1, 99, 'bicycle', 1, 99);
INSERT INTO green_products VALUES (2, 337, 'bicycle', 1, 129);
INSERT INTO green_products VALUES (3, 5012, 'bicycle', 1, 199);
INSERT INTO green_products VALUES (1, 989, 'scooter', 2, 899);
INSERT INTO green_products VALUES (3, 3211, 'scooter', 2, 599);
INSERT INTO green_products VALUES (4, 2312, 'scooter', 2, 1009);

WITH price_increase_th AS (
  SELECT SUM(sales_n) * 7 / 10 AS threshold, product_type
  FROM green_products
  GROUP BY product_type
)
UPDATE green_products gp JOIN price_increase_th th ON
gp.product_type = th.product_type
SET price = price + (price / 10)
WHERE sales_n >= threshold;
```

또한, **DELETE** 의 부질의에서도 사용가능하다:

```

WITH product_removal_th AS (
    SELECT SUM (sales_n) / 20 AS threshold, product_type
    FROM green_products
    GROUP BY product_type
)
DELETE
FROM green_products gp
WHERE sales_n < (select threshold from product_removal_th WHERE
product_type = gp.product_type);

```

CALL

CALL 문은 데이터 베이스에 정의된 메서드 또는 저장 함수/프로시저와 같은 저장 루틴을 호출하기 위해 사용한다. CUBRID에서 지원하는 저장 루틴은 다음과 같다.

- **CREATE PROCEDURE** 구문을 통해 등록한 저장 프로시저
- **CREATE FUNCTION** 구문을 통해 등록한 저장 함수
- C로 작성한 **메서드**

```

CALL <routine_clause> ;

<routine_clause> ::=
    routine_name ([<arg_value> [{, <arg_value> } ...]])
[<method_call_target>] [<to_variable>]

<arg_value>
    literal |
    :host_variable |
    any CSQL expression

<method_call_target> ::=
    ON [CLASS] object-valued-expression

<to_variable> ::=
    INTO :host_variable |
    TO :host_variable

```

- *routine_name*: 호출할 메서드 또는 저장 함수/프로시저의 이름을 지정한다. 대소문자를 구분하지 않는다.
- *<arg_value>*: 호출할 저장 루틴의 인수를 지정한다. 리터럴 값, 호스트 변수와 표현식을 지정할 수 있다.

- `<method_call_target>`: C로 작성한 메서드를 사용할 때만 지정한다. 클래스 이름, 변수, 혹은 또 다른 메서드 호출(객체를 반환하는)을 포함하는 객체 값의 표현식(object-valued-expression)을 지정한다. 만약 클래스 객체에서 동작하는 클래스 메서드를 호출하려면, `< call_target>` 앞에 반드시 **CLASS** 키워드가 있어야 한다. 이 경우 클래스 이름은 클래스 메서드가 정의된 클래스의 이름이어야 한다. 만약 인스턴스 메서드를 호출하려면, 인스턴스 객체를 나타내는 식을 지정해야 한다.
- `<to_variable>`: 저장 루틴이 반환하는 값을 호스트 변수에 저장할 수 있다.

CALL 문에서 지정한 이름에 대하여 CUBRID에 등록된 저장 루틴을 다음과 같이 검색한다.

1. **CALL** 문에 대상 클래스가 있는 경우 메서드로 처리한다.
2. **CALL** 문에 대상 클래스가 없는 경우 먼저 Java 저장 함수/프로시저 수행 여부를 검사하고 같은 이름이 존재하면 수행한다.
3. 2에서 Java 저장 함수/프로시저가 존재하지 않으면 메서드 수행 여부를 검사하여 같은 이름이 존재하면 수행한다.

```
CALL Hello() INTO :HELLO;
CALL Sp_int(3) INTO :i;
CALL phone_info('Tom', '016-111-1111');
```

만약 존재하지 않는 메소드 또는 저장 함수/프로시저를 호출하는 경우에는 다음과 같은 에러가 나타난다.

- **CALL** 문에 인자가 없는 경우는 메서드와 구분되므로 “ERROR: Stored procedure/function ‘deposit’ does not exist.”라는 오류 메시지가 나타난다.
- **CALL** 문에 인자가 있는 경우에는 메서드와 구분할 수 없기 때문에 “ERROR: Methods require an object as their target.”이라는 메시지가 나타난다.

```
// CALL 문에 인자가 없는 경우
CALL deposit();
```

```
ERROR: Stored procedure/function 'deposit' does not exist.
```

```
// CALL 문에 인자가 있는 경우
CALL deposit('Tom', 3000000);
```

```
ERROR: Methods require an object as their target.
```

그리고, 아래와 같이 저장 함수/프로시저를 호출하는 **CALL** 문 안에 **CALL** 문이 중첩되는 경우와 **CALL** 문을 사용하여 저장 함수/프로시저 호출 시 인자로 서브 질의를 사용할 경우 **CALL** 문은 수행이 되지 않는다.

```
CALL phone_info('Tom', CALL sp_int(999));
CALL phone_info((SELECT * FROM Phone WHERE id='Tom'));
```

질의 최적화

조인 방식

관계형 데이터베이스는 정규화를 통해 데이터를 여러 테이블로 나누어 저장한다. 이 방법은 데이터 중복을 줄이고 데이터 무결성을 보장하는 데 효과적이다. 그러나 나누어진 데이터를 통합적으로 질의하려면 테이블 간의 관계를 결합해야 한다. 조인 연산은 이러한 관계를 결합하여 마치 단일 테이블에서 가져온 것처럼 결과를 반환한다.

조인 유형

조인 결과에 포함되는 레코드는 지정된 조인 유형에 따라 달라진다. 자세한 내용은 [조인 질의](#)에서 확인할 수 있다.

조인 방식

테이블 간 레코드를 결합하는 방식은 조인 방식에 따라 달라진다. CUBRID는 다음과 같은 조인 방식을 지원한다:

- 중첩 반복(Nested Loop) 조인
- 정렬 병합(Sort-Merge) 조인
- 해시(Hash) 조인

조인 방식은 질의 최적화기가 비용 비교를 통해 결정하며, 사용자가 직접 지정할 수도 있다. 질의 최적화기는 데이터베이스가 수집한 통계 정보를 바탕으로 각 조인 방식의 비용을 계산한 뒤 가장 낮은 비용을 가진 방식을 선택한다. 질의에 두 개 이상의 조인이 포함된 경우, 각 조인에 다른 조인 방식을 적용할 수 있다. 각 조인 방식은 한 번에 두 개의 테이블만 처리하며, 순차적으로 수행된다. 즉, 다음 조인은 이전 조인이 완료된 후에만 수행된다.

CUBRID에서는 기본적으로 **정렬 병합 조인** 또는 **해시 조인**에 대한 실행 계획이 생성되지 않는다. 질의 최적화기가 이 두 조인 방식을 고려하도록 하려면 **SQL 힌트**를 사용해야 한다. **optimizer_enable_merge_join** 파라미터를 **yes**(기본값은 **no**)로 설정하면 **USE_MERGE** 힌트를 사용하지 않고도 **정렬 병합 조인**을 수행할 수 있다. 자세한 내용은 [기타 파라미터](#)에서 확인할 수 있다.

참고

여기서 **테이블**은 물리적 테이블뿐만 아니라 뷰(가상 테이블) 및 파생 테이블을 포함하는 넓은 개념으로 사용된다.

중첩 반복(Nested Loop) 조인

중첩 반복 조인은 외부 테이블에서 레코드를 하나씩 읽은 후, 해당 레코드와 일치하는 내부 테이블의 레코드를 찾아 결합하는 방식이다. 이 방식은 프로그래밍에서 사용하는 중첩 반복문의 개념과 유사하다. 외부 반복에 사용되는 테이블을 **드라이빙 테이블(driving table)**이라고 하며, 내부 반복에 사용되는 테이블은 **드리븐 테이블(driven table)**이라고 한다.

드라이빙 테이블

두 테이블 중 하나를 드라이빙 테이블로 설정할 때는 일반적으로 레코드 수가 적은 테이블을 선택하는 것이 더 효율적이다. 드라이빙 테이블의 레코드 수가 적을수록 조인 연산 횟수가 줄어들어 전체 조인 비용이 감소한다. 랜덤 액세스 비용은 대체로 일정하므로, driven 테이블에 대한 조인 연산 횟수가 줄어들면 성능이 더욱 향상된다. 질의 최적화기는 이러한 조인 비용을 비교하여 드라이빙 테이블을 자동으로 선택한다. 사용자가 **ORDERED** 또는 **LEADING** 힌트를 사용해 드라이빙 테이블을 직접 지정할 수도 있다.

드리븐 테이블

중첩 반복 조인은 driven 테이블에 랜덤 액세스가 가능한 경우 가장 효율적으로 작동한다. 이를 위해서는 조인 조건에 적합한 인덱스가 driven 테이블에 생성되어 있어야 한다. 만약 조인 조건에 적합한 인덱스가 없다면, driven 테이블의 모든 레코드를 순차적으로 읽어야 하므로 조인 성능이 저하된다. 조인 조건에 적합한 인덱스를 통해 driven 테이블에 랜덤 액세스할 수 있으면, 실행 계획에서 중첩 반복 조인은 **인덱스 조인(idx-join)**으로 표시된다.

부분 범위 처리

중첩 반복 조인은 두 테이블에서 각각 하나의 레코드를 읽은 상태에서도 조인 연산을 수행할 수 있으며, 모든 레코드에 대해서 조인이 완료되지 않은 상태에서도 즉시 반환할 수 있는 조인 결과를 유지한다. 이러한 동작은 부분 범위 처리를 가능하게 하므로, 처음 N개의 결과만 필요한 경우 N개의 결과가 준비되는 즉시 반환할 수 있다. 부분 범위 처리를 통해 불필요한 조인 연산을 피함으로써 전체적인 조인 성능이 향상된다.

해시 리스트 스캔

드리븐 테이블이 뷰 또는 파생 테이블인 경우, 인덱스를 사용할 수 없으므로 임시 결과의 레코드를 순차적으로 읽는 리스트 스캔 방식으로 조인을 수행한다. 해시 리스트 스캔은 리스트 스캔의 한 종류로, 임시 결과에서 조인 조건을 만족하는 레코드를 랜덤 액세스하기 위해 사용된다. 리스트 스캔은 임시 결과의 레코드를 읽는 방식이므로 물리적 테이블에는 사용할 수 없다. 따라서 해시 리스트 스캔

도 트리본 테이블이 뷰 또는 파생 테이블인 경우에만 사용된다. 해시 리스트 스캔은 **해시 조인**과 동작 방식이 유사하며, 조인 조건이 모두 동등 비교 조건일 때만 사용할 수 있다. 해시 리스트 스캔이 사용되었는지는 **질의 프로파일링**을 통해 확인할 수 있다.

해시 리스트 스캔과 해시 조인의 차이점은 다음 두 가지이다:

1. 해시 리스트 스캔은 해시 테이블 빌드에 트리본 테이블만 사용한다.
2. 해시 리스트 스캔은 중첩 반복 조인의 일부로 사용되므로 때문에 부분 범위 처리가 가능하다.

질의 최적화기는 조인 비용을 계산할 때 해시 리스트 스캔의 수행 여부는 반영하지 않는다.

예시 1. 인덱스 스캔을 수행하지 않는 중첩 반복 조인

이 예시에서는 중첩 반복 조인 수행 시 트리본 테이블에서 인덱스 스캔이 수행되지 않는다.

ORDERED 힌트를 사용하여 조인 순서를 고정했으며, 그 결과 nation 테이블이 드라이빙 테이블이 되고, athlete 테이블이 트리본 테이블이 된다. 트리본 테이블에 조인 조건에 적합한 인덱스가 없기 때문에, nation 테이블의 각 레코드에 대해 athlete 테이블의 모든 레코드를 순차적으로 읽어야 한다.

```
select /*+ recompile ordered use_nl */
      n.name, a.name
from
      nation as n
      inner join athlete as a on n.code = a.nation_code
where
      n.continent = 'Asia'
      and a.event = 'Baseball';
```

Query plan:

```
nl-join (inner join)
  edge: term[0]
  outer: sscan
        class: n node[0]
        sargs: term[2]
        cost: 2 card 31
  inner: sscan
        class: a node[1]
        sargs: term[0] AND term[1]
        cost: 49 card 202
  cost: 4697 card 29
```

Trace Statistics:

```
SELECT (time: 92, fetch: 1720, fetch_time: 0, ioread: 0)
SCAN (table: public.nation), (heap time: 0, fetch: 216,
ioread: 0, readrows: 215, rows: 47)
```

```
SCAN (table: public.athlete), (heap time: 92, fetch:
1504, ioread: 0, readrows: 313819, rows: 103)
```

수행 과정:

1. nation 테이블에서 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드를 읽는다.
2. athlete 테이블에서 필터 조건(`a.event = 'Baseball'`)과 조인 조건(`n.code = a.nation_code`)을 만족하는 레코드를 읽는다.
3. 조건이 만족되면 두 레코드를 결합해 결과에 추가한다.
4. athlete 테이블에서 다음 레코드를 읽은 후 **2단계**를 반복한다.
5. nation 테이블에서 다음 레코드를 읽은 후 **1~3단계**를 반복한다.
6. 모든 레코드를 처리할 때까지 이 과정을 반복한다.

예시 2. 인덱스를 사용한 중첩 반복 조인

이 예시에서는 드리븐 테이블에서 인덱스 스캔을 사용하는 중첩 반복 조인을 수행한다. 드리븐 테이블(athlete)에 조인 조건에 적합한 인덱스를 생성한다. 그 결과, nation 테이블의 각 레코드에 대해 athlete 테이블의 모든 레코드를 읽을 필요 없이 랜덤 액세스가 가능하다.

```
create index i_athlete_nation_code on athlete (nation_code);

select /** recompile ordered use_nl */
  n.name, a.name
from
  nation as n
  inner join athlete as a on n.code = a.nation_code
where
  n.continent = 'Asia'
  and a.event = 'Baseball';

drop index i_athlete_nation_code on athlete;
```

Query plan:

```
idx-join (inner join)
  outer: sscan
        class: n node[0]
        sargs: term[2]
        cost:  2 card 31
  inner: iscan
        class: a node[1]
        index: i_athlete_nation_code term[0]
        sargs: term[1]
```

```
cost: 4 card 202
cost: 68 card 29
```

```
Trace Statistics:
  SELECT (time: 1, fetch: 1265, fetch_time: 0, ioread: 0)
    SCAN (table: public.nation), (heap time: 1, fetch: 216,
ioread: 0, readrows: 215, rows: 47)
      SCAN (index: public.athlete.i_athlete_nation_code),
(btrees time: 0, fetch: 1045, ioread: 0, readkeys: 24,
filteredkeys: 0, rows: 948) (lookup time: 0, rows: 103)
```

예시 1과 비교했을 때, 조인 조건을 비교하기 전 athlete 테이블에서 읽는 레코드 수가 크게 감소했다 (readrows: 313,819 -> rows: 948).

예시 3. 드라이빙 테이블에 따른 성능 차이

이 예시에서는 이전 두 예시와 달리 드라이빙 테이블을 athlete로 변경한다. **LEADING** 힌트를 사용해 조인 순서를 지정했으며, 결과적으로 athlete 테이블이 드라이빙 테이블, nation 테이블이 드리븐 테이블이 된다.

```
select /*+ recompile leading(a, n) use_nl */
  n.name, a.name
from
  nation as n
  inner join athlete as a on n.code = a.nation_code
where
  n.continent = 'Asia'
  and a.event = 'Baseball';
```

Query plan:

```
idx-join (inner join)
  outer: sscan
    class: a node[0]
    sargs: term[1]
    cost: 49 card 202
  inner: iscan
    class: n node[1]
    index: pk_nation_code term[0]
    sargs: term[2]
    cost: 2 card 31
cost: 161 card 29
```

```
Trace Statistics:
  SELECT (time: 5, fetch: 7168, fetch_time: 0, ioread: 0)
    SCAN (table: public.athlete), (heap time: 5, fetch:
```

```
6709, ioread: 0, readrows: 6677, rows: 229)
      SCAN (index: public.nation.pk_nation_code), (btree
time: 0, fetch: 458, ioread: 0, readkeys: 229,
filteredkeys: 0, rows: 229) (lookup time: 0, rows: 103)
```

athlete 테이블을 드라이빙 테이블로 사용하면 레코드 수가 많아 성능이 저하될 수 있다.

예시 4. 해시 리스트 스캔이 포함된 중첩 반복 조인

이 예시에서는 드리븐 테이블에서 해시 리스트 스캔이 수행되는 중첩 반복 조인을 보여준다.

ORDERED 및 **NO_MERGE** 힌트를 사용해 인덱스 스캔이 불가능한 인라인 뷰를 드리븐 테이블로 설정한다. 또한, 질의는 상위 20개의 결과만 조회해 해시 리스트 스캔과 부분 범위 처리를 함께 수행한다.

```
select /*+ recompile ordered use_nl */
      n.name, v.name
from
      nation as n
      inner join (
            select /*+ no_merge */
                  a.name, a.nation_code
            from
                  athlete as a
            where
                  a.event = 'Baseball'
        ) as v on n.code = v.nation_code
where
      n.continent = 'Asia'
limit 20;
```

Query plan:

```
nl-join (inner join)
  edge: term[0]
  outer: sscan
        class: n node[0]
        sargs: term[1] AND term[2]
        cost: 2 card 31
  inner: sscan
        class: v node[1]
        sargs: term[0]
        cost: 2 card 202
  sargs: term[1]
  cost: 148 card 29
```

Trace Statistics:

```
SELECT (time: 2, fetch: 33, fetch_time: 0, ioread: 0)
      SCAN (table: public.nation), (heap time: 0, fetch: 1,
```

```
ioread: 0, readrows: 26, rows: 8)
  SCAN (hash temp(m), build time: 0, time: 0, fetch: 0,
ioread: 0, readrows: 249, rows: 20)
  SUBQUERY (uncorrelated)
    SELECT (time: 2, fetch: 32, fetch_time: 0, ioread: 0)
      SCAN (table: public.athlete), (heap time: 2, fetch:
32, ioread: 0, readrows: 6677, rows: 229)
```

실제로 해시 리스트 스캔이 드리븐 테이블에서 수행되었는지는 실행 계획에서 확인할 수 없지만, 질의 프로파일링을 통해 확인할 수 있다.

정렬 병합(Sort-Merge) 조인

정렬 병합 조인은 두 테이블의 데이터를 조인 조건에 따라 정렬한 뒤, 정렬된 데이터를 순차적으로 비교하며 결합하는 방식이다. 이 방식은 조인 조건이 동등 조건일 때만 수행할 수 있다.

조인 전에 두 테이블의 데이터를 정렬하는 과정은 실시간으로 인덱스를 생성하는 것과 유사하다. 하지만 조인 조건에 적합한 인덱스가 존재하더라도 정렬 과정은 생략되지 않는다. 정렬된 데이터를 결합하는 과정은, 정렬된 두 임시 결과에서 각각 하나의 레코드를 읽으며 조인 연산을 수행한다는 점에서 중첩 반복 조인과 동작 방식이 비슷하다. 정렬 과정에서 사용되는 버퍼 크기는 **sort_buffer_size** 파라미터로 설정할 수 있으며, 자세한 내용은 **메모리 관련 파라미터**에서 확인할 수 있다.

조인 비용

정렬 병합 조인은 정렬 단계에서 비용이 크게 발생할 경우 비효율적일 수 있다. 질의 최적화기가 조인 비용을 비교할 때, 중첩 반복 조인은 사전 작업이 필요하지 않고, 해시 조인은 두 테이블 중 하나의 데이터만 사용해 해시 테이블을 빌드하므로 상대적으로 비용이 낮게 평가될 수 있다. 반면, 정렬 병합 조인은 두 테이블의 데이터를 모두 정렬해야 하므로 더 높은 비용으로 평가된다.

부분 범위 처리

정렬 병합 조인은 두 테이블의 정렬이 완료된 후에만 조인을 수행할 수 있으므로 부분 범위 처리가 불가능하다. 따라서 처음 N개의 결과만 필요한 경우에도 조인 작업이 모두 완료된 후에야 결과를 조회할 수 있다.

예시 1. 정렬 병합 조인의 수행 과정

이 예시는 정렬 병합 조인을 수행하는 과정을 보여준다. **ORDERED** 힌트는 조인 성능에 직접적인 영향을 미치지 않지만, 여기서는 조인이 아래 지정된 순서대로 실행되도록 하기 위해 사용되었다.

```
create index i_nation_continent_code on nation (continent,
code);
create index i_athlete_event_nation_code on athlete (event,
nation_code);

select /** recompile ordered use_merge */
```

```

    n.name, a.name
from
    nation as n
    inner join athlete as a on n.code = a.nation_code
where
    n.continent = 'Asia'
    and a.event = 'Baseball';

drop index i_nation_continent_code on nation;
drop index i_athlete_event_nation_code on athlete;

```

Query plan:

```

temp
  order: UNORDERED
  subplan: m-join (inner join)
    edge: term[0]
    outer: temp
      order: code[0]
      subplan: iscan
        class: n node[0]
        index:
i_nation_continent_code term[1]
      cost: 2 card 1
    cost: 8 card 1
  inner: temp
    order: nation_code[1]
    subplan: iscan
      class: a node[1]
      index:
i_athlete_event_nation_code term[2]
    cost: 2 card 7
  cost: 8 card 7
cost: 17 card 1
cost: 23 card 1

```

Trace Statistics:

```

  SELECT (time: 1, fetch: 287, fetch_time: 1, ioread: 0)
    SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 103,
rows: 103)
      MERGELIST
        SELECT (time: 0, fetch: 52, fetch_time: 0, ioread:
0)
          SCAN (index:
public.nation.i_nation_continent_code), (btree time: 0,
fetch: 49, ioread: 0, readkeys: 47, filteredkeys: 0, rows:
47) (lookup time: 0, rows: 47)
            ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
              SELECT (time: 1, fetch: 235, fetch_time: 1, ioread:

```

```
0)
      SCAN (index:
public.athlete.i_athlete_event_nation_code), (btree time:
1, fetch: 231, ioread: 0, readkeys: 6, filteredkeys: 0,
rows: 229) (lookup time: 1, rows: 229)
      ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```

수행 과정:

1. nation 테이블에서 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드를 가져와 code 컬럼 값으로 정렬한다.

```
select n.code, n.name from nation as n where n.continent =
'Asia' order by n.code;
```

1. athlete 테이블에서 필터 조건(`a.event = 'Baseball'`)을 만족하는 레코드를 가져와 nation_code 컬럼 값으로 정렬한다.

```
select a.nation_code, a.name from athlete as a where
a.event = 'Baseball' order by a.nation_code;
```

1. 두 정렬된 결과에서 레코드를 하나씩 순차적으로 읽고, 조인 조건(`n.code = a.nation_code`)을 비교한다. 조건이 만족되면 두 레코드를 결합하여 결과에 추가한다.
2. 모든 레코드가 처리될 때까지 이 과정을 반복하여 조인을 완료한다.

해시(Hash) 조인

해시 조인은 해시 함수를 사용해 동일한 해시 값을 가지는 두 테이블의 레코드를 결합하는 방식이다. 동일한 해시 값을 가지는 레코드를 효율적으로 비교하기 위해 해시 테이블이 사용된다. 두 테이블 중 하나는 해시 테이블을 **빌드(Build)**하는 데 사용되고, 다른 하나는 해시 테이블을 **프로브(Probe)**하는 데 사용된다. 해시 값은 조인 컬럼 값을 입력으로 사용하는 해시 함수에 의해 생성된다. 다른 입력 값에서 동일한 해시 값이 생성될 경우 이를 해시 충돌이라고 한다. 해시 충돌이 발생하면 동일한 해시 값을 가진 레코드를 순차적으로 조사하며 조인 조건을 확인한다. 해시 조인은 모든 조인 조건이 동등 비교 조건인 경우에만 수행할 수 있다.

SQL 힌트

해시 조인과 관련된 힌트는 두 가지가 있다:

- **USE_HASH**: 질의 최적화기가 해시 조인에 대한 실행 계획을 생성할 수 있도록 허용한다.

- **NO_USE_HASH**: 질의 최적화기가 해시 조인에 대한 실행 계획을 생성하지 못하도록 방지한다.

두 힌트 모두 하나 이상의 테이블을 지정해서 사용할 수 있다. 힌트에 특정 테이블이 명시된 경우, 해당 힌트는 내부 테이블(드리븐 테이블)에 적용된다. 테이블이 명시되지 않은 경우, 힌트는 모든 테이블에 적용된다. 동일한 테이블에 대해 **USE_HASH**와 **NO_USE_HASH** 힌트가 모두 사용된 경우, **NO_USE_HASH** 힌트가 우선 적용된다.

비용 계산

해시 조인에 참여하는 각 테이블은 해시 테이블을 빌드하거나 프로브하는 데 사용될 수 있다. 따라서 두 가지 경우의 비용을 계산하여 비교한 뒤, 더 낮은 비용을 해시 조인의 최종 비용으로 결정한다. 예를 들어, 다음 두 가지 경우를 비교하여 비용이 낮은 방식을 선택한다:

- 외부 테이블(드라이빙 테이블)로 해시 테이블을 빌드하고, 내부 테이블(드리븐 테이블)로 이를 프로브하는 경우.
- 내부 테이블(드리븐 테이블)로 해시 테이블을 빌드하고, 외부 테이블(드라이빙 테이블)로 이를 프로브하는 경우.

해시 테이블을 빌드하는 데 사용되는 테이블

해시 테이블을 빌드할 테이블은 **ORDERED** 또는 **LEADING** 힌트를 사용해 지정할 수 없다. 일반적으로 레코드 수가 적은 테이블이 해시 테이블 빌드에 사용된다. 그러나 외부 조인의 경우, 조인 방향에 따라 해시 테이블을 빌드할 테이블이 고정된다:

- **왼쪽 외부 조인**: 오른쪽 테이블이 해시 테이블 빌드에 사용된다.
- **오른쪽 외부 조인**: 왼쪽 테이블이 해시 테이블 빌드에 사용된다.

해시 테이블을 빌드하는 방식

해시 테이블을 빌드하는 방식은 임시 결과 크기와 해시 테이블의 최대 크기에 따라 결정된다. 해시 테이블의 최대 크기는 **max_hash_list_scan_size** 파라미터를 통해 설정할 수 있다. 자세한 내용은 [메모리 관련 파라미터](#)에서 확인할 수 있다.

CUBRID는 다음과 같은 해시 방식을 지원한다:

- **메모리 해시**: 임시 결과를 모두 해시 테이블에 저장할 수 있는 경우 사용된다.

임시 결과 페이지 수 * 페이지 크기 <= 해시 테이블 최대 크기

- **하이브리드 해시**: 해시 테이블이 임시 결과 레코드의 위치를 저장할 수 있는 경우 사용된다.

임시 결과 레코드 수 * (해시 엔트리 크기 + 위치 정보 크기) <= 해시 테이블 최대 크기

- **파일 해시**: 위 두 방식 모두 사용할 수 없는 경우 사용된다.
- **스킵 해시**: 두 임시 결과 중 하나가 비어 있거나 불필요한 경우(외부 조인 시), 해시 테이블 빌드는 생략된다.

하이브리드 해시와 파일 해시는 레코드 위치를 해시 테이블에 저장하므로, 레코드를 읽기 전에 위치를 찾는 과정이 추가로 필요하다.

메모리 해시와 하이브리드 해시는 메모리 페이지를 사용해 해시 테이블을 빌드하지만, **파일 해시**는 임시 볼륨의 페이지를 사용해 해시 테이블을 빌드한다. 임시 볼륨의 페이지 접근 시 논리적 I/O가 추가로 발생하며, 이는 성능 저하를 초래할 수 있다.

질의를 실행하는 동안, 조인이 완료되면 해시 테이블은 삭제되고 사용된 메모리는 반환된다. 해시 조인을 수행할 때 실제로 사용된 방식은 질의 프로파일링을 통해 확인할 수 있으며, 자세한 내용은 [질의 프로파일링](#)에서 확인할 수 있다.

부분 범위 처리

해시 조인은 해시 테이블 빌드가 완료된 이후에만 조인이 진행되므로, 부분 범위 처리는 불가능하다.

예시 1. 해시 조인의 수행 과정

이 예시는 해시 조인을 수행하는 과정을 설명한다. **ORDERED** 힌트를 사용하더라도 해시 테이블을 빌드할 테이블은 지정할 수 없다.

```
select /*+ recompile ordered use_hash */
      n.name, a.name
from
      nation as n
      inner join athlete as a on n.code = a.nation_code
where
      n.continent = 'Asia'
      and a.event = 'Baseball';
```

Query plan:

```
hash-join (inner join)
  edge: term[0]
  outer: sscan
         class: n node[0]
         sargs: term[2]
         cost: 2 card 31
  inner: sscan
         class: a node[1]
```

```
sargs: term[1]
cost: 49 card 202
cost: 63 card 29
```

```
Trace Statistics:
  SELECT (time: 2, fetch: 33, fetch_time: 0, ioread: 0)
    SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 103,
rows: 103)
      HASHJOIN (time: 2, fetch: 33, fetch_time: 0, ioread:
0)
        BUILD (time: 0, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: memory)
          SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
            SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 47)
              PROBE (time: 2, probe_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, readkeys: 103, rows: 103,
max_collisions: 1)
                SELECT (time: 2, fetch: 32, fetch_time: 0,
ioread: 0)
                  SCAN (table: public.athlete), (heap time: 1,
fetch: 32, ioread: 0, readrows: 6677, rows: 229)
```

nation 테이블에서는 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드가 47개이고, athlete 테이블에서는 필터 조건(`a.event = 'Baseball'`)을 만족하는 레코드가 229개다. 따라서 더 적은 레코드를 가진 nation 테이블이 해시 테이블을 빌드하는 데 사용된다.

질의 프로파일링 결과:

여기서는 해시 조인과 관련된 항목들만 설명한다. 다른 항목들에 대해서는 [질의 프로파일링](#)에서 확인할 수 있다.

1. HASHJOIN

- **time**: 해시 조인 전체 실행 시간(ms)
- **fetch**: 해시 조인에서 fetch한 페이지 수
- **fetch_time**: 해시 조인에서 페이지를 fetch하는 데 걸린 시간(ms)
- **ioread**: 해시 조인에서 페이지를 fetch하는 동안 발생한 물리적 I/O 읽기 횟수

1. **BUILD**: **time**을 제외하고, 나머지 항목들은 임시 결과를 조회할 때 누적된 정보를 포함하지 않는다.

- **time**: 임시 결과 조회를 포함한 해시 테이블 빌드 시간(ms)

- **build_time**: 해시 테이블 빌드 시간(ms)
- **fetch**: 해시 테이블 빌드를 위해 fetch한 페이지 수
- **fetch_time**: 해시 테이블 빌드에서 페이지를 fetch하는 데 걸린 시간(ms)
- **ioread**: 해시 테이블 빌드에서 발생한 물리적 I/O 읽기 횟수
- **hash_method**: 해시 테이블 빌드에 사용된 방식 (**memory**, **hybrid**, **file**, 또는 **skip**)
 - **memory**: 메모리 해시 방식 사용
 - **hybrid**: 하이브리드 해시 방식 사용
 - **file**: 파일 해시 방식 사용
 - **skip**: 해시 테이블 빌드를 생략

1. **PROBE: time**을 제외하고, 나머지 항목들은 임시 결과를 조회할 때 누적된 정보를 포함하지 않는다.

- **time**: 임시 결과 조회를 포함한 해시 테이블 프로브 시간(ms)
- **probe_time**: 해시 테이블 프로브 시간(ms)
- **fetch**: 해시 테이블 프로브를 위해 fetch한 페이지 수
- **fetch_time**: 해시 테이블 프로브에서 페이지를 fetch하는 데 걸린 시간(ms)
- **ioread**: 해시 테이블 프로브에서 발생한 물리적 I/O 읽기 횟수
- **readkeys**: 해시 테이블 프로브 중 해시 값이 일치한 레코드 수
- **rows**: 해시 테이블 프로브 중 조인 조건을 만족하는 레코드 수 (**readkeys** $\geq$ **rows**)
- **max_collisions**: 해시 테이블 프로브 중 하나의 해시 값에 대해 읽은 최대 레코드 수

수행 과정:

1. nation 테이블에서 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드를 가져와 임시 결과를 생성한다.

```
select n.code, n.name from nation as n where
n.continent = 'Asia';
```

1. athlete 테이블에서 필터 조건(`a.event = 'Baseball'`)을 만족하는 레코드를 가져와 임시 결과를 생성한다.

```
select a.nation_code, a.name from athlete as a
where a.event = 'Baseball';
```

1. nation 테이블의 임시 결과를 읽어 해시 테이블을 빌드한다.
2. athlete 테이블의 임시 결과에서 각 레코드에 대해 nation 테이블의 해시 테이블을 프로브한다.
3. 동일한 해시 값을 가진 레코드를 찾아(`readkeys`) 조인 조건(`n.code = a.nation_code`)을 비교한다. 조건이 만족(`rows`)되면 두 레코드를 결합하여 결과에 추가한다.
4. 동일한 해시 값을 가진 다음 레코드에 대해 5단계를 반복한다.
5. athlete 테이블의 다음 레코드에 대해 4~6단계를 반복한다.
6. 모든 레코드가 처리될 때까지 이 과정을 계속 반복하여 조인을 완료한다.

예시 2. 힌트를 사용한 해시 조인

이 예시는 **USE_HASH** 및 **NO_USE_HASH** 힌트를 사용해 특정 테이블에 대해 해시 조인을 수행하거나 수행하지 않는 방법을 보여준다. **ORDERED** 힌트를 사용하여 내부 테이블(드리븐 테이블)을 고정한다.

```
select /* recompile ordered use_hash no_use_hash(r) */
      n.name, a.name, a.code, r.medal
from
  nation as n
  inner join athlete as a on n.code = a.nation_code
  inner join record as r on a.code = r.athlete_code
where
  n.continent = 'Asia'
  and a.event = 'Shooting'
  and r.host_year = 1992;
```

Query plan:

```
idx-join (inner join)
  outer: hash-join (inner join)
    edge: term[0]
    outer: sscan
```

```

                                class: n node[0]
                                sargs: term[3]
                                cost: 2 card 31
        inner: sscan
                                class: a node[1]
                                sargs: term[2]
                                cost: 49 card 202
        cost: 63 card 29
    inner: iscan
        class: r node[2]
        index:
pk_record_host_year_event_code_athlete_code_medal term[4]
(covers)
        filtr: term[1]
        cost: 5 card 400
    cost: 109 card 2

```

USE_HASH 힌트를 통해 nation과 athlete 테이블에 대해 해시 조인이 수행된다.

NO_USE_HASH 힌트를 사용하여 record 테이블에서는 해시 조인이 수행되지 않는다.

```

Trace Statistics:
  SELECT (time: 5, fetch: 139, fetch_time: 0, ioread: 0)
    SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 35,
rows: 35)
      SCAN (index:
public.record.pk_record_host_year_event_code_athlete_code_me
dal), (btree time: 3, fetch: 105, ioread: 0, readkeys: 7,
filteredkeys: 0, rows: 7, covered: true)
        HASHJOIN (time: 2, fetch: 33, fetch_time: 0, ioread:
0)
          BUILD (time: 1, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: memory)
            SELECT (time: 1, fetch: 1, fetch_time: 0, ioread:
0)
              SCAN (table: public.nation), (heap time: 1,
fetch: 1, ioread: 0, readrows: 215, rows: 47)
                PROBE (time: 1, probe_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, readkeys: 35, rows: 35,
max_collisions: 1)
                  SELECT (time: 1, fetch: 32, fetch_time: 0,
ioread: 0)
                    SCAN (table: public.athlete), (heap time: 1,
fetch: 32, ioread: 0, readrows: 6677, rows: 173)

```

예시 3. 인덱스 스킵 스캔을 수행하는 해시 조인

이 예시는 해시 조인을 수행할 때 인덱스 스킵 스캔을 사용하여 임시 결과를 가져오는 방법을 보여 준다.

```

/* for nested loop join */
create index i_athlete_nation_code_event on athlete
(nation_code, event);

/* for hash join */
create index i_nation_continent_code on nation (continent,
code);

select /** recompile use_hash index_ss */
      n.name, a.name
from
      nation as n
      inner join athlete as a on n.code = a.nation_code
where
      n.continent = 'Asia'
      and a.event = 'Baseball';

drop index i_athlete_nation_code_event on athlete;
drop index i_nation_continent_code on nation;

```

Query plan:

```

hash-join (inner join)
  edge: term[0]
  outer: iscan
        class: n node[0]
        index: i_nation_continent_code term[1]
        cost: 2 card 1
  inner: iscan
        class: a node[1]
        index: i_athlete_nation_code_event term[2]
(index skip scan)
      cost: 13 card 202
cost: 25 card 1

```

Trace Statistics:

```

SELECT (time: 1, fetch: 748, fetch_time: 0, ioread: 0)
  SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 103,
rows: 103)
    HASHJOIN (time: 1, fetch: 748, fetch_time: 0, ioread:
0)
      BUILD (time: 0, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: memory)
        SELECT (time: 0, fetch: 52, fetch_time: 0,
ioread: 0)
          SCAN (index:
public.nation.i_nation_continent_code), (btree time: 0,
fetch: 49, ioread: 0, readkeys: 47, filteredkeys: 0, rows:

```

```

47) (lookup time: 0, rows: 47)
      PROBE (time: 1, probe_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, readkeys: 103, rows: 103,
max_collisions: 1)
      SELECT (time: 1, fetch: 696, fetch_time: 0,
ioread: 0)
      SCAN (index:
public.athlete.i_athlete_nation_code_event), (btree time:
1, fetch: 692, ioread: 0, readkeys: 121, filteredkeys: 0,
rows: 229) (lookup time: 0, rows: 229)

```

예시 4. 해시 오른쪽 외부 조인

이 예시는 외부 조인의 방향에 따라 해시 테이블을 빌드하는 테이블이 결정되는 방식을 보여준다.

```

select /*+ recompile use_hash */
  n.name, p.host_year, p.gold
from
  participant as p
  right outer join nation as n on n.code = p.nation_code
and p.gold > 0
where
  n.continent = 'Asia';

```

Query plan:

```

hash-join (right outer join)
  edge: term[0]
  outer: sscan
        class: p node[0]
        sargs: term[1]
        cost: 5 card 92
  inner: sscan
        class: n node[1]
        sargs: term[2]
        cost: 2 card 31
cost: 15 card 31

```

Trace Statistics:

```

SELECT (time: 1, fetch: 4, fetch_time: 0, ioread: 0)
  SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 70,
rows: 70)
  HASHJOIN (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
    BUILD (time: 0, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: memory)
      SELECT (time: 0, fetch: 3, fetch_time: 0, ioread:
0)
        SCAN (table: public.participant), (heap time:

```

```

0, fetch: 3, ioread: 0, readrows: 916, rows: 228)
    PROBE (time: 0, probe_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, readkeys: 36, rows: 70,
max_collisions: 5)
    SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
    SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 47)

```

participant 테이블에는 필터 조건(`p.gold > 0`)을 만족하는 레코드가 228개 있고, nation 테이블에는 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드가 47개 있다. participant 테이블의 레코드 수가 더 많지만, 우측 외부 조인의 경우 왼쪽 테이블(participant)이 해시 테이블을 빌드한다.

예시 5. 해시 테이블 빌드를 생략한 해시 외부 조인

이 예시는 해시 외부 조인에서 해시 테이블 빌드를 생략하는 경우를 보여준다. 왼쪽 외부 조인의 경우 오른쪽 테이블이 해시 테이블을 빌드한다. 그러나 participant 테이블에 필터 조건(`p.gold > 100`)을 만족하는 레코드가 없으므로 해시 테이블 빌드가 생략된다. 대신 nation 테이블에서 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드가 질의 결과로 반환된다.

```

select /** recompile use_hash */
    n.name, p.host_year, p.gold
from
    nation as n
    left outer join participant as p on n.code =
p.nation_code and p.gold > 100
where
    n.continent = 'Asia';

```

Query plan:

```

hash-join (left outer join)
  edge: term[0]
  outer: sscan
        class: n node[0]
        sargs: term[2]
        cost: 2 card 31
  inner: sscan
        class: p node[1]
        sargs: term[1]
        cost: 5 card 92
cost: 15 card 31

```

Trace Statistics:

```

SELECT (time: 1, fetch: 4, fetch_time: 0, ioread: 0)
  SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 47,
rows: 47)

```



```

HASHJOIN (time: 1, fetch: 4, fetch_time: 0, ioread: 0)
  BUILD (time: 1, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: skip)
    SELECT (time: 1, fetch: 3, fetch_time: 0, ioread:
0)
      SCAN (table: public.participant), (heap time:
1, fetch: 3, ioread: 0, readrows: 916, rows: 0)
        PROBE (time: 0, probe_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, readkeys: 0, rows: 47,
max_collisions: 0)
          SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
            SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 47)

```

BUILD 항목에서는 해시 테이블 빌드 방식이 `hash_method: skip` 으로 표시된다. `skip` 은 해시 테이블 빌드가 생략되었음을 의미한다. 대신 nation 테이블에서 필터 조건(`n.continent = 'Asia'`)을 만족하는 레코드가 질의 결과로 반환된다.

예시 6. 하이브리드 해시 방식을 사용한 해시 조인

이 예시는 하이브리드 해시 조인을 수행하는 방법을 보여준다. 예시 데이터의 양이 적기 때문에, 하이브리드 해시 방식을 사용하려면 해시 테이블의 최대 크기를 낮춰야 한다. 이를 위해 `max_hash_list_scan_size` 파라미터를 **12K**(기본값: **8M**)로 설정한다.

```

set system parameters 'max_hash_list_scan_size=12k';

select /** recompile use_hash */
  count (*)
from
  nation as n
  inner join athlete as a on n.code = a.nation_code;

set system parameters 'max_hash_list_scan_size=8M';

```

Query plan:

```

hash-join (inner join)
  edge: term[0]
  outer: sscan
          class: n node[0]
          cost: 2 card 215
  inner: sscan
          class: a node[1]
          cost: 49 card 6677
  cost: 400 card 6677

```

```
Trace Statistics:
  SELECT (time: 5, fetch: 126, fetch_time: 0, ioread: 0)
    SCAN (temp time: 0, fetch: 19, ioread: 0, readrows:
6677, rows: 6677)
      HASHJOIN (time: 5, fetch: 105, fetch_time: 0, ioread:
0)
        BUILD (time: 0, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: hybrid)
          SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
            SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 215)
              PROBE (time: 5, probe_time: 2, fetch: 49,
fetch_time: 0, ioread: 0, readkeys: 6692, rows: 6677,
max_collisions: 2)
                SELECT (time: 3, fetch: 53, fetch_time: 0,
ioread: 0)
                  SCAN (table: public.athlete), (heap time: 2,
fetch: 32, ioread: 0, readrows: 6677, rows: 6677)
```

동일한 테이블에 하나 이상의 조인 힌트를 사용하면, 질의 최적화기는 지정된 조인 방식들의 비용을 비교해 더 비용이 적은 방식을 선택한다. 생성된 실행 계획을 통해 질의 최적화기가 선택한 조인 방식을 확인할 수 있다. 다음은 **USE_HASH**와 **USE_NL** 힌트를 함께 사용했을 때, 질의 최적화기가 하이브리드 해시 방식을 선택하여 해시 조인으로 실행 계획을 생성하고 수행한 예시이다.

```
set system parameters 'max_hash_list_scan_size=12k';

select /** recompile use_hash use_nl */
  count (*)
from
  nation as n
  inner join athlete as a on n.code = a.nation_code;

set system parameters 'max_hash_list_scan_size=8M';
```

Query plan:

```
hash-join (inner join)
  edge: term[0]
  outer: sscan
        class: n node[0]
        cost: 2 card 215
  inner: sscan
        class: a node[1]
        cost: 49 card 6677
cost: 400 card 6677
```

```

Trace Statistics:
  SELECT (time: 5, fetch: 126, fetch_time: 0, ioread: 0)
    SCAN (temp time: 1, fetch: 19, ioread: 0, readrows:
6677, rows: 6677)
      HASHJOIN (time: 4, fetch: 105, fetch_time: 0, ioread:
0)
        BUILD (time: 0, build_time: 0, fetch: 0,
fetch_time: 0, ioread: 0, hash_method: hybrid)
          SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
            SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 215)
              PROBE (time: 4, probe_time: 2, fetch: 49,
fetch_time: 0, ioread: 0, readkeys: 6692, rows: 6677,
max_collisions: 2)
                SELECT (time: 2, fetch: 53, fetch_time: 0,
ioread: 0)
                  SCAN (table: public.athlete), (heap time: 2,
fetch: 32, ioread: 0, readrows: 6677, rows: 6677)

```

예시 7. 파일 해시 방식을 사용한 해시 조인

이 예시는 파일 해시 방식을 사용하여 해시 조인을 수행하는 방법을 보여준다. 예시 데이터의 양이 적기 때문에, 파일 해시 방식을 사용하려면 해시 테이블의 최대 크기를 **12K**보다 더 낮춰야 한다. 이를 위해 **max_hash_list_scan_size** 파라미터를 **8K**(기본값: **8M**)로 설정한다.

```

set system parameters 'max_hash_list_scan_size=8k';

select /*+ recompile use_hash */
  count (*)
from
  nation as n
  inner join athlete as a on n.code = a.nation_code;

set system parameters 'max_hash_list_scan_size=8M';

```

Query plan:

```

hash-join (inner join)
  edge: term[0]
  outer: sscan
        class: n node[0]
        cost: 2 card 215
  inner: sscan
        class: a node[1]
        cost: 49 card 6677
cost: 3846 card 6677

```

질의 최적화기는 데이터베이스가 수집한 통계를 바탕으로, 다음 계산을 통해 메모리 해시 방식 또는 하이브리드 해시 방식의 사용 가능 여부를 평가한다:

```

임시 결과의 레코드 수 * (해시 엔트리 크기 + 위치 정보 크기) <= 해시
테이블 최대 크기
215 * (32 byte * 16 byte) <=
8,192 byte
10,320 <=
8192 byte (False)

```

계산 결과가 설정된 메모리 값보다 커서 메모리나 하이브리드 방식 대신 파일 해시 방식이 선택된다. 파일 해시 방식(cost: 3,846)은 임시 볼륨 페이지를 사용하므로 다른 해시 방식(cost: 400)보다 비용이 더 높게 평가된다.

```

Trace Statistics:
  SELECT (time: 10, fetch: 27510, fetch_time: 3, ioread: 0)
    SCAN (temp time: 0, fetch: 19, ioread: 0, readrows:
6677, rows: 6677)
      HASHJOIN (time: 10, fetch: 27489, fetch_time: 3,
ioread: 0)
        BUILD (time: 1, build_time: 1, fetch: 645,
fetch_time: 1, ioread: 0, hash_method: file)
          SELECT (time: 0, fetch: 1, fetch_time: 0, ioread:
0)
            SCAN (table: public.nation), (heap time: 0,
fetch: 1, ioread: 0, readrows: 215, rows: 215)
              PROBE (time: 9, probe_time: 7, fetch: 26772,
fetch_time: 2, ioread: 0, readkeys: 6692, rows: 6677,
max_collisions: 2)
                SELECT (time: 2, fetch: 53, fetch_time: 0,
ioread: 0)
                  SCAN (table: public.athlete), (heap time: 1,
fetch: 32, ioread: 0, readrows: 6677, rows: 6677)

```

예시 6의 질의 프로파일링 결과와 비교했을 때, 파일 해시 방식이 비용과 비례해서 수행시간이 느린 것을 확인할 수 있다.

동일한 테이블에 하나 이상의 조인 힌트를 사용하면, 질의 최적화기는 지정된 조인 방식들의 비용을 비교해 더 비용이 적은 방식을 선택한다. 생성된 실행 계획을 통해 질의 최적화기가 선택한 조인 방식을 확인할 수 있다. 다음은 **USE_HASH**와 **USE_NL** 힌트를 함께 사용했을 때, 파일 해시 방식의 해시 조인 비용(cost: 3,846)이 높아 질의 최적화기가 중첩 반복 조인(cost: 3,405)으로 실행 계획을 생성하고 수행한 예시이다.

```

set system parameters 'max_hash_list_scan_size=8k';

select /** recompile use_hash use_nl */
count (*)

```

```

from
  nation as n
  inner join athlete as a on n.code = a.nation_code;

set system parameters 'max_hash_list_scan_size=8M';

```

Query plan:

```

idx-join (inner join)
  outer: sscan
        class: a node[1]
        cost: 49 card 6677
  inner: iscan
        class: n node[0]
        index: pk_nation_code term[0] (covers)
        cost: 2 card 215
cost: 3405 card 6677

```

Trace Statistics:

```

SELECT (time: 19, fetch: 13387, fetch_time: 1, ioread: 0)
  SCAN (table: public.athlete), (heap time: 4, fetch:
6709, ioread: 0, readrows: 6677, rows: 6677)
  SCAN (index: public.nation.pk_nation_code), (btree
time: 10, fetch: 6677, ioread: 0, readkeys: 6677,
filteredkeys: 0, rows: 0, covered: true, count_only: true)

```

통계 정보 갱신

테이블과 인덱스에 대한 통계 정보는 데이터베이스 시스템이 질의를 효과적으로 처리할 수 있게 한다. 통계 정보는 인덱스의 생성, 테이블의 생성 등 DDL 문과 INSERT, DELETE 등 DML 문에 대해서 자동으로 갱신되지 않는다. **UPDATE STATISTICS** 문만이 통계정보를 갱신하는 유일한 방법이다. 필요한 경우 사용자가 직접 **UPDATE STATISTICS** 문을 수행하여 통계 정보를 갱신해야 한다([통계 정보 확인 참고](#))

UPDATE STATISTICS 문은 주기적으로 수행할 것을 권장한다. 또한 새로운 인덱스가 추가되거나, 대량의 INSERT, 혹은 DELETE 문이 수행되어 실제 정보와 통계 정보 사이에 차이가 커질 때 수행할 것을 권장한다.

통계정보 갱신으로 관련된 질의의 실행계획 캐시는 삭제되지 않으며, 질의 실행 시 아래 두 가지 기준을 만족할 때 실행계획이 재생성된다.

1. 실행계획 캐시 생성 혹은 재생성 확인 후 6분이 지난 시점
2. 페이지가 10배 이상 커지거나 작아진 테이블의 통계정보가 갱신됨

필요한 경우 사용자가 **PLANDUMP** 유틸리티를 사용해서 직접 삭제할 수 있다. **PLANDUMP** 대한 자세한 내용은 [plandump](#)를 참고한다.

```
UPDATE STATISTICS ON [schema_name.]class-name [{,
[schema_name.]class-name}] [WITH FULLSCAN];

UPDATE STATISTICS ON ALL CLASSES [WITH FULLSCAN];

UPDATE STATISTICS ON CATALOG CLASSES [WITH FULLSCAN];
```

- **WITH FULLSCAN:** 지정된 테이블의 전체 데이터를 가지고 통계 정보를 업데이트한다. 생략 시 샘플링한 데이터를 가지고 통계 정보를 업데이트한다. 데이터 샘플은 테이블 전체 페이지 수와 상관없이 5000페이지이다.
- **ALL CLASSES:** 모든 테이블의 통계 정보를 업데이트한다.
- **CATALOG CLASSES:** 카탈로그 테이블에 대한 통계 정보를 업데이트한다.

```
CREATE TABLE foo (a INT, b INT);
CREATE INDEX idx1 ON foo (a);
CREATE INDEX idx2 ON foo (b);

UPDATE STATISTICS ON foo;
UPDATE STATISTICS ON foo WITH FULLSCAN;

UPDATE STATISTICS ON ALL CLASSES;
UPDATE STATISTICS ON ALL CLASSES WITH FULLSCAN;

UPDATE STATISTICS ON CATALOG CLASSES;
UPDATE STATISTICS ON CATALOG CLASSES WITH FULLSCAN;
```

통계 정보 갱신 시작과 종료 시 서버 에러 로그에 NOTIFICATION 메시지를 출력하며, 이를 통해 통계 정보 갱신에 걸리는 시간을 확인할 수 있다.

```
Time: 05/07/13 15:06:25.052 - NOTIFICATION *** file ../../src/
storage/statistics_sr.c, line 123 CODE = -1114 Tran = 1, CLIENT =
testhost:csql(21060), EID = 4
Started to update statistics (class "code", oid : 0|522|3).

Time: 05/07/13 15:06:25.053 - NOTIFICATION *** file ../../src/
storage/statistics_sr.c, line 330 CODE = -1115 Tran = 1, CLIENT =
testhost:csql(21060), EID = 5
Finished to update statistics (class "code", oid : 0|522|3, error
code : 0).
```

i 참고

CUBRID 10.0 부터는 HA 환경의 마스터에서 수행한 **UPDATE STATISTICS** 문이 슬레이브/레플리카에 복제된다.

i 참고

CUBRID 11.3부터는 **UPDATE STATISTICS** 문을 실행할 때 동의어를 사용할 수 없다.

```
/* CURRENT_USER: PUBLIC */
CREATE TABLE t (c int);
CREATE SYNONYM s for t;

UPDATE STATISTICS ON t;
/* Execute OK. */

UPDATE STATISTICS ON s;
/* ERROR: before ' ; '
 * Class public.s does not exist. */
```

i 참고

CUBRID 11.4 부터는 **UPDATE STATISTICS** 문을 실행할 때 **SELECT** 권한이 필요하다.

통계 정보 확인

CSQL 인터프리터의 세션 명령어로 지정한 테이블의 통계 정보를 확인한다.

```
csql> ;info stats [schema_name.]table_name
```

- *schema_name*: 테이블의 스키마를 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 통계 정보를 확인할 테이블 이름

다음은 CSQL 인터프리터에서 *t1* 테이블의 통계 정보를 출력하는 예제이다.

```
CREATE TABLE t1 (code INT, name VARCHAR(20));
INSERT INTO t1 VALUES(1, 'Park'),(2, 'Park'),(3, 'Joo'),(4, 'Joo'),
(5, 'Song');
CREATE INDEX i_t1_code ON t1(code,name);
UPDATE STATISTICS ON t1;
;info stats t1
```

CLASS STATISTICS

```
Class name: t1 Timestamp: Thu Dec 19 15:15:10 2024
Total pages in class heap: 1
Total objects: 5
Number of attributes: 2
Attribute: code (integer)
    Number of Distinct Values: 5
    B+tree statistics:
        BTID: { 0 , 5760 }
        Cardinality: 5 (5,5) , Total pages: 3 , Leaf pages: 1 ,
Height: 2

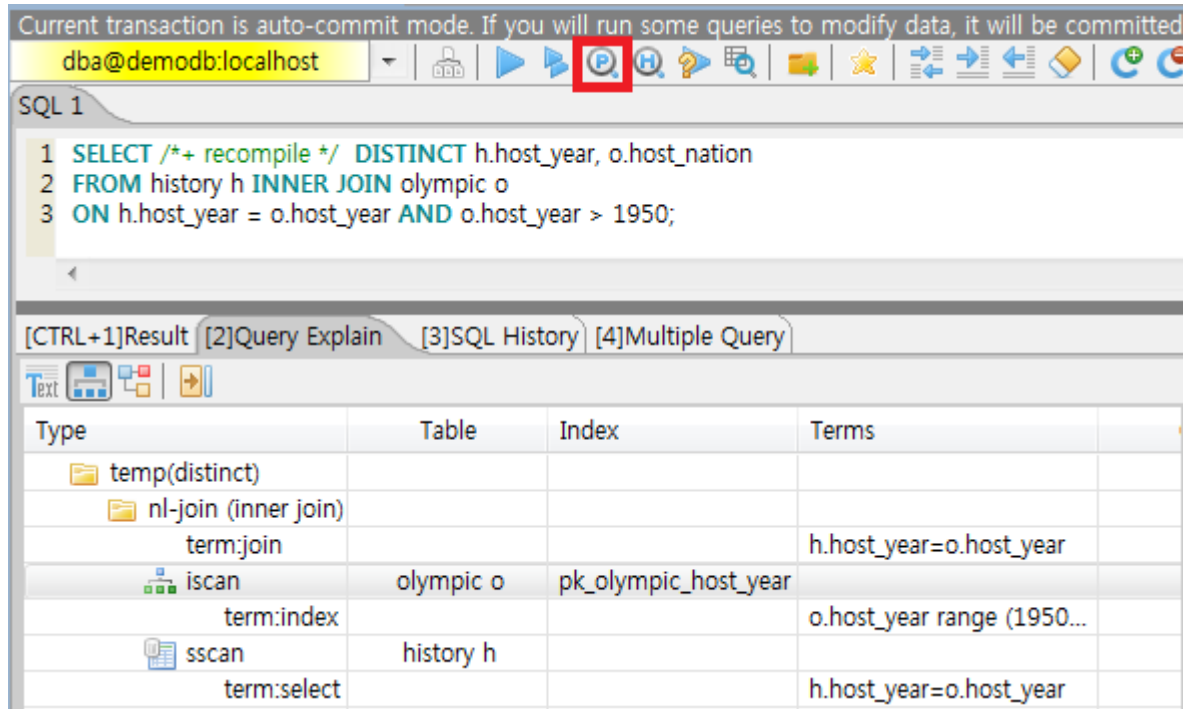
Attribute: name (character varying)
    Number of Distinct Values: 3
```

- *Timestamp*: 통계정보가 갱신된 시간이다.
- *Total pages*: 테이블의 페이지 개수이다.
- *Total objects*: 테이블의 총 행의 개수이다.
- *Number of Distinct Values*: 중복이 제거된 값의 개수이다. **code** 컬럼은 모두 다른 값이므로 5개이다. **LOB** 타입과 4000자리가 넘는 **VARCHAR** 컬럼은 해당값이 생성되지 않는다. 옵티마이저에서 선택도를 산정하는 데 사용되며 자세한 내용은 **최적화기 원리**를 참고한다.
- *B+tree statistics*: 인덱스 통계정보이다.
 - *B+tree Cardinality*: 인덱스 key 값의 누적된 중복이 제거된 값의 개수이다. 위 예제에서 **(5,5)**는 인덱스 컬럼 (**code,name**)과 매칭된다. 첫 번째 **5**는 **code** 컬럼의 중복이 제거된 값의 개수이고, 두 번째 **5**는 **code,name** 두 컬럼의 값의 중복이 제거된 개수이다. *Number of Distinct Values*와 달리 앞 컬럼과 함께 누적된 중복이 제거된 값의 개수이므로 5개이다.
 - *Total pages*: 인덱스 전체 페이지 개수이다.
 - *Leaf pages*: 인덱스 리프 노드의 페이지 개수이다.
 - *Height*: 리프 노드를 포함한 B+tree 인덱스의 높이이다.

질의 실행 계획 보기

CUBRID SQL 질의에 대한 실행 계획(query plan)을 보기 위해서는 다음의 방법을 사용할 수 있다.

- CUBRID Manager에서 “플랜 보기” 버튼을 누른다.



- CSQL 인터프리터에서 ;plan simple 또는 ;plan detail 명령을 실행하거나 **SET OPTIMIZATION** 구문을 이용해서 최적화 수준(optimization level) 값을 변경시킨다. 현재의 최적화 수준 값은 **GET OPTIMIZATION** 구문으로 얻을 수 있다. CSQL 인터프리터에 대한 자세한 내용은 [세션 명령어](#)를 참고한다.

SET OPTIMIZATION 또는 **GET OPTIMIZATION LEVEL** 구문은 다음과 같다.

```
SET OPTIMIZATION LEVEL opt-level [;]
GET OPTIMIZATION LEVEL [ { TO | INTO } variable ] [;]
```

- *opt-level* : 최적화 수준을 지정하는 값으로 다음과 같은 의미를 갖으며, 이 외에 값을 설정할 경우 에러를 발생한다.
 - 1 : 질의 최적화를 수행한다. CUBRID에서 사용되는 기본 설정 값으로 대부분의 경우 변경할 필요가 없다.
 - 2 : 질의 최적화를 수행하여 실행 계획을 생성하나 질의 자체는 수행되지 않는다. 일반적으로 사용되지 않고 다음 질의 실행 계획 보기를 위한 설정값과 같이 설정되어 사용된다.
 - 257 : 질의 최적화를 수행하여 생성된 질의 실행 계획(플랜)을 출력한다. 256+1의 값으로 해석하여 값을 1로 설정하고 질의 실행 계획 출력을 지정한 것과 같다.

- 258 : 질의 최적화를 수행하여 생성된 질의 실행 계획을 출력하나 질의를 수행하지는 않는다. 256+2의 값으로 해석하여 2로 설정하고 질의 실행 계획 출력을 지정한 것과 같다. 질의 실행 계획을 살펴보고자 하나 실행 결과에는 관심이 없을 경우 유용한 설정이다.
- 513 : 질의 최적화를 수행하고 상세 질의 실행 계획을 출력한다. 512+1의 의미이다.
- 514 : 질의 최적화를 수행하고 상세 질의 실행 계획을 출력하나 질의는 실행하지는 않는다. 512+2의 의미이다.

참고

2, 258, 514와 같이 질의를 실행하지 않게 최적화 수준을 설정하는 경우 SELECT 문 뿐만 아니라 INSERT, UPDATE, DELETE, REPLACE, TRIGGER, SERIAL 문 등 모든 질의문이 실행되지 않는다.

CUBRID 질의 최적화기는 사용자에게 의해 설정된 최적화 수준 값을 참조하여 최적화 여부와 질의 실행 계획의 출력 여부를 결정한다.

다음은 CSQL에서 “;plan simple” 명령 입력 또는 “SET OPTIMIZATION LEVEL 257;”을 입력 후 질의를 수행한 결과이다.

```
SET OPTIMIZATION LEVEL 257;
-- csql> ;plan simple
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year AND o.host_year > 1950;
```

Query plan:

```
Sort(distinct)
  Nested-loop join(h.host_year=o.host_year)
    Index scan(olympic o, pk_olympic_host_year, (o.host_year> ? :
0 ))
    Sequential scan(history h)
```

- Sort(distinct): DISTINCT를 수행한다.
- Nested-loop join: 조인 방식이 Nested-loop이다.
- Index scan: olympic 테이블에 대해 pk_olympic_host_year를 사용하여 index scan. 이때 인덱스를 사용한 조건은 “o.host_year> ?”이다.

CSQL에서 “;plan detail” 명령 입력 또는 “SET OPTIMIZATION LEVEL 513;”을 입력 후 질의를 수행하면 상세 내용을 출력한다.

```
SET OPTIMIZATION LEVEL 513;
-- csql> ;plan detail
```

```
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year AND o.host_year > 1950;
```

```
Join graph segments (f indicates final):
seg[0]: [0]
seg[1]: host_year[0] (f)
seg[2]: [1]
seg[3]: host_nation[1] (f)
seg[4]: host_year[1]
Join graph nodes:
node[0]: history h(147/1)
node[1]: olympic o(25/1) (sargs 1)
Join graph equivalence classes:
eqclass[0]: host_year[0] host_year[1]
Join graph edges:
term[0]: h.host_year=o.host_year (sel 0.04) (join term) (mergeable)
(inner-join) (indexable host_year[1]) (loc 0)
Join graph terms:
term[1]: o.host_year range (1950 gt_inf max) (sel 0.1) (rank 2)
(sarg term) (not-join eligible) (indexable host_year[1]) (loc 0)
```

Query plan:

```
temp(distinct)
  subplan: nl-join (inner join)
    edge: term[0]
    outer: iscan
      class: o node[1]
      index: pk_olympic_host_year term[1]
      cost: 1 card 2
    inner: sscan
      class: h node[0]
      sargs: term[0]
      cost: 1 card 147
    cost: 3 card 15
  cost: 9 card 15
```

Query stmt:

```
select distinct h.host_year, o.host_nation from history h, olympic o
where h.host_year=o.host_year and (o.host_year> ?:0 )
```

위의 출력 결과에서 질의 계획과 관련하여 봐야 할 정보는 “Query plan:”이며, 가장 안쪽의 윗줄부터 순서대로 실행된다. 즉, outer: iscan -> inner:scan이 nl-join에서 반복 수행되고, 마지막으로

temp(distinct)가 수행된다. “Join graph segments”는 “Query plan:”에서 필요한 정보를 좀더 확인하는 용도로 사용한다. 예를 들어 “Query plan:”에서 “term[0]”는 “Join graph segments”에서 “term[0]: h.host_year=o.host_year (sel 0.04) (join term) (mergeable) (inner-join) (indexable host_year[1]) (loc 0)”로 표현됨을 확인할 수 있다.

위의 “Query plan:” 각 항목에 대한 설명은 다음과 같다.

- temp(distinct): (distinct)는 DISTINCT를 실행함을 의미한다. temp는 실행 결과를 임시 공간에 저장했음을 의미한다.
 - nl-join: “nl-join”은 조인 방식이 중첩 루프 조인(Nested loop join)임을 의미한다.
 - (inner join): 조인 종류가 “inner join”임을 의미한다.
 - outer: iscan: outer 테이블에서는 iscan(index scan)을 수행한다.
 - class: o node[1]: o라는 테이블을 사용하며 상세 정보는 Join graph segments의 node[1]을 확인한다.
 - index: pk_olympic_host_year term[1]: pk_olympic_host_year 인덱스를 사용하며 상세 정보는 Join graph segments의 term[1]을 확인한다.
 - cost: 해당 구문을 수행하는데 드는 비용이다.
 - card: 카디널리티(cardinality)를 의미한다. 이 값은 근사치임에 유의한다.
 - inner: sscan: inner 테이블에 sscan(sequential scan)을 수행한다.
 - class: h node[0]: h라는 테이블을 사용하며 상세 정보는 Join graph segments의 node[0]을 확인한다.
 - sargs: term[0]: sargs는 데이터 필터(인덱스를 사용하지 않는 WHERE 조건)를 나타내며, term[0]는 데이터 필터로 사용된 조건을 의미한다.
 - cost: 해당 구문을 수행하는데 드는 비용이다.
 - card: 카디널리티(cardinality)를 의미한다. 이 값은 근사치임에 유의한다.
 - cost: 전체 구문을 수행하는데 드는 비용이다. 앞서 수행된 모든 비용을 포함한다.
 - card: 카디널리티(cardinality)를 의미한다. 이 값은 근사치임에 유의한다.

질의 계획 관련 용어

다음은 질의 계획으로 출력되는 각 용어에 대한 의미를 정리한 것이다.

- 조인 방식: 질의 계획에서 출력되는 조인 방식은 위에서 “nl-join” 부분으로 다음과 같다.
 - nl-join: 중첩 루프 조인, Nested loop join

- m-join: 정렬 병합 조인, Sort merge join
- idx_join: 중첩 루프 조인인데 outer 테이블의 행(row)을 읽으면서 inner 테이블에서 인덱스를 사용하는 조인
- hash-join: 해시 조인, Hash join
- 조인 종류: 위에서 (inner join) 부분으로, 질의 계획에서 출력되는 조인 종류는 다음과 같다.
 - inner join
 - left outer join
 - right outer join: 질의 계획에서는 질의문의 “outer” 방향과 다른 방향이 출력될 수도 있다. 예를 들어, 질의문에서는 “right outer”로 지정했는데 질의 계획에는 “left outer”로 출력될 수도 있다.
 - cross join
- 조인 테이블의 종류: 위에서 outer/inner 부분으로, 중첩 루프 조인에서 루프의 어느 쪽에 위치하는가를 기준으로 outer 테이블과 inner 테이블로 나뉜다.
 - outer 테이블: 조인할 때 가장 처음에 읽을 기준 테이블
 - inner 테이블: 조인할 때 나중에 읽을 대상 테이블
- 스캔 방식: 위에서 iscan/sscan 부분으로, 해당 질의가 인덱스를 사용하는지 여부를 판단할 수 있다.
 - sscan: 순차 스캔(sequential scan). 풀 테이블 스캔(full table scan)이라고도 하며 인덱스를 사용하지 않고 테이블 전체를 스캔한다.
 - iscan: 인덱스 스캔(index scan). 인덱스를 사용하여 스캔할 데이터의 범위를 한정한다.
- cost: CPU, IO 등 주로 리소스의 사용과 관련하여 비용을 내부적으로 산정한다.
- card: 카디널리티(cardinality)를 의미하며, 선택될 것으로 예측되는 행의 개수이다.

다음은 USE_MERGE 힌트를 명시하여 m-join(정렬 병합 조인, sort merge join)이 적용되는 경우의 예이다. 일반적으로 정렬 병합 조인은 outer 테이블과 inner 테이블을 정렬하여 병합하는 것이 인덱스를 사용하여 중첩 루프 조인(nested loop join)을 수행하는 것보다 유리하다고 판단될 때만 사용해야 하며, 조인되는 두 테이블 모두 행의 개수가 매우 많은 경우 유리할 수 있다. 대부분의 경우 정렬 병합 조인을 수행하지 않는 것이 바람직하다.

참고

9.3 버전부터 질의문에 `USE_MERGE` 힌트를 명시하거나 `cubrid.conf`의 `optimizer_enable_merge_join` 값을 `yes`로 설정해야 정렬 병합 조인의 적용이 고려된다.

```
SET OPTIMIZATION LEVEL 513;
-- csql> ;plan detail

SELECT /** RECOMPILE USE_MERGE*/ DISTINCT h.host_year, o.host_nation
FROM history h LEFT OUTER JOIN olympic o ON h.host_year =
o.host_year AND o.host_year > 1950;
```

Query plan:

```
temp(distinct)
  subplan: temp
    order: host_year[0]
    subplan: m-join (left outer join)
      edge: term[0]
      outer: temp
        order: host_year[0]
        subplan: sscan
          class: h
node[0]
cost: 1 card
147
cost: 10 card 147
inner: temp
  order: host_year[1]
  subplan: iscan
    class: o
node[1]
index:
pk_olympic_host_year term[1]
cost: 1 card 2
cost: 7 card 2
cost: 18 card 147
cost: 24 card 147
cost: 30 card 147
```

다음은 `idx-join`(인덱스 조인, index join)을 수행하는 경우의 예이다. inner 테이블의 조인 조건 컬럼에 인덱스가 있는 경우 inner 테이블의 인덱스를 사용하여 조인을 수행하는 것이 유리하다고 판단되면 `USE_IDX` 힌트를 명시하여 `idx-join`의 실행을 보장할 수 있다.

```
SET OPTIMIZATION LEVEL 513;
-- csql> ;plan detail
```

```
CREATE INDEX i_history_host_year ON history(host_year);

SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o ON h.host_year = o.host_year;
```

Query plan:

```
temp(distinct)
  subplan: idx-join (inner join)
    outer: sscan
      class: o node[1]
      cost: 1 card 25
    inner: iscan
      class: h node[0]
      index: i_history_host_year term[0]
(covers)
      cost: 1 card 147
    cost: 2 card 147
  cost: 9 card 147
```

위의 질의 계획에서 “inner: iscan”의 “index: i_history_host_year term[0]”에 “(covers)”가 출력되는데, 이는 **커버링 인덱스** 기능이 적용된다는 의미이다. 즉, inner 테이블에서 인덱스 내에 필요한 데이터가 있어서 데이터 저장소를 추가로 검색할 필요가 없게 된다.

조인 테이블 중 왼쪽 테이블이 오른쪽 테이블보다 행의 개수가 훨씬 작음을 확신할 때 **ORDERED** 힌트를 명시하여 왼쪽 테이블을 outer 테이블로, 오른쪽 테이블을 inner 테이블로 지정할 수 있다.

```
SELECT /*+ RECOMPILE ORDERED */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o ON h.host_year = o.host_year;
```

LEADING 힌트를 명시하여 테이블 순서를 지정할 수 있다. **ORDERED** 힌트와 다르게 작성된 순서와 상관없이 지정이 가능하다.

```
SELECT /*+ RECOMPILE LEADING(o,h) */ DISTINCT h.host_year,
o.host_nation
FROM history h INNER JOIN olympic o ON h.host_year = o.host_year;
```

최적화기 원리

CUBRID의 최적화기는 질의 실행 계획을 생성할 때 비용 기반 최적화를 진행하며, 통계정보를 통해 선택도와 예측되는 행과 페이지 개수를 산정한다. 이를 기반으로 비용을 계산하여 가장 적은 비용을 가진 실행계획을 선택한다.

선택도

선택도는 특정 조회 조건을 필터하였을 때 전체 대비 선택될 데이터의 비율이다. 최적화기는 **WHERE** 절의 각각의 조회 조건에 대해서 선택도를 산정한다. **CUBRID**는 전체 데이터가 고르게 분포되어 있다고 가정하고, 통계정보의 **Number of Distinct Value**을 사용하여 선택도를 산정한다.

```
SELECTIVITY = 1 / Number of Distinct Value
```

```
CREATE TABLE t1 (code INT, name VARCHAR(20));
INSERT INTO t1 VALUES(1,'Park'),(2,'Park'),(3,'Park'),(4,'joo'),
(5,'joo'),(5,'joo');
CREATE INDEX i_t1_code ON t1(code,name);
UPDATE STATISTICS ON t1;

;info stats t1
```

```
CLASS STATISTICS
*****
Class name: t1 Timestamp: Fri Dec 20 13:07:58 2024
Total pages in class heap: 1
Total objects: 6
Number of attributes: 2
Attribute: code (integer)
    Number of Distinct Values: 5
    B+tree statistics:
        BTID: { 0 , 5760 }
        Cardinality: 5 (5,5) , Total pages: 3 , Leaf pages: 1 ,
Height: 2

Attribute: name (character varying)
    Number of Distinct Values: 2
```

```
;plan detail
SELECT /*+ recompile */ * FROM t1 WHERE code = 3 AND name IN
('Song', 'Ham');
```

```
Join graph terms:
term[0]: [dba.t1].code=3 (sel 0.2)
term[1]: [dba.t1].[name] range ('Ham' = or 'Song' = ) (sel 0.75)
```

term[0]: [dba.t1].code=3는 통계정보에서 **code**의 **Number of Distinct Values**가 5이므로 선택도는 1/5, 즉 0.2이다. **term[1]: [dba.t1].[name] range ('Ham' = or 'Song' =)**는 **name**을 두 가지 값으로 **OR** 연산하고 있다. **name**의 **Number of Distinct Values**가 2이므로 선택도는 1/2이고, 두 값을 더하고 교집합을 제외하면 **0.5 + 0.5 - (0.5 x 0.5)** 계산으로 0.75가 나온다.

예측되는 행수

질의를 실행했을 때 조회될 것으로 예측되는 행수는 전체 데이터 행수에 선택도 곱해서 산정한다.

```
Number of expected rows = total rows of a table * selectivity
```

```
;plan detail
SELECT /** recompile */ * FROM t1 WHERE name = 'Park';
```

```
Join graph nodes:
node[0]: dba.t1 dba.t1(6/1)
Join graph terms:
term[0]: [dba.t1].[name]='Park' (sel 0.5)
```

Query plan:

```
sscan
  class: t1 node[0]
  sargs: term[0]
  cost: 1 card 3
```

dba.t1(6/1) 은 t1 테이블의 전체 행수가 6개이고 페이지수가 1개인 것을 나타낸다. **[dba.t1].[name]='Park'**의 선택도가 0.5이므로 예측되는 행수는 **6 x 0.5** 계산으로 3개이다. 실행계획의 **cost: 1 card 3**는 비용은 1이고 예측되는 행수가 3인 것을 나타낸다.

순차 스캔 비용

최적화기는 비용을 산정할 때 아래 두 가지 사항을 고려한다.

1. 페이지를 읽는 비용
2. 반복되는 루틴의 CPU 비용

아래는 순차 스캔의 비용이 어떻게 산정되는지 보여준다.

```
페이지 비용 = 테이블의 페이지 개수
CPU 비용 = 테이블의 전체 행의 개수 X CPU 가중치
```

```
drop table if exists t1;
create table t1 (col1 int, col2 int, col3 int, col4 int);
insert into t1 select mod(rownum,2), mod(rownum,4), rownum, rownum
from dual connect by level <= 4000;
update statistics on t1;
```

```
;plan detail
select /** recompile */ count(*) from t1 where col2 = 2;
```

```
Join graph nodes:
node[0]: dba.t1 dba.t1(4000/9)
Join graph terms:
term[0]: [dba.t1].col2=2 (sel 0.25)
```

Query plan:

```
sscan
  class: t1 node[0]
  sargs: term[0]
  cost: 19 card 1000
```

node[0]: dba.t1 dba.t1(4000/9) 에서 t1 테이블의 전체 행의 개수는 4000개이고 페이지 개수는 9개인 것을 알 수 있다. 그러므로 페이지 비용은 t1 테이블의 전체 페이지 개수인 9이다. CPU 비용은 테이블의 전체 행수에 CPU 가중치를 곱한 것으로, **4000 x 0.0025** 계산하면 10이다. 두 값을 더하면 19이며, **cost: 19 card 1000** 에서 비용은 19이고 조회가 예상되는 행수는 1000개인 것을 알 수 있다.

인덱스 스캔 비용

인덱스 스캔은 논리프 노드, 리프 노드, 그리고 힙 영역을 순차적으로 탐색하는 방식으로 이루어진다. 스캔은 먼저 루트 노드를 포함한 **논리프 노드(logical internal nodes)**를 탐색하여, **리프 노드(leaf node)**의 시작 지점을 찾는 것으로 시작된다.

그다음, 리프 노드를 순차적으로 탐색하며 조건을 만족하는 키의 OID(Object Identifier) 정보를 조회한다. 필요한 경우, 이 OID 정보를 기반으로 **힙 영역(heap area)**을 스캔하여 실제 데이터를 가져온다.

아래 예제는 이러한 인덱스 스캔 과정이 어떻게 수행되는지를 보여주고 있다.

```
drop table if exists t2;
create table t2 (col1 int, col2 int, col3 int, col4 int);
insert into t2 select mod(rownum,20), mod(rownum,80), rownum, rownum
from dual connect by level <= 4000;
create index idx on t2(col1,col2,col3);
update statistics on t2;

;plan detail
select /** recompile */ count(*) from t2 where col1 = 1 and col3 = 1
and col4 = 1;
```

```
Join graph terms:
term[0]: [dba.t2].col4=1 (sel 0.00025)
term[1]: [dba.t2].col3=1 (sel 0.0125)
term[2]: [dba.t2].col1=1 (sel 0.05)
```

Query plan:

```

iscan
  class: t2 node[0]
  index: idx term[2]
  filtr: term[1]
  sargs: term[0]
  cost: 4 card 1

```

Join graph terms의 정보를 통해 실행계획에 사용된 조회 조건의 확인이 가능하다. **index: idx term[2]**는 인덱스 수직 스캔으로 논리프 노드의 탐색에 사용된 조회 조건이다. 인덱스 수평 스캔 시 리프 노드에 대해서 키 필터하는데 **filtr: term[1]** 조회 조건이 사용되었다. 힙에 접근하여 조회 시에는 **sargs: term[0]** 조회 조건을 사용하여 필터한다.

```
;info stats t2
```

```

Attribute: col1 (integer)
  Number of Distinct Values: 20
  B+tree statistics:
    BTID: { 0 , 5760 }
    Cardinality: 4000 (20,4000,4000) , Total pages: 11 , Leaf
    pages: 9 , Height: 2

```

인덱스 스캔의 비용은 아래와 같이 계산된다.

1. 페이지를 읽는 비용 = 접근 예측되는 논리프 노드 페이지 + 리프 노드 페이지 + 힙 페이지
2. 반복되는 루틴의 CPU 비용 = 접근 예측되는 리프 노드 키 개수

논리프 노드의 페이지 개수는 통계정보의 인덱스 **Height**가 2이고 리프노드를 제외한 읽어야 하는 페이지는 **2 - 1** 이므로 1개이다. 리프 노드 페이지 개수는 **Leaf pages** 에 논리프 노드의 탐색에 사용된 조회조건 **index: idx term[2]** 의 선택도를 곱하여 구한다. **MAX(9 X 0.05, 1)** 이므로 읽어야 할 리프 노드 페이지 개수는 1개이다. 읽어야 하는 힙 페이지 개수는 테이블의 전체 page수에 **index**와 **filtr** 조건의 선택도를 곱하여 구한다. 여기서는 **MAX(9 X 0.05 X 0.0125, 1)** 으로 계산되어 1개의 페이지를 읽어야 하는 것으로 계산된다. 마지막으로 CPU 비용은 테이블의 전체 행수에서 **index: idx term[2]** 의 선택도와 CPU 가중치를 곱하여 구한다. **MAX(4000 X 0.05 X 0.0025, 1)** 으로 계산하면 1이 나오며, 지금까지의 모든 결과를 합하면 비용은 4라는 것을 알 수 있다. 실행계획에서 **cost: 4 card 1** 를 보면 비용은 4이고 예상되는 행수는 1개라는 것을 알 수 있다.

조인 질의 비용

조인 질의의 비용은 선행 테이블의 비용과 후행 테이블의 비용을 각각 산출하고, 조인 방법에 따라 비용을 산정한다. 아래 예제는 중첩 루프 조인에서 비용이 어떻게 계산되는지 보여준다.

```
create index idx1 on t2(col4);

--;plan detail
select /*+ recompile */ count(*)
from t1 a, t2 b
where a.col4 = b.col4
and b.col1 = 1
and b.col3 = 1
and a.col2 = 2;
```

Query plan:

```
idx-join (inner join)
  outer: sscan
        class: a node[0]
        sargs: term[3]
        cost: 19 card 1000
  inner: iscan
        class: b node[1]
        index: idx1 term[0]
        sargs: term[1] AND term[2]
        cost: 2 card 3
cost: 572 card 1
```

선행되는 테이블은 *a*이며 후행 테이블은 *b*이다. 중첩 루프 조인 특성상 선행 테이블의 행수만큼 후행 테이블의 실행이 반복되므로 *b* 테이블의 가변 비용과 *a* 테이블의 예상되는 행수를 곱하여 후행 테이블의 비용을 산정 할 수 있다. **CUBRID**는 내부적으로 고정 비용과 가변 비용을 따로 관리하고 있으며 실행계획에서는 이 정보를 확인 할 수 없다. 대략 후행 테이블의 가변비용은 0.553이며 조인 시 발생하는 후행 테이블의 비용은 $0.553 * 1000$ 계산하면 553이고 선행 테이블의 비용 19를 더하면 최종 비용은 572이다.

질의 프로파일링

SQL에 대한 성능 분석을 위해서는 질의 프로파일링(profiling) 기능을 사용할 수 있다. 질의 프로파일링을 위해서는 **SET TRACE ON** 구문으로 SQL 트레이스를 설정해야 하며, 프로파일링 결과를 출력하려면 **SHOW TRACE** 구문을 수행해야 한다.

또한 **SHOW TRACE** 결과 출력 시 질의 실행 계획을 항상 포함하려면 **/*+ RECOMPILE */** 힌트를 추가해야 한다.

SET TRACE ON 구문의 형식은 다음과 같다.

```
SET TRACE {ON | OFF} [OUTPUT {TEXT | JSON}]
```

- ON: SQL 트레이스를 on한다.
- OFF: SQL 트레이스를 off한다.
- OUTPUT TEXT: 일반 TEXT 형식으로 출력한다. OUTPUT 이하 절을 생략하면 TEXT 형식으로 출력한다.
- OUTPUT JSON: JSON 형식으로 출력한다.

아래와 같이 **SHOW TRACE** 구문을 실행하면 SQL을 트레이스한 결과를 문자열로 출력한다.

```
SHOW TRACE;
```

다음은 SQL 트레이스를 ON으로 설정하고 질의를 수행한 후, 해당 질의에 대해 트레이스 결과를 출력하는 예이다.

```
csql> SET TRACE ON;
csql> SELECT /*+ RECOMPILE */ o.host_year, o.host_nation,
o.host_city, SUM(p.gold)
      FROM OLYMPIC o, PARTICIPANT p
      WHERE o.host_year = p.host_year AND p.gold > 20
      GROUP BY o.host_nation;
csql> SHOW TRACE;
```

```
=== <Result of SELECT Command in Line 2> ===
```

```
  trace
=====
  ,
```

Query Plan:

```
  SORT (group by)
    NESTED LOOPS (inner join)
      TABLE SCAN (o)
      INDEX SCAN (p.fk_participant_host_year) (key range:
o.host_year=p.host_year)
```

```
  rewritten query: select o.host_year, o.host_nation, o.host_city,
sum(p.gold) from OLYMPIC o, PARTICIPANT p where
o.host_year=p.host_year and (p.gold> ? :0 ) group by o.host_nation
```

Trace Statistics:

```
  SELECT (time: 2, fetch: 975, fetch_time: 1, ioread: 2)
    SCAN (table: olympic), (heap time: 0, fetch: 26, ioread: 0,
readrows: 25, rows: 25)
```

```
SCAN (index: participant.fk_participant_host_year), (btree
time: 1, fetch: 941, ioread: 2, readkeys: 5, filteredkeys: 5, rows:
916) (lookup time: 0, rows: 14)
  GROUPBY (time: 0, sort: true, page: 0, ioread: 0, rows: 5)
,
```

위에서 “Trace Statistics:” 이하가 트레이스 결과를 출력한 것이며 트레이스 결과의 각 항목을 설명하면 다음과 같다.

• **SELECT** (time: 2, fetch: 975, fetch_time: 1, ioread: 2)

- time: 2 => 전체 질의 시간 2ms 소요.
- fetch: 975 => 페이지에 대해 975회 fetch(개수가 아닌 접근 회수임. 같은 페이지를 다시 fetch하더라도 회수가 증가함).
- fetch_time: 1 => fetch 시간 1ms 소요.
- ioread: 2회 디스크 접근.

: SELECT 질의에 대한 전체 통계이다. fetch 회수와 ioread 회수는 질의를 재실행하면 질의 결과의 일부를 버퍼에서 가져오게 되면서 줄어들 수 있다.

◦ **SCAN** (table: olympic), (heap time: 0, fetch: 26, ioread: 0, readrows: 25, rows: 25)

- heap time: 0 => 소요 시간은 1ms 미만. millisecond보다 작은 값은 버림하기 때문에 1ms 미만의 소요 시간은 0으로 표시된다.
- fetch: 26 => 페이지를 fetch한 회수는 26건.
- ioread: 0 => 디스크에 접근한 회수는 0.
- readrows: 25 => 스캔 시 읽은 행의 개수는 25.
- rows: 25 => 결과 행의 개수는 25.

: olympic 테이블에 대한 힙 스캔 통계이다.

- **SCAN** (index: participant.fk_participant_host_year), (btree time: 1, fetch: 941, ioread: 2, readkeys: 5, filteredkeys: 5, rows: 916) (lookup time: 0, rows: 14)
 - btree time: 1 => 소요 시간은 1ms.
 - fetch: 941 => 페이지를 fetch한 회수는 941.
 - ioread: 2 => 디스크에 접근한 회수는 2회.
 - readkeys: 5 => 읽은 키의 개수는 5.
 - filteredkeys: 5 => 키 필터가 적용된 키의 개수는 5.

- rows: 916 => 스캔한 행 개수는 916.
- lookup time: 0 => 인덱스 스캔 후 데이터에 접근하는데 소요된 시간은 1ms 미만.
- rows: 14 => 데이터 필터까지 적용한 이후의 행 개수로, 이 질의문에서는 데이터 필터인 "p.gold > 20"을 적용했을 때 행의 개수는 14.

: participant.fk_participant_host_year 인덱스에 대한 인덱스 스캔 통계이다.

◦ **GROUPBY** (time: 0, sort: true, page: 0, ioread: 0, rows: 5)

- time: 0 => group by 적용 시 소요된 시간은 1ms 미만.
- sort: true => 정렬이 적용되므로 true.
- page: 0 => 정렬에 사용된 임시 페이지 개수가 0.
- ioread: 0 => 디스크 접근에 소요된 시간은 1ms 미만.
- rows: 5 => group by에 대한 결과 행의 개수는 5개.

: group by에 대한 통계이다.

다음은 3개의 테이블을 조인한 예이다.

```
csql> SET TRACE ON;
csql> SELECT /*+ RECOMPILE ORDERED */ o.host_year, o.host_nation,
o.host_city, n.name, SUM(p.gold), SUM(p.silver), SUM(p.bronze)
      FROM OLYMPIC o,
           (select /*+ NO_MERGE */ * from PARTICIPANT p where
p.gold > 10) p,
           NATION n
      WHERE o.host_year = p.host_year AND p.nation_code = n.code
      GROUP BY o.host_nation;
csql> SHOW TRACE;

      trace
=====
,

Query Plan:
  TABLE SCAN (p)

rewritten query: (select p.host_year, p.nation_code, p.gold,
p.silver, p.bronze from PARTICIPANT p where (p.gold> ?:0 ))

SORT (group by)
  NESTED LOOPS (inner join)
    NESTED LOOPS (inner join)
      TABLE SCAN (o)
      TABLE SCAN (p)
    INDEX SCAN (n.pk_nation_code) (key range: p.nation_code=n.code)
```

```
rewritten query: select /** ORDERED */ o.host_year, o.host_nation,
o.host_city, n.[name], sum(p.gold), sum(p.silver), sum(p.bronze)
from OLYMPIC o, (select p.host_year, p.nation_code, p.gold,
p.silver, p.bronze from PARTICIPANT p where (p.gold>?:0 )) p
(host_year, nation_code, gold,
silver, bronze), NATION n where o.host_year=p.host_year and
p.nation_code=n.code group by o.host_nation
```

Trace Statistics:

```
SELECT (time: 6, fetch: 880, fetch_time: 2, ioread: 0)
  SCAN (table: olympic), (heap time: 0, fetch: 104, ioread: 0,
readrows: 25, rows: 25)
    SCAN (hash temp(m), buildtime : 0, time: 0, fetch: 0, ioread:
0, readrows: 76, rows: 38)
      SCAN (index: nation.pk_nation_code), (btree time: 2, fetch:
760, ioread: 0, readkeys: 38, filteredkeys: 0, rows: 38) (lookup
time: 0, rows: 38)
        GROUPBY (time: 0, hash: true, sort: true, page: 0, ioread: 0,
rows: 5)
          SUBQUERY (uncorrelated)
            SELECT (time: 2, fetch: 12, ioread: 0)
              SCAN (table: participant), (heap time: 2, fetch: 12, ioread:
0, readrows: 916, rows: 38)
            ,
```

다음은 트레이스 항목에 대한 설명이다.

SELECT

- time: 해당 질의에 대한 전체 수행 시간(ms)
- fetch: 해당 질의에 대해 페이지를 fetch한 회수
- fetch_time : 해당 질의에 대해 페이지 fetch 수행 시간(ms)
- ioread: 해당 질의에 대한 전체 I/O 읽기 회수. 데이터를 읽을 때 물리적으로 디스크에 접근한 회수

FUNC: 현재 질의와 서브 질의에서 호출한 저장 함수의 결과를 누적하여 출력한다. fetch와 ioread는 저장 함수 내 질의 수행 시에 발생한 횟수는 포함하지 않는다.

- time: 저장 함수 전체 수행 시간(ms)
- fetch: 저장 함수 호출 중 페이지를 fetch한 횟수.
- ioread: 저장 함수 호출 중 발생한 전체 I/O 읽기 횟수. 데이터를 읽을 때 물리적으로 디스크에 접근한 횟수
- calls: 저장 함수 호출 횟수

SCAN

- heap: 인덱스 없이 데이터를 스캔하는 작업
 - time, fetch, ioread: heap에서 해당 연산 수행 시 소요된 시간(ms), fetch 회수, I/O 읽기 회수
 - readrows: 해당 연산 수행 시 읽은 행의 개수
 - rows: 해당 연산에 대한 결과 행의 개수
- btree: 인덱스 스캔하는 작업
 - time, fetch, ioread: btree에서 해당 연산 수행 시 소요된 시간(ms), fetch 회수, I/O 읽기 회수
 - readkeys: btree에서 해당 연산 수행 시 읽은 키의 개수
 - filteredkeys: 읽은 키 중에 키 필터가 적용된 키의 개수
 - rows: 해당 연산에 대한 결과 행의 개수로, 키 필터가 적용된 결과 행의 개수
- temp: 템프 파일에서 데이터를 스캔하는 작업
 - hash temp(m): 해시 리스트 스캔 사용 여부. 데이터 양에 따라서, IN-MEMORY(m), HYBRID(h), FILE(f) 해시 자료구조를 사용한다.
 - buildtime: 해시 테이블 빌드 수행 시 소요된 시간(ms)
 - time: 해시 테이블 조사 수행 시 소요된 시간(ms)
 - fetch, ioread: temp file에서 해당 연산 수행 시 소요된 fetch 회수, I/O 읽기 회수
 - readrows: 해당 연산 수행 시 읽은 행의 개수
 - rows: 해당 연산에 대한 결과 행의 개수
- lookup: 인덱스 스캔 후 데이터에 접근하는 작업
 - time: 해당 연산 수행 시 소요된 시간(ms)
 - rows: 해당 연산에 대한 결과 행의 개수로, 데이터 필터가 적용된 결과 행의 개수
- noscan: aggregate 연산 시 스캔 없이 인덱스 헤더의 통계 정보를 이용하는 작업 (aggregate: count, min, max)
 - agl: aggregate lookup을 의미하며, aggregate 연산을 위해 사용된 인덱스 리스트

다음은 noscan, agl에 대한 예제이다.

```
SET TRACE ON;
CREATE TABLE agl_tbl (id INTEGER PRIMARY KEY, phone VARCHAR(20));
INSERT INTO agl_tbl VALUES (1, '123-456-789');
```

```
INSERT INTO agl_tbl VALUES (999, '999-999-999');

SELECT count(*), min(id), max(id) FROM agl_tbl;

SHOW TRACE;
```

```
Trace Statistics:
  SELECT (time: 0, fetch: 16, fetch_time: 0, ioread: 0)
    SCAN (table: agl_tbl), (noscan time: 0, fetch: 0, ioread: 0,
readrows: 0, rows: 0, agl: pk_agl_tbl_id)
```

GROUPBY

- time: 해당 연산 수행 시 소요된 시간(ms)
- sort: 정렬 여부
- page: 정렬에 사용된 임시 페이지 개수로, 내부 정렬 버퍼 외에 사용한 페이지 개수.
- rows: 해당 연산에 대한 결과 행의 개수
- hash: 집계 함수에서 튜플 정렬 시 해시 집계 방식 적용 여부(true/false). **NO_HASH_AGGREGATE** 힌트를 참고한다.

INDEX SCAN

- key range: 키의 범위
- covered: 커버링 인덱스 적용 여부(true/false)
- loose: loose index scan 적용 여부(true/false)

위의 예는 JSON 형식으로도 출력할 수 있다.

```
csql> SET TRACE ON OUTPUT JSON;
csql> SELECT n.name, a.name FROM athlete a, nation n WHERE
n.code=a.nation_code;
csql> SHOW TRACE;

trace
=====
'{'
  "Trace Statistics": {
    "SELECT": {
      "time": 29,
      "fetch": 5836,
      "ioread": 3,
      "SCAN": {
        "access": "temp",
```


- 여러 개의 SQL을 한 번에 처리하는 경우(batch query, array query) 질의는 프로파일링되지 않는다.

SQL 힌트

사용자는 질의문에 힌트를 주어 해당 질의 성능을 높일 수 있다. 질의 최적화기는 질의문에 대한 최적화 작업을 수행할 때 SQL 힌트를 참고하여 효율적인 실행 계획을 생성한다. CUBRID에서 지원하는 SQL 힌트는 테이블 조인 관련 힌트, 인덱스 관련 힌트가 있다.

```
{ SELECT | UPDATE | DELETE } /** <hint> [ { <hint> } ... ] */ ...;

MERGE /** <merge_statement_hint> [ { <merge_statement_hint> } ... ]
*/ INTO ...;

<hint> ::=
USE_NL [ (<spec_name_comma_list>) ] |
USE_IDX [ (<spec_name_comma_list>) ] |
USE_MERGE [ (<spec_name_comma_list>) ] |
USE_HASH [ (<spec_name_comma_list>) ] |
NO_USE_HASH [ (<spec_name_comma_list>) ] |
ORDERED |
LEADING |
USE_DESC_IDX |
USE_SBR |
INDEX_SS [ (<spec_name_comma_list>) ] |
INDEX_LS |
NO_DESC_IDX |
NO_COVERING_IDX |
NO_MULTI_RANGE_OPT |
NO_SORT_LIMIT |
NO_SUBQUERY_CACHE |
NO_PUSH_PRED |
NO_MERGE |
NO_ELIMINATE_JOIN |
NO_HASH_AGGREGATE |
NO_HASH_LIST_SCAN |
NO_LOGGING |
PARALLEL (<degree>) |
NO_PARALLEL_SCAN |
NO_PARALLEL_HASH_JOIN |
NO_PARALLEL_SUBQUERY |
RECOMPILE

<spec_name_comma_list> ::= <spec_name> [, <spec_name>, ... ]
    <spec_name> ::= [schema_name.]table_name |
[schema_name.]view_name
```

```
<merge_statement_hint> ::=
USE_UPDATE_INDEX (<update_index_list>) |
USE_DELETE_INDEX (<insert_index_list>) |
RECOMPILE |
QUERY_CACHE
```

SQL 힌트는 주석에 더하기 기호(+)를 함께 사용하여 지정한다. 힌트를 사용하는 방법은 **주석** 절에 소개된 바와 같이 세 가지 방식이 있다. 따라서 SQL 힌트도 다음과 같이 세 가지 방식으로 사용할 수 있다.

- /*+ hint */
- --+ hint
- //+ hint

힌트 주석은 반드시 키워드 **SELECT**, **UPDATE** or **DELETE** 등의 예약어 다음에 나타나야 하고, 더하기 기호(+)가 주석에서 첫 번째 문자로 시작되어야 한다. 여러 개의 힌트를 지정할 때는 공백이 구분자로 사용된다. 여러 개의 힌트를 지정할 때는 공백이 구분자로 사용된다.

SELECT, **UPDATE**, **DELETE** 문에는 다음 힌트가 지정될 수 있다.

- **USE_NL**: 테이블 조인과 관련한 힌트로서, 질의 최적화기 중첩 루프 조인 실행 계획을 만든다.
- **USE_MERGE**: 테이블 조인과 관련한 힌트로서, 질의 최적화기는 정렬 병합 조인 실행 계획을 만든다.
- **USE_HASH**: 테이블 조인과 관련한 힌트로서, 질의 최적화기는 해시 조인 실행 계획을 만든다. 자세한 내용은 **해시(Hash) 조인**을 참고한다.
- **NO_USE_HASH**: 테이블 조인과 관련한 힌트로서, 질의 최적화기가 해시 조인 실행 계획을 만들지 않는다. 자세한 내용은 **해시(Hash) 조인**을 참고한다.
- **ORDERED**: 테이블 조인과 관련한 힌트로서, 질의 최적화기는 **FROM** 절에 명시된 테이블의 순서대로 조인하는 실행 계획을 만든다. **FROM** 절에서 왼쪽 테이블은 조인의 외부 테이블이 되고, 오른쪽 테이블은 내부 테이블이 된다.
- **LEADING**: 테이블 조인과 관련한 힌트로서, 질의 최적화기는 LEADING 힌트에 명시된 테이블의 순서대로 조인하는 실행 계획을 만든다.
- **USE_IDX**: 인덱스 관련한 힌트로서, 질의 최적화기는 명시된 테이블에 대해 인덱스 조인 실행 계획을 만든다.
- **USE_DESC_IDX**: 내림차순 스캔을 위한 힌트이다. 자세한 내용은 **내림차순 인덱스 스캔**을 참고한다.
- **USE_SBR**: 구문 기반 복제(statement-based replication)를 위한 힌트로서, 기본키가 설정되지 않은 테이블에 대한 데이터 복제도 지원한다.

참고

슬레이브 노드에서 트랜잭션 로그가 반영되는 시점에 해당 구문을 다시 실행하기 때문에 노드 간 반영된 데이터의 불일치가 발생할 수 있다.

- **INDEX_SS**: index skip scan 실행 계획을 고려한다. 자세한 내용은 [Index Skip Scan](#)을 참고한다.
- **INDEX_LS**: loose index scan 실행 계획을 고려한다. 자세한 내용은 [Loose Index Scan](#)을 참고한다.
- **NO_DESC_IDX**: 내림차순 스캔을 사용하지 않도록 하는 힌트이다.
- **NO_COVERING_IDX**: 커버링 인덱스 기능을 사용하지 않도록 하는 힌트이다. 자세한 내용은 [커버링 인덱스](#)를 참고한다.
- **NO_MULTI_RANGE_OPT**: 다중 키 범위 최적화 기능을 사용하지 않도록 하는 힌트이다. 자세한 내용은 [다중 키 범위 최적화](#)를 참고한다.
- **NO_SORT_LIMIT**: SORT-LIMIT 최적화를 사용하지 않기 위한 힌트이다. 자세한 내용은 [SORT-LIMIT 최적화](#)를 참고한다.
- **NO_SUBQUERY_CACHE**: SUBQUERY_CACHE 최적화를 사용하지 않기 위한 힌트이다. 자세한 내용은 [서브 쿼리 캐시 \(correlated\)](#)를 참고한다.
- **NO_PUSH_PRED**: PREDICATE-PUSH 최적화를 사용하지 않기 위한 힌트이다.
- **NO_MERGE**: VIEW-MERGE 최적화를 사용하지 않기 위한 힌트이다.
- **NO_ELIMINATE_JOIN**: 조인 제거 최적화를 사용하지 않기 위한 힌트이다. 자세한 내용은 [조인 변환 최적화](#)를 참고한다.
- **NO_HASH_AGGREGATE**: 집계 함수에서 투플을 정렬할 때 해싱을 사용하지 않도록 하는 힌트이다. 그 대신, 외부 정렬(external sorting)이 집계 함수에서 사용된다. 인-메모리(in-memory) 해시 테이블을 사용하여, CUBRID는 정렬할 때 필요로 하는 데이터의 양을 줄이거나 심지어는 제거할 수 있다. 그러나, 어떤 경우에는 해시 집계 방식이 실패할 것이라는 것을 미리 알고 전체적으로 해시 집계 과정을 생략하기 위해 이 힌트를 사용할 수 있다. 해시 집계 방식의 메모리 크기를 설정하기 위해서는 [max_agg_hash_size](#)를 참고한다.

참고

DISTINCT한 값을 계산하는 함수들(예. AVG(DISTINCT x))과 GROUP_CONCAT, MEDIAN 함수들은 각 그룹의 튜플들에 대해 외부 정렬(external sorting) 과정을 요구하므로 해시 집계 방식이 동작하지 않을 것이다.

- **NO_HASH_LIST_SCAN**: 부질의 스캔 시 해시 리스트 스캔을 사용하지 않도록 하는 힌트이다. 그 대신, 템프 파일 스캔을 위해 리스트 스캔이 사용된다. 해시 테이블을 빌드 및 조사 함으로써, CUBRID는 조회할 때 필요로 하는 데이터의 양을 줄일 수 있다. 그러나, 어떤 경우에는 외부 데이터양이 매우 적다는 것을 미리 알고 전체적으로 해시 리스트 스캔 과정을 생략하기 위해 이 힌트를 사용할 수 있다. 해시 리스트 스캔의 메모리 크기를 설정하기 위해서는 `max_hash_list_scan_size`를 참고한다.

참고

해시 리스트 스캔은 오직 동등 연산자를 가지고 있는 조회조건에서 동작하며, OID 타입을 가지고 있는 조회 조건에서는 동작하지 않는다.

- **NO_LOGGING**: 테이블에 레코드 삽입, 갱신, 삭제 시 생성되는 로그에 리두(redo)가 포함되지 않도록 하는 힌트이다.

참고

현재 레코드 삽입, 갱신, 삭제 시 힙 파일에서 생성되는 로그에만 영향을 준다. 따라서 복구 후 테이블과 인덱스의 데이터가 불일치하는 문제, 커밋된 레코드가 복구되지 않는 문제 등이 발생할 수 있다. 반드시 주의하여 사용하여야 한다.

- **PARALLEL** (*degree*): 병렬 질의 실행(병렬 스캔(힙/리스트/인덱스), 병렬 부질의 실행, 병렬 해시 조인, 병렬 정렬)을 활성화하고 병렬 처리 정도를 지정하는 힌트이다. *degree* 는 0 이상의 정수 값이어야 하며, 병렬로 처리할 워커 스레드의 수를 의미한다. 0이나 1로 지정할 경우 병렬 처리 기능이 비활성화되며, 시스템의 CPU 코어 수를 초과하는 값은 코어 수로 낮추어 적용된다. 힌트를 지정해도 대상의 페이지 수가 활성화 조건에 미달하면 병렬 실행이 적용되지 않는다. 자세한 내용은 **병렬 실행**를 참고한다.

```
SELECT /*+ PARALLEL(4) */ * FROM large_table WHERE condition;
```

- **NO_PARALLEL_SCAN**: 해당 질의 블록의 모든 병렬 스캔(힙/리스트/인덱스)을 사용하지 않도록 하는 힌트이다. **PARALLEL** 힌트와 같이 사용하는 경우에는 **NO_PARALLEL_SCAN** 이 우선 적용된다. 자세한 내용은 **병렬 실행**를 참고한다.

```
SELECT /*+ NO_PARALLEL_SCAN */ * FROM large_table WHERE condition;
```

- **NO_PARALLEL_HASH_JOIN**: 해당 질의 블록에서 병렬 해시 조인을 사용하지 않도록 하는 힌트이다. 해시 조인 자체는 유지되고 병렬화만 비활성화된다. **PARALLEL** 힌트와 같이 사용하는 경우에는 **NO_PARALLEL_HASH_JOIN** 이 우선 적용된다. 자세한 내용은 **병렬 실행**를 참고한다.

```
SELECT /*+ NO_PARALLEL_HASH_JOIN */ o.order_id, t.category
FROM orders o JOIN large_table t ON o.order_id = t.id;
```

- **NO_PARALLEL_SUBQUERY**: 부질의(subquery)의 병렬 실행을 사용하지 않도록 하는 힌트이다. 자세한 내용은 **병렬 실행**를 참고한다.

```
SELECT /*+ NO_PARALLEL_SUBQUERY */ *
FROM table1
WHERE id IN (SELECT id FROM table2 WHERE status = 'active');
```

- **RECOMPILE**: 질의 실행 계획을 리컴파일한다. 캐시에 저장된 기존 질의 실행 계획을 삭제하고 새로운 질의 실행 계획을 수립하기 위해 이 힌트를 사용한다.
- **QUERY_CACHE**: 질의와 그 결과를 캐시한다. 이 힌트는 **SELECT** 질의에서만 사용할 수 있으며, 자세한 내용은 **쿼리 캐시**를 참고한다.

참고

<spec_name>이 **USE_NL**, **USE_IDX**, **USE_MERGE**와 함께 지정될 경우 해당 조인 방법은 <spec_name>에 대해서만 적용된다.

```
SELECT /*+ ORDERED USE_NL(b) USE_NL(c) USE_MERGE(d) */ *
FROM a INNER JOIN b ON a.col=b.col
INNER JOIN c ON b.col=c.col INNER JOIN d ON c.col=d.col;
```

위와 같은 질의를 수행한다면 테이블 a와 b가 조인될 때는 **USE_NL**이 적용되고 테이블 c가 조인될 때도 **USE_NL**이 적용되며, 테이블 d가 조인될 때는 **USE_MERGE**가 적용된다.

<spec_name>이 주어지지 않고 **USE_NL**과 **USE_MERGE**가 함께 지정된 경우 주어진 힌트는 무시된다. 일부 경우에 질의 최적화기는 주어진 힌트에 따라 질의 실행 계획을 만들지 못할 수 있다. 예를 들어 오른쪽 외부 조인에 대해 **USE_NL**을 지정한 경우 이 질의는 내부적으로 왼쪽 외부 조인 질의로 변환이 되어 조인 순서는 보장되지 않을 수 있다.

참고

ORDERED가 **LEADING**과 함께 지정될 경우 **LEADING** 힌트는 무시된다. **LEADING** 힌트가 여러개 지정될 경우 첫번째 **LEADING** 힌트만 적용된다.

```
SELECT /*+ ORDERED LEADING(b, d) */ *
FROM a INNER JOIN b ON a.col=b.col
INNER JOIN c ON b.col=c.col INNER JOIN d ON c.col=d.col;
```

위와 같은 질의를 수행한다면 **LEADING** 힌트는 무시되며, **ORDERED** 힌트에 따라서 **FROM** 절의 순서인, 테이블 a, b, c, d의 순서로 조인된다.

```
SELECT /*+ LEADING(b, d) LEADING(c, d) */ *
FROM a INNER JOIN b ON a.col=b.col
INNER JOIN c ON b.col=c.col INNER JOIN d ON c.col=d.col;
```

위와 같은 질의를 수행한다면 두번째 **LEADING** 힌트는 무시되며, 테이블 b와 d가 첫번째로 조인되는 조인순서가 생성된다.

MERGE 문에는 다음과 같은 힌트를 사용할 수 있다.

- **USE_INSERT_IDX** (<insert_index_list>): MERGE 문의 INSERT 절에서 사용되는 인덱스 힌트. *insert_index_list*에 INSERT 절을 수행할 때 사용할 인덱스 이름을 나열한다. MERGE 문의 <join_condition>에 해당 힌트가 적용된다.
- **USE_UPDATE_IDX** (<update_index_list>): MERGE 문의 UPDATE 절에서 사용되는 인덱스 힌트. *update_index_list*에 UPDATE 절을 수행할 때 사용할 인덱스 이름을 나열한다. MERGE 문의 <join_condition>과 <update_condition>에 해당 힌트가 적용된다.
- **RECOMPILE**: 위의 **RECOMPILE**을 참고한다.

조인 시 사용하는 힌트의 경우 힌트 안에 조인할 테이블이나 뷰 이름을 명시할 수 있는데, 이때 테이블 이름/뷰 이름은 “”로 구분한다.

```
SELECT /*+ USE_NL(a, b) */ *
FROM a INNER JOIN b ON a.col=b.col;
```

다음은 ‘심권호’ 선수가 메달을 획득한 연도와 메달 종류를 구하는 예제이다. 다음과 같은 질의로 표현이 되는데, 질의최적화기는 *athlete* 테이블을 외부 테이블로 하고, *game* 테이블을 내부 테이블로 하는 중첩 루프 조인 실행 계획을 만든다.

```
-- csql> ;plan_detail
```

```
SELECT /*+ USE_NL ORDERED */ a.name, b.host_year, b.medal
FROM athlete a, game b
WHERE a.name = 'Sim Kwon Ho' AND a.code = b.athlete_code;
```

Query plan:

```
idx-join (inner join)
  outer: sscan
        class: a node[0]
        sargs: term[1]
        cost: 44 card 7
  inner: iscan
        class: b node[1]
        index: fk_game_athlete_code term[0]
        cost: 3 card 8653
cost: 73 card 9
```

다음은 **USE_NL** 힌트 사용 시 사용하는 테이블을 명시하는 예이다.

```
-- csql> ;plan_detail
```

```
SELECT /*+ USE_NL(a,b) */ a.name, b.host_year, b.medal
FROM athlete a, game b
WHERE a.name = 'Sim Kwon Ho' AND a.code = b.athlete_code;
```

인덱스 힌트

인덱스 힌트 구문은 질의에서 인덱스를 지정할 수 있도록 해서 질의 처리기가 적절한 인덱스를 선택할 수 있게 한다. 이와 같은 인덱스 힌트 구문은 **USING INDEX** 절을 사용하는 방식과 FROM 절에 { **USE | FORCE | IGNORE** } **INDEX** 구문을 사용하는 방식이 있다.

USING INDEX

USING INDEX 절은 **SELECT**, **DELETE**, **UPDATE** 문의 **WHERE** 절 다음에 지정되어야 한다. **USING INDEX** 절에 강제로 순차 스캔 또는 인덱스 스캔이 사용되게 하거나, 성능에 유리한 인덱스가 포함되도록 한다.

USING INDEX 절에 인덱스 이름의 리스트가 지정되면 질의 최적화기는 지정된 인덱스만을 대상으로 질의 실행 비용을 계산하고, 지정된 인덱스의 인덱스 스캔 비용과 순차 스캔 비용을 비교하여 최적의 실행 계획을 만든다(CUBRID는 실행 계획을 선택할 때 비용 기반의 질의 최적화를 수행한다).

USING INDEX 절은 **ORDER BY** 없이 원하는 순서로 결과를 얻고자 할 때 유용하게 사용될 수 있다. CUBRID는 인덱스 스캔을 하면 인덱스에 저장된 순서로 결과가 생성되는데, 한 테이블에 여러 인덱스가 있을 경우 특정 인덱스의 순서로 질의 결과를 얻고자 할 때 **USING INDEX** 를 사용할 수 있다.

```
SELECT ... WHERE ...
[USING INDEX { NONE | [ ALL EXCEPT ] <index_spec> [ {,
<index_spec> } ...] } ] [ ; ]

DELETE ... WHERE ...
[USING INDEX { NONE | [ ALL EXCEPT ] <index_spec> [ {,
<index_spec> } ...] } ] [ ; ]

UPDATE ... WHERE ...
[USING INDEX { NONE | [ ALL EXCEPT ] <index_spec> [ {,
<index_spec> } ...] } ] [ ; ]

<index_spec> ::=
[table_spec.]index_name [(+) | (-)] |
table_spec.NONE
```

- **NONE**: **NONE** 을 지정한 경우 모든 테이블에 대해서 순차 스캔이 사용된다.
- **ALL EXCEPT**: 질의 수행 시 지정한 인덱스를 제외한 모든 인덱스가 사용될 수 있다.
- *index_name*(+): 인덱스 이름 뒤에 (+)를 지정하면 해당 인덱스 선택이 우선시 된다. 해당 인덱스가 해당 질의를 수행하는데 적합하지 않으면 선택하지 않는다.
- *index_name*(-): 인덱스 이름 뒤에 (-)를 지정하면 해당 인덱스가 선택에서 제외된다.
- *table_spec*.**NONE**: 해당 테이블의 모든 인덱스가 선택에서 제외되어 순차 스캔이 사용된다.

USE, FORCE, IGNORE INDEX

FROM 절의 테이블 명세 뒤에 **USE, FORCE, IGNORE INDEX** 구문을 통해서 인덱스 힌트를 지정할 수 있다.

```
FROM table_spec [ <index_hint_clause> ] ...

<index_hint_clause> ::=
{ USE | FORCE | IGNORE } INDEX ( <index_spec> [,
<index_spec> ...] )

<index_spec> ::=
[table_spec.]index_name
```

- **USE INDEX** (<index_spec>): 지정한 인덱스들만 선택 시에 고려한다.

- **FORCE INDEX** (<index_spec>): 해당 인덱스 선택이 우선시 된다.
- **IGNORE INDEX** (<index_spec>): 지정한 인덱스들은 선택에서 제외된다.

USE, FORCE, IGNORE INDEX 구문은 시스템에 의해 자동적으로 적절한 USING INDEX 구문으로 재작성된다.

인덱스 힌트 사용 예

```
CREATE TABLE athlete2 (
  code          SMALLINT PRIMARY KEY,
  name          VARCHAR(40) NOT NULL,
  gender        CHAR(1),
  nation_code   CHAR(3),
  event         VARCHAR(30)
);
CREATE UNIQUE INDEX athlete2_idx1 ON athlete2 (code, nation_code);
CREATE INDEX athlete2_idx2 ON athlete2 (gender, nation_code);
```

아래 2개의 질의는 같은 동작을 수행하며, 지정된 *athlete2_idx2* 인덱스 스캔 비용이 순차 스캔 비용보다 작을 경우 해당 인덱스 스캔을 선택하게 된다.

```
SELECT /** RECOMPILE */ *
FROM athlete2 USE INDEX (athlete2_idx2)
WHERE gender='M' AND nation_code='USA';

SELECT /** RECOMPILE */ *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX athlete2_idx2;
```

아래 2개의 질의는 같은 동작을 수행하며, 항상 *athlete2_idx2*를 사용한다.

```
SELECT /** RECOMPILE */ *
FROM athlete2 FORCE INDEX (athlete2_idx2)
WHERE gender='M' AND nation_code='USA';

SELECT /** RECOMPILE */ *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX athlete2_idx2(+);
```

아래 2개의 질의는 같은 동작을 수행하며, 질의 수행 시 *athlete2_idx2*를 사용하지 않는다.

```

SELECT /** RECOMPILE */ *
FROM athlete2 IGNORE INDEX (athlete2_idx2)
WHERE gender='M' AND nation_code='USA';

SELECT /** RECOMPILE */ *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX athlete2_idx2(-);

```

다음 질의는 수행 시 항상 순차 스캔을 선택한다.

```

SELECT *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX NONE;

SELECT *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX athlete2.NONE;

```

다음 질의는 수행 시 *athlete2_idx2*를 제외한 모든 인덱스의 사용이 가능하도록 한다.

```

SELECT *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX ALL EXCEPT athlete2_idx2;

```

다음과 같이 **USE INDEX** 구문 또는 **USING INDEX** 구문에서 여러 인덱스를 지정한 경우 질의 최적화기는 지정된 인덱스 중 하나를 선택한다.

```

SELECT *
FROM athlete2 USE INDEX (athlete2_idx2, athlete2_idx1)
WHERE gender='M' AND nation_code='USA';

SELECT *
FROM athlete2
WHERE gender='M' AND nation_code='USA'
USING INDEX athlete2_idx2, athlete2_idx1;

```

여러 개의 테이블에 대해 질의를 수행하는 경우, 한 테이블에서는 특정 인덱스를 사용하여 인덱스 스캔을 하고 다른 테이블에서는 순차 스캔을 하도록 지정할 수 있다. 이러한 질의는 다음과 같은 형태가 된다.

```
SELECT *
FROM tab1, tab2
WHERE ...
USING INDEX tab1.idx1, tab2.NONE;
```

인덱스 힌트 구문이 있는 질의를 수행할 때 질의 최적화기는 인덱스가 지정되지 않는 테이블에 대해서는 해당 테이블의 사용 가능한 모든 인덱스를 고려한다. 예를 들어, *tab1* 테이블에는 인덱스 *idx1*, *idx2* 이 있고 *tab2* 테이블에는 인덱스 *idx3*, *idx4*, *idx5* 가 있는 경우, *tab1* 에 대한 인덱스만 지정하고 *tab2* 에 대한 인덱스를 지정하지 않으면 질의 최적화기는 *tab2* 의 인덱스도 고려하여 동작한다.

```
SELECT ...
FROM tab1, tab2 USE INDEX(tab1.idx1)
WHERE ... ;

SELECT ...
FROM tab1, tab2
WHERE ...
USING INDEX tab1.idx1;
```

위의 예제의 경우에 테이블 *tab1*의 순차 스캔과 *idx1* 인덱스 스캔을 비교하여 테이블 *tab1*의 스캔 방법을 선택하며, 테이블 *tab2*의 순차 스캔과 *idx3*, *idx4*, *idx5* 인덱스 스캔을 비교하여 테이블 *tab2*의 스캔 방법을 선택하게 된다.

특별한 인덱스

필터링된 인덱스

필터링된 인덱스(filtered index)는 한 테이블에 대해 잘 정의된 부분 집합을 정렬하거나 찾거나 연산해야 할 때 사용되며, 전체 데이터에서 조건에 부합하는 일부 데이터만 인덱스에 유지하므로 부분 인덱스(partial index)라고도 한다.

```
CREATE /** hints */ INDEX index_name
ON [schema_name.]table_name (col1, col2, ...)
WHERE <filter_predicate>;

ALTER /** hints */ INDEX index_name
[ ON [schema_name.]table_name (col1, col2, ...)
[ WHERE <filter_predicate> ] ]
REBUILD;
```

```
<filter_predicate> ::= <filter_predicate> AND <expression> |
<expression>
```

- *<filter_predicate>*: 칼럼과 상수 간 비교 조건. 조건이 여러 개인 경우 **AND** 로 연결된 경우에만 필터가 될 수 있다. 필터 조건으로 CUBRID에서 지원하는 대부분의 연산자와 함수가 포함될 수 있다. 그러나 현재 날짜/시간을 출력하는 날짜/시간 함수(예: `SYS_DATETIME()`), 랜덤 함수(예: `RAND()`)와 같이 같은 입력에 대해 다른 결과를 출력하는 함수는 허용되지 않는다.

필터링된 인덱스를 적용하여 질의를 처리하려면 **USE INDEX** 구문 또는 **FORCE INDEX** 구문을 통해 해당 필터링된 인덱스를 반드시 명시해야 한다.

- **USING INDEX** 절 또는 **USE INDEX** 구문을 통해 필터링된 인덱스를 명시하는 경우:

인덱스를 구성하는 칼럼이 **WHERE** 절의 조건에 포함되어 있지 않으면 필터링된 인덱스를 사용하지 않는다.

```
CREATE TABLE blogtopic
(
    blogID BIGINT NOT NULL,
    title VARCHAR(128),
    author VARCHAR(128),
    content VARCHAR(8096),
    postDate TIMESTAMP NOT NULL,
    deleted SMALLINT DEFAULT 0
);

CREATE INDEX my_filter_index ON blogtopic(postDate) WHERE deleted=0;
```

아래 질의에서 `my_filter_index`를 구성하는 칼럼인 `postDate`가 **WHERE** 조건에 포함되어 있으므로, **USE INDEX** 구문으로도 인덱스를 사용할 수 있다.

```
SELECT *
FROM blogtopic USE INDEX(my_filter_index)
WHERE postDate>'2010-01-01' AND deleted=0;
```

- **USING INDEX** *<index_name>(+)* 절 또는 **FORCE INDEX** 구문을 통해 필터링된 인덱스를 명시하는 경우:

인덱스를 구성하는 칼럼이 **WHERE** 절의 조건에 포함되어 있지 않더라도 필터링된 인덱스를 사용한다.

아래 질의에서는 `my_filter_index`의 인덱스를 구성하는 칼럼이 **WHERE** 조건에 포함되어 있지 않으므로, **USE INDEX** 구문으로는 인덱스를 사용할 수 없다.

```
SELECT *
FROM blogtopic USE INDEX (my_filter_index)
WHERE author = 'David' AND deleted=0;
```

따라서, my_filter_index 인덱스를 사용하려면 다음과 같이 **FORCE INDEX** 구문을 사용하여 인덱스 사용을 강제해야 한다.

```
SELECT *
FROM blogtopic FORCE INDEX (my_filter_index)
WHERE author = 'David' AND deleted=0;
```

다음은 버그/이슈를 유지하는 버그 트래킹 시스템의 예이다. 일정 기간의 개발 활동 이후 bugs 테이블에는 버그들이 기록되어 있는데, 이들 대부분은 오래 전에 종료된 상태이다. 버그 트래킹 시스템은 여전히 열린(open) 상태의 새로운 버그를 찾기 위해 해당 테이블에 질의를 한다. 이 경우 버그 테이블의 인덱스는 닫힌(closed) 버그의 레코드들에 대해 알 필요가 없다. 이런 경우 필터링된 인덱스는 열린 버그만 인덱싱하는 것을 허용한다.

```
CREATE TABLE bugs
(
    bugID BIGINT NOT NULL,
    CreationDate TIMESTAMP NOT NULL,
    Author VARCHAR(255),
    Subject VARCHAR(255),
    Description VARCHAR(255),
    CurrentStatus INTEGER,
    Closed SMALLINT
);
```

열린 상태의 버그만을 위한 인덱스는 다음 문장으로 생성될 수 있다.

```
CREATE INDEX idx_open_bugs ON bugs(bugID) WHERE Closed = 0;
```

열린 상태의 버그에만 관심있는 질의 처리를 위해 해당 인덱스를 인덱스 힌트로 지정하면, 필터링된 인덱스를 통하여 더 적은 인덱스 페이지를 접근하여 질의 결과를 생성할 수 있게 된다.

```
SELECT *
FROM bugs
WHERE Author = 'madden' AND Subject LIKE '%fopen%' AND Closed = 0
USING INDEX idx_open_bugs(+);

SELECT *
FROM bugs FORCE INDEX (idx_open_bugs)
WHERE CreationDate > CURRENT_DATE - 10 AND Closed = 0;
```


위의 예에서 “**USING INDEX** *idx_open_bugs*” 또는 “**USE INDEX** (*idx_open_bugs*)” 를 사용하는 경우, *idx_open_bugs* 인덱스를 사용하지 않고 질의를 수행하게 된다.

⚠ 경고

필터링된 인덱스 생성 조건과 질의 조건이 부합되지 않음에도 불구하고 인덱스 힌트 구문으로 인덱스를 명시하여 질의를 수행하면 명시된 인덱스를 선택하여 수행하므로, 주어진 검색 조건에 부합하지 않는 질의 결과를 출력할 수 있음에 주의한다.

i 참고

제약 사항

필터링된 인덱스는 일반 인덱스만 허용한다. 예를 들어, 필터링된 유일한(unique) 인덱스는 허용하지 않는다. 또한, 필터링된 인덱스를 구성하는 칼럼 값이 모두 NULL이 가능한 경우는 허용하지 않는다. 예를 들어, 아래의 경우는 Author 값이 NULL일 수 있으므로 허용하지 않는다.

```
CREATE INDEX idx_open_bugs ON bugs (Author) WHERE Closed = 0;
```

```
ERROR: before ' ; '
Invalid filter expression (bugs.Closed=0) for index.
```

하지만 아래의 경우는 Author 값이 NULL이더라도 CreationDate 값이 NULL일 수 없으므로 허용한다.

```
CREATE INDEX idx_open_bugs ON bugs (Author, CreationDate) WHERE
Closed = 0;
```

다음은 인덱스 필터 조건으로 허용하지 않는 경우이다.

- 날짜/시간 함수 또는 랜덤 함수와 같이 입력이 같은데 결과가 매번 다른 함수

```
CREATE INDEX idx ON bugs(creationdate) WHERE creationdate >
SYS_DATETIME;
```

```
ERROR: before ' ; '
'sys_datetime' is not allowed in a filter expression for index.
```

```
CREATE INDEX idx ON bugs(bugID) WHERE bugID > RAND();
```

```
ERROR: before ' ; '
'rand ' is not allowed in a filter expression for index.
```

- OR 연산자를 사용하는 경우

```
CREATE INDEX IDX ON bugs (bugID) WHERE bugID > 10 OR bugID = 3;
```

```
ERROR: before ' ; '
' or ' is not allowed in a filter expression for index.
```

- INCR(), DECR() 함수와 같이 테이블의 데이터를 수정하는 함수를 포함한 경우
- 시리얼 관련 함수와 의사 칼럼을 포함한 경우
- MIN(), MAX(), STDDEV() 등 집계 함수를 포함한 경우
- 인덱스를 생성할 수 없는 타입을 사용하는 함수
 - SET 타입을 인자로 받는 연산자와 함수
 - LOB 파일을 생성하는 함수 (CHAR_TO_BLOB(), CHAR_TO_CLOB(), BIT_TO_BLOB(), BLOB_FROM_FILE(), CLOB_FROM_FILE())
- IS NULL 연산자는 인덱스를 구성하는 칼럼들 중 적어도 하나가 NULL 이 아닐 경우에만 사용 가능

```
CREATE TABLE t (a INT, b INT);
```

```
-- IS NULL cannot be used with expressions
CREATE INDEX idx ON ta (a) WHERE (not ta.a<>0 ) IS NULL;
```

```
ERROR: before ' ; '
Invalid filter expression (( not [dba.ta].a<>0) is null ) for index.
```

```
CREATE INDEX idx ON t (a) WHERE (a+1) IS NULL;
```

```
ERROR: before ' ; '
Invalid filter expression ((t.a+1) is null ) for index.
```

```
-- At least one attribute must not be used with IS NULL
CREATE INDEX idx ON t(a,b) WHERE a IS NULL ;
```

```
ERROR: before ' ; '
Invalid filter expression (t.a is null ) for index.
```

```
CREATE INDEX idx ON t(a,b) WHERE a IS NULL and b IS NULL;
```

```
ERROR: before ' ; '
Invalid filter expression (t.a is null and t.b is null ) for
index.
```

```
CREATE INDEX idx ON t(a,b) WHERE a IS NULL and b IS NOT NULL;
```

- 필터링된 인덱스에 대한 index skip scan(ISS)은 지원되지 않는다.
- 필터링된 인덱스에서 사용되는 조건 문자열의 길이는 128자로 제한한다.

```
CREATE TABLE t(VeryLongColumnNameOfTypeInteger INT);

CREATE INDEX idx ON t(VeryLongColumnNameOfTypeInteger)
WHERE VeryLongColumnNameOfTypeInteger > 3 AND
VeryLongColumnNameOfTypeInteger < 10 AND
SQRT(VeryLongColumnNameOfTypeInteger) < 3 AND
SQRT(VeryLongColumnNameOfTypeInteger) < 10;
```

```
ERROR: before ' ; '
The maximum length of filter predicate string must be 128.
```

함수 기반 인덱스

함수 기반 인덱스(function-based index)는 특정 함수를 이용하여 테이블 행들로부터 값의 조합에 기반한 데이터를 정렬하거나 찾고 싶을 때 사용한다. 예를 들어, 공백을 무시한 문자열을 찾는 작업

을 하고 싶을 때 이러한 기능을 수행하는 함수를 이용하게 되는데, 함수를 통해 칼럼 값을 변경하게 되면 일반 인덱스를 통해서 인덱스 스캔을 할 수 없다. 이러한 경우에 함수 기반 인덱스를 생성하면 이를 통해 해당 질의 처리를 최적화할 수 있다. 다른 예로, 대소문자를 구분하지 않는 이름을 검색할 때 활용할 수 있다.

```
CREATE /** hints */ INDEX index_name
ON [schema_name.]table_name (function_name (argument_list));

ALTER /** hints */ INDEX index_name
[ ON [schema_name.]table_name (function_name (argument_list)) ]
REBUILD;
```

다음 인덱스가 생성된 이후 **SELECT** 질의는 자동으로 함수 기반 인덱스를 사용한다.

```
CREATE INDEX idx_trim_post ON posts_table(TRIM(keyword));

SELECT *
FROM posts_table
WHERE TRIM(keyword) = 'SQL';
```

LOWER 함수로 함수 기반 인덱스를 생성하면, 대소문자 구분을 안 하는 이름을 검색할 때 사용될 수 있다.

```
CREATE INDEX idx_last_name_lower ON clients_table(LOWER(LastName));

SELECT *
FROM clients_table
WHERE LOWER(LastName) = LOWER('Timothy');
```

질의 계획을 생성할 때 인덱스가 선택되게 하기 위해서는, 이 인덱스에서 사용되는 함수가 질의 조건에서 같은 방법으로 사용되어야 한다. 위의 **SELECT** 질의는 위에서 생성된 last_name_lower 인덱스를 사용한다. 하지만 다음과 같은 조건에서는 함수 기반 인덱스 형태와 다른 표현식이 주어졌기 때문에 인덱스가 사용되지 않는다.

```
SELECT *
FROM clients_table
WHERE LOWER(CONCAT('Mr. ', LastName)) = LOWER('Mr. Timothy');
```

함수 기반 인덱스의 사용을 강제하려면 **USING INDEX** 구문을 사용할 수 있다.

```
CREATE INDEX i_tbl_first_four ON tbl(LEFT(col, 4));

SELECT *
FROM clients_table
```

```
WHERE LEFT(col, 4) = 'CAT5'
USING INDEX i_tbl_first_four;
```

함수 기반 인덱스로 사용할 수 있는 함수는 다음과 같다.

ABS	ACOS	ADD_MONTHS	ADDDATE	ASIN
ATAN	ATAN2	BIT_COUNT	BIT_LENGTH	CEIL
CHAR_LENGTH	CHR	COS	COT	DATE
DATE_ADD	DATE_FORMAT	DATE_SUB	DATEDIFF	DAY
DAYOFMONTH	DAYOFWEEK	DAYOFYEAR	DEGREES	EXP
FLOOR	FORMAT	FROM_DAYS	FROM_UNIXTIME	GREATEST
HOUR	IFNULL	INET_ATON	INET_NTOA	INSTR
LAST_DAY	LEAST	LEFT	LN	LOCATE
LOG	LOG10	LOG2	LOWER	LPAD
LTRIM	MAKEDATE	MAKETIME	MD5	MID
MINUTE	MOD	MONTH	MONTHS_BETWEEN	NULLIF
NVL	NVL2	OCTET_LENGTH	POSITION	POWER

QUARTER	RADIANS	REPEAT	REPLACE	REVERSE
RIGHT	ROUND	RPAD	RTRIM	SECOND
SECTOTIME	SIN	SQRT	STR_TO_DATE	STRCMP
SUBDATE	SUBSTR	SUBSTRING	SUBSTRING_I NDEX	TAN
TIME	TIME_FORMA T	TIMEDIFF	TIMESTAMP	TIMETOSEC
TO_CHAR	TO_DATE	TO_DATETIME	TO_DAYS	TO_NUMBER
TO_TIME	TO_TIMESTAM P	TRANSLATE	TRIM	TRUNC
UNIX_TIMESTA MP	UPPER	WEEK	WEEKDAY	YEAR

함수 기반 인덱스에서 사용할 함수의 인자는 테이블의 칼럼 이름 혹은 상수인 경우만 허용하며, 복잡한 중첩된 표현식은 허용하지 않는다. 예를 들어 아래의 문장은 오류를 발생한다.

```
CREATE INDEX my_idx ON tbl (TRIM(LEFT(col, 3)));
CREATE INDEX my_idx ON tbl (LEFT(col1, col2 + 3));
```

묵시적인 타입 변환(implicit type cast)은 허용된다. 아래의 예에서 `LEFT()` 함수는 첫 번째 인자 타입이 **VARCHAR** 이고 두 번째 인자 타입이 **INTEGER** 여야 하지만 정상 동작한다.

```
CREATE INDEX my_idx ON tbl (LEFT(int_col, str_col));
```

함수 기반 인덱스는 필터링된 인덱스와 함께 사용될 수 없다. 아래의 예는 오류를 발생한다.

```
CREATE INDEX my_idx ON tbl (TRIM(col)) WHERE col > 'SQL';
```

함수 기반 인덱스는 다중 칼럼 인덱스가 될 수 없다. 아래의 예는 오류를 발생한다.

```
CREATE INDEX my_idx ON tbl (TRIM(col1), col2, LEFT(col3, 5));
```

인덱스를 활용한 최적화

커버링 인덱스

질의 수행 시 **SELECT** 리스트, **WHERE**, **HAVING**, **GROUP BY**, **ORDER BY** 절에 있는 모든 칼럼의 데이터를 포함하는 인덱스를 커버링 인덱스(covering index)라고 한다.

커버링 인덱스는 질의 수행 시 인덱스 내에 필요한 모든 데이터를 지니고 있어서 인덱스 페이지만 검색하면 되며, 데이터 저장소를 추가로 검색할 필요가 없어 데이터 저장소 접근을 위한 I/O 비용을 줄일 수 있다. 데이터 검색 속도를 향상시키기 위해 커버링 인덱스로 생성하는 것을 고려할 수 있지만, 인덱스의 크기가 커지면 **INSERT** 와 **DELETE** 작업은 느려질 수 있다는 점을 감안해야 한다.

커버링 인덱스의 적용 여부에 대한 규칙은 다음과 같다.

- CUBRID 질의 최적화기는 커버링 인덱스의 적용이 가능하면 이를 가장 먼저 사용한다.
- 조인 질의의 경우 인덱스가 **SELECT** 리스트에 있는 테이블의 칼럼을 포함하면, 이 인덱스를 사용한다.
- 인덱스를 사용할 수 있는 조건이 아닌 경우 커버링 인덱스를 사용할 수 없다.

```
CREATE TABLE t (col1 INT, col2 INT, col3 INT);
CREATE INDEX i_t_col1_col2_col3 ON t (col1,col2,col3);
INSERT INTO t VALUES (1,2,3),(4,5,6),(10,8,9);
```

다음의 예는 **SELECT** 하는 칼럼과 **WHERE** 조건의 칼럼이 모두 인덱스 내에 존재하므로, 해당 인덱스가 커버링 인덱스로 사용된다.

```
-- csql> ;plan simple
SELECT * FROM t WHERE col1 < 6;
```

```
Query plan:
  Index scan(t t, i_t_col1_col2_col3, [(t.col1 range (min inf_lt
t.col3))] (covers))
```

col1	col2	col3
1	2	3
4	5	6

다음의 예는 커버링 인덱스 사용시 **SELECT** 리스트에 **COUNT(*)** 만 있는 경우 불필요한 스캔을 줄여 주는 최적화를 보여준다.

```
-- insert dummy data
INSERT INTO t select rownum % 8, rownum % 100, rownum % 1000 FROM
dual connect by level <= 40000;

-- csql> ;trace on

-- count(*) optimization
SELECT count(*) FROM t WHERE col1 < 9;
```

```
Trace Statistics:
  SELECT (time: 1, fetch: 66, fetch_time: 0, ioread: 0)
    SCAN (index: dba.t.i_t_col1_col2_col3), (btree time: 1, fetch:
65, ioread: 0, readkeys: 1002, filteredkeys: 0, rows: 0, covered:
true, count_only: true)
```

```
-- no count(*) optimization
SELECT count(col1) FROM t WHERE col1 < 9;
```

```
Trace Statistics:
  SELECT (time: 13, fetch: 180, fetch_time: 0, ioread: 0)
    SCAN (index: dba.t.i_t_col1_col2_col3), (btree time: 8, fetch:
179, ioread: 0, readkeys: 1002, filteredkeys: 0, rows: 40002,
covered: true)
```

참고

커버링 인덱스 사용시 단일 절의 **COUNT(*)** 최적화는 CUBRID 11.2 부터 지원되고, 조인 테이블에 대한 **COUNT(*)** 최적화는 CUBRID 11.3 부터 지원된다.

경고

VARCHAR 타입의 칼럼에서 값을 가져올 때 커버링 인덱스가 적용되는 경우, 뒤에 따라오는 공백 문자열은 잘리게 된다. 질의 최적화 수행 시 커버링 인덱스가 적용되면 질의 결과 값을 인덱스에서 가져오는데, 인덱스에는 뒤이어 나타나는 공백 문자열을 제거한 채로 값을 저장하기 때문이다.

이러한 현상을 원하지 않는다면 커버링 인덱스 기능을 사용하지 않도록 하는 **NO_COVERING_IDX** 힌트를 사용한다. 이 힌트를 사용하면 결과값을 인덱스 영역이 아닌 데이터 영역에서 가져오도록 한다.

다음은 위의 상황의 자세한 예이다. 먼저 **VARCHAR** 타입의 칼럼을 갖는 테이블을 생성하고, 여기에 시작 문자열의 값이 같고 문자열 뒤에 따르는 공백 문자의 개수가 다른 값을 **INSERT** 한다. 그리고 해당 칼럼에 인덱스를 생성한다.

```
CREATE TABLE tab(c VARCHAR(32));
INSERT INTO tab VALUES('abcd'),('abcd '),('abcd ');
CREATE INDEX i_tab_c ON tab(c);
```

인덱스를 반드시 사용하도록(커버링 인덱스가 적용되도록) 했을 때의 질의 결과는 다음과 같다.

```
-- csql>;plan simple
SELECT * FROM tab WHERE c='abcd ' USING INDEX i_tab_c(+);
```

```
Query plan:
  Index scan(tab tab, i_tab_c, (tab.c='abcd ') (covers))

  c
  =====
'abcd'
'abcd'
'abcd'
```

다음은 인덱스를 사용하지 않도록 했을 때의 질의 결과이다.

```
SELECT * FROM tab WHERE c='abcd ' USING INDEX tab.NONE;
```

```
Query plan:
  Sequential scan(tab tab)

  c
  =====
'abcd'
'abcd '
'abcd '
```

위의 두 결과 비교에서 알 수 있듯이, 커버링 인덱스가 적용되면 **VARCHAR** 타입에서는 인덱스로부터 값을 가져오면서 뒤이어 나타나는 공백 문자열이 잘린 채로 나타난다.

참고

커버링 인덱스 최적화가 적용될 수 있으면 디스크 입출력을 상당히 줄일 수 있기 때문에 성능 향상을 기대할 수 있다. 하지만 특정한 상황에서 커버링 인덱스 스캔 최적화를 원하지 않는다면, 질의에 **NO_COVERING_IDX** 힌트를 명시하면 된다. 힌트를 지정하는 방법은 [SQL 힌트](#)를 참고하면 된다.

ORDER BY 절 최적화

ORDER BY 절에 있는 모든 칼럼을 포함하는 인덱스를 정렬된 인덱스(ordered index)라고 한다. **ORDER BY** 절이 있는 질의를 최적화하면 정렬된 인덱스를 통해 질의 결과를 탐색하므로 별도의 정렬 과정을 거치지 않는다(skip order by). 정렬된 인덱스가 되기 위한 일반적인 조건은 **ORDER BY** 절에 있는 칼럼들이 인덱스의 가장 앞부분에 위치하는 경우이다.

```
SELECT *
FROM tab
WHERE col1 > 0
ORDER BY col1, col2;
```

- *tab (col1, col2)* 으로 구성된 인덱스는 정렬된 인덱스이다.
- *tab (col1, col2, col3)* 으로 구성된 인덱스도 정렬된 인덱스이다. **ORDER BY** 절에서 참조하지 않는 *col3* 는 *col1, col2* 뒤에 오기 때문이다.
- *tab (col1)* 으로 구성된 인덱스는 정렬된 인덱스가 아니다.
- *tab (col3, col1, col2)* 혹은 *tab (col1, col3, col2)*로 구성된 인덱스는 최적화에 사용할 수 없다. 이는 *col3* 가 **ORDER BY** 절의 칼럼들 뒤에 위치하지 않기 때문이다.

인덱스를 구성하는 칼럼이 **ORDER BY** 절에 없더라도 그 칼럼의 조건이 상수일 때는 정렬된 인덱스의 사용이 가능하다.

```
SELECT *
FROM tab
WHERE col2=val
ORDER BY col1,col3;
```

*tab (col1, col2, col3)*로 구성된 인덱스가 존재하고 *tab (col1, col2)*로 구성된 인덱스는 없이 위의 질의를 수행할 때, 질의 최적화기는 *tab (col1, col2, col3)*로 구성된 인덱스를 정렬된 인덱스로 사용한다. 즉, 인덱스 스캔 시 요구하는 순서대로 결과를 가져오므로, 레코드를 정렬할 필요가 없다.

정렬된 인덱스와 커버링 인덱스를 함께 사용할 수 있으면 커버링 인덱스를 먼저 사용한다. 커버링 인덱스를 사용하면 요청한 데이터의 결과가 인덱스 페이지에 모두 들어 있어 추가적인 데이터를 검색할 필요가 없으며, 이 인덱스가 순서까지 만족한다면, 결과를 정렬할 필요가 없기 때문이다.

질의가 조건을 포함하지 않으며 정렬된 인덱스를 사용할 수 있다면, 인덱스의 첫 번째 칼럼이 **NOT NULL** 조건을 만족한다는 전제 하에서는 정렬된 인덱스가 사용될 것이다.

```
CREATE TABLE tab (i INT, j INT, k INT);
CREATE INDEX i_tab_j_k ON tab (j, k);
INSERT INTO tab VALUES (1,2,3), (6,4,2), (3,4,1), (5,2,1), (1,5,5),
(2,6,6), (3,5,4);
```

다음의 예는 j, k 칼럼으로 **ORDER BY** 를 수행하므로 $tab(j, k)$ 로 구성된 인덱스는 정렬된 인덱스가 되고 별도의 정렬 과정을 거치지 않는다.

```
SELECT i,j,k
FROM tab
WHERE j > 0
ORDER BY j,k;
```

```
-- the selection from the query plan dump shows that the ordering
index i_tab_j_k was used and sorting was not necessary
-- (/* --> skip ORDER BY */)
Query plan:
iscan
  class: tab node[0]
  index: i_tab_j_k term[0]
  sort:  2 asc, 3 asc
  cost:  1 card 0
Query stmt:
select tab.i, tab.j, tab.k from tab tab where ((tab.j> ?:0 )) order
by 2, 3
/* ---> skip ORDER BY */
```

i	j	k
5	2	1
1	2	3
3	4	1
6	4	2
3	5	4
1	5	5
2	6	6

다음의 예는 j, k 칼럼으로 **ORDER BY** 를 수행하며 **SELECT** 하는 칼럼을 모두 포함하는 인덱스가 존재하므로 $tab(j,k)$ 로 구성된 인덱스가 커버링 인덱스로서 사용된다. 따라서 인덱스 자체에서 값을 가져 오게 되며 별도의 정렬 과정을 거치지 않는다.

```
SELECT /*+ RECOMPILE */ j,k
FROM tab
WHERE j > 0
ORDER BY j,k;
```

```
-- in this case the index i_tab_j_k is a covering index and also
-- respects the ordering index property.
-- Therefore, it is used as a covering index and sorting is not
-- performed.
```

Query plan:

```
iscan
  class: tab node[0]
  index: i_tab_j_k term[0] (covers)
  sort:  1 asc, 2 asc
  cost:  1 card 0
```

```
Query stmt: select tab.j, tab.k from tab tab where ((tab.j> ?:0 ))
order by 1, 2
/* ---> skip ORDER BY */
```

j	k
2	1
2	3
4	1
4	2
5	4
5	5
6	6

다음의 예는 i 칼럼 조건이 있으며 j, k 칼럼으로 **ORDER BY** 를 수행하고, **SELECT** 하는 칼럼이 i, j, k 이므로 $tab(i, j, k)$ 로 구성된 인덱스가 커버링 인덱스로서 사용된다. 따라서 인덱스 자체에서 값을 가져 오게 되지만, **ORDER BY** j, k 에 대한 별도의 정렬 과정을 거친다.

```
CREATE INDEX i_tab_j_k ON tab (i,j,k);
SELECT /*+ RECOMPILE */ i,j,k
FROM tab
WHERE i > 0
ORDER BY j,k;
```

```
-- since an index on (i,j,k) is now available, it will be used as
covering index. However, sorting the results according to
-- the ORDER BY clause is needed.
```

Query plan:

```
temp(order by)
  subplan: iscan
            class: tab node[0]
            index: i_tab_i_j_k term[0] (covers)
            sort:  1 asc, 2 asc, 3 asc
            cost:  1 card 1
      sort:  2 asc, 3 asc
      cost:  7 card 1
```

Query stmt: select tab.i, tab.j, tab.k from tab tab where ((tab.i>?:
0)) order by 2, 3

i	j	k
5	2	1
1	2	3
3	4	1
6	4	2
3	5	4
1	5	5
2	6	6

참고

CAST() 연산자 등을 통해 **ORDER BY** 절의 칼럼이 타입 변환되더라도, 타입 변환 전의 정렬 순서와 타입 변환 이후의 정렬 순서가 같다면 **ORDER BY** 절 최적화가 수행된다.

변환 전	변환 이후
수치형 타입	수치형 타입
문자열 타입	문자열 타입
DATETIME	TIMESTAMP
TIMESTAMP	DATETIME

변환 전	변환 이후
DATETIME	DATE
TIMESTAMP	DATE
DATE	DATETIME

내림차순 인덱스 스캔

다음과 같이 내림차순 정렬이 있는 질의를 수행할 때 일반적으로 내림차순 인덱스를 생성하여 인덱스를 사용하도록 하면 별도의 정렬 과정이 필요 없다.

```
SELECT *
FROM tab
[WHERE ...]
ORDER BY a DESC;
```

그런데 같은 칼럼에 대해 오름차순 인덱스와 내림차순 인덱스를 생성하면 교착 상태(deadlock)의 발생 가능성이 높아진다. 이러한 경우를 줄이기 위해 CUBRID는 별도의 내림차순 인덱스를 생성하지 않아도, 오름차순 인덱스만으로 내림차순 인덱스 스캔을 사용할 수 있다. 사용자는 **USE_DESC_IDX** 힌트를 사용하여 내림차순 스캔을 사용하도록 명시할 수 있다. 이 힌트가 명시되지 않으면 **ORDER BY** 절에 나열된 칼럼이 인덱스를 사용할 수 있다는 전제 조건 하에서 아래의 3가지 질의 실행 계획을 고려할 수 있다.

- 순차 스캔 + 내림차순 정렬
- 일반적인 오름차순 스캔 + 내림차순 정렬
- 별도의 정렬 작업이 필요 없는 내림차순 스캔

내림차순 스캔을 위해 **USE_DESC_IDX** 힌트가 생략된다 하더라도 질의 최적화기는 위에서 나열한 3가지 중 제일 마지막 실행 계획을 최적의 계획으로 결정한다.

USE_DESC_IDX 힌트는 조인 질의에 대해서는 지원하지 않는다.

```
CREATE TABLE di (i INT);
CREATE INDEX i_di_i ON di (i);
INSERT INTO di VALUES (5),(3),(1),(4),(3),(5),(2),(5);
```

다음 예는 **USE_DESC_IDX** 힌트 없이 오름차순 스캔을 통해 질의를 수행한다.

```
-- The query will be executed with an ascending scan.

SELECT *
FROM di
WHERE i > 0
LIMIT 3;
```

Query plan:

```
Index scan(di di, i_di_i, (di.i range (0 gt_inf max) and inst_num()
range (min inf_le 3)) (covers))
```

```

      i
=====
      1
      2
      3
```

위의 질의에 **USE_DESC_IDX** 힌트를 추가하면 내림차순 스캔을 통해 다른 결과가 나온다.

```
-- We now run the same query, using the 'use_desc_idx' SQL hint:

SELECT /*+ USE_DESC_IDX */ *
FROM di
WHERE i > 0
LIMIT 3;
```

Query plan:

```
Index scan(di di, i_di_i, (di.i range (0 gt_inf max) and inst_num()
range (min inf_le 3)) (covers) (desc_index))
```

```

      i
=====
      5
```

```
5
5
```

다음 예는 **ORDER BY** 절을 통해 내림차순 정렬이 요구되는 경우이다. 이 경우 **USE_DESC_IDX** 힌트가 없지만 내림차순 스캔하게 된다.

```
-- We also run the same query, this time asking that the results are
-- displayed in descending order.
-- However, no hint is given.
-- Since ORDER BY...DESC clause exists, CUBRID will use descending
-- scan, even though the hint is not given,
-- thus avoiding to sort the records.
```

```
SELECT *
FROM di
WHERE i > 0
ORDER BY i DESC LIMIT 3;
```

```
Query plan:
  Index scan(di di, i_di_i, (di.i range (0 gt_inf max)) (covers)
(desc_index))
```

```
      i
=====
      5
      5
      5
```

GROUP BY 절 최적화

GROUP BY 절에 있는 모든 칼럼이 인덱스에 포함되어 질의 수행 시 인덱스를 사용할 수 있으므로 별도의 정렬 작업을 하지 않는 것을 **GROUP BY** 절 최적화라고 한다. 이를 위해서는 **GROUP BY** 절에 있는 칼럼들이 인덱스를 구성하는 칼럼들의 제일 앞 쪽에 모두 존재해야 한다.

```
SELECT *
FROM tab
WHERE col1 > 0
GROUP BY col1,col2;
```

- `tab (col1, col2)`로 구성된 인덱스는 최적화에 사용할 수 있다.
- `tab (col1, col2, col3)`로 구성된 인덱스도 사용될 수 있는데, **GROUP BY** 절에서 참조하지 않는 `col3`는 `col1, col2` 뒤에 오기 때문이다.

- `tab (col1)`로 구성된 인덱스는 최적화에 사용할 수 없다.
- `tab (col3, col1, col2)` 혹은 `tab (col1, col3, col2)`로 구성된 인덱스도 최적화에 사용할 수 없는데, `col3`가 **GROUP BY** 절의 칼럼들 뒤에 위치하지 않기 때문이다.

인덱스를 구성하는 칼럼이 **GROUP BY** 절에 없더라도 그 칼럼의 조건이 상수일 때는 인덱스를 사용할 수 있다.

```
SELECT *
FROM tab
WHERE col2=val
GROUP BY col1,col3;
```

위의 예에서 `tab (col1, col2, col3)`로 구성된 인덱스가 있으면 이 인덱스를 **GROUP BY** 최적화에 사용한다.

이 경우에도 인덱스 스캔 시 요구하는 순서대로 결과를 가져오므로, **GROUP BY**에 의해서 행에 대한 정렬이 불필요하게 된다.

WHERE 절이 없어도 **GROUP BY** 칼럼으로 구성된 인덱스가 있고 그 인덱스의 첫번째 칼럼이 **NOT NULL**이면 **GROUP BY** 최적화가 적용된다.

집계 함수 사용 시에도 **GROUP BY** 칼럼으로 구성된 인덱스가 있으면 **GROUP BY** 최적화가 적용된다.

```
CREATE INDEX i_T_a_b_c ON T(a, b, c);
SELECT a, MIN(b), c, MAX(b) FROM T WHERE a > 18 GROUP BY a, b;
```

참고

GROUP BY 절 또는 **DISTINCT**의 칼럼이 인덱스 부분 키(subkey)를 포함할 때, 부분 키를 구성하는 칼럼 각각의 고유(unique) 값에 대해 동적으로 범위를 조정하여 B-트리 검색을 시작한다. 이와 관련하여 [Loose Index Scan](#)을 참고한다.

예제

```
CREATE TABLE tab (i INT, j INT, k INT);
CREATE INDEX i_tab_j_k ON tab (j, k);
INSERT INTO tab VALUES (1,2,3), (6,4,2), (3,4,1), (5,2,1), (1,5,5),
(2,6,6), (3,5,4);

UPDATE STATISTICS on tab;
```

다음의 예는 j, k 칼럼으로 **GROUP BY** 를 수행하므로 $tab(j, k)$ 로 구성된 인덱스가 사용되고 별도의 정렬 과정이 필요 없다.

```
SELECT /*+ RECOMPILE */ j,k
FROM tab
WHERE j > 0
GROUP BY j,k;

-- the selection from the query plan dump shows that the index
-- i_tab_j_k was used and sorting was not necessary
-- (/* ---> skip GROUP BY */)
```

Query plan:

```
iscan
  class: tab node[0]
  index: i_tab_j_k term[0]
  sort:  2 asc, 3 asc
  cost:  1 card 0
```

Query stmt:

```
select tab.i, tab.j, tab.k from tab tab where ((tab.j> ?:0 )) group
by tab.j, tab.k
```

```
/* ---> skip GROUP BY */
```

i	j	k
5	2	1
1	2	3
3	4	1
6	4	2
3	5	4
1	5	5
2	6	6

다음의 예는 j, k 칼럼으로 **GROUP BY** 를 수행하며 j 에 대한 조건이 없지만 j 칼럼은 **NOT NULL** 속성을 지니므로, $tab(j, k)$ 로 구성된 인덱스가 사용되고 별도의 정렬 과정이 필요 없다.

```
ALTER TABLE tab CHANGE COLUMN j j INT NOT NULL;
```

```
SELECT *
FROM tab
GROUP BY j,k;
```

```
-- the selection from the query plan dump shows that the index
-- i_tab_j_k was used (since j has the NOT NULL constraint )
-- and sorting was not necessary (/* ---> skip GROUP BY */)
```

Query plan:

```
iscan
```

```

class: tab node[0]
index: i_tab_j_k
sort:  2 asc, 3 asc
cost:  1 card 0

```

Query stmt: select tab.i, tab.j, tab.k **from** tab tab group by tab.j, tab.k

/* ---> skip GROUP BY */

=== <Result of SELECT Command in Line 1> ===

i	j	k
5	2	1
1	2	3
3	4	1
6	4	2
3	5	4
1	5	5
2	6	6

```

CREATE TABLE tab (k1 int, k2 int, k3 int, v double);
INSERT INTO tab
  SELECT
    RAND(CAST(UNIX_TIMESTAMP() AS INT)) MOD 5,
    RAND(CAST(UNIX_TIMESTAMP() AS INT)) MOD 10,
    RAND(CAST(UNIX_TIMESTAMP() AS INT)) MOD 100000,
    RAND(CAST(UNIX_TIMESTAMP() AS INT)) MOD 100000
  FROM db_class a, db_class b, db_class c, db_class d LIMIT 20000;
CREATE INDEX idx ON tab(k1, k2, k3);

```

위의 테이블과 인덱스를 생성했을 때 다음의 예는 k1, k2 칼럼으로 **GROUP BY**를 수행하며 k3로 집계 함수를 수행하므로 tab(k1, k2, k3)로 구성된 인덱스가 사용되고 별도의 정렬 과정이 필요 없다. 또한 **SELECT** 리스트에 있는 k1, k2, k3 칼럼이 모두 tab(k1, k2, k3)로 구성된 인덱스 내에 존재하므로 커버링 인덱스가 적용된다.

```

SELECT /*+ RECOMPILE INDEX_SS */ k1, k2, SUM(DISTINCT k3)
FROM tab
WHERE k2 > -1 GROUP BY k1, k2;

```

Query plan:

```

iscan
  class: tab node[0]
  index: idx term[0] (covers) (index skip scan)
  sort:  1 asc, 2 asc
  cost:  85 card 2000

```

Query stmt:

```
select tab.k1, tab.k2, sum(distinct tab.k3) from tab tab where
(tab.k2 > ? : 0 ) group by tab.k1, tab.k2

/* ---> skip GROUP BY */
```

다음의 예는 $k1$, $k2$ 칼럼으로 **GROUP BY**를 수행하므로 $tab^{*}(k1, k2, k3)$ 로 구성된 인덱스가 사용되고 별도의 정렬 과정이 필요 없다. 하지만 **SELECT** 리스트에 있는 v 칼럼은 $tab^{*}(k1, k2, k3)$ 로 구성된 인덱스 내에 존재하지 않으므로 커버링 인덱스가 적용되지 않는다.

```
SELECT /*+ RECOMPILE INDEX_SS */ k1, k2, stddev_samp(v)
FROM tab
WHERE k2 > -1 GROUP BY k1, k2;
```

Query plan:

```
iscan
  class: tab node[0]
  index: idx term[0] (index skip scan)
  sort:  1 asc, 2 asc
  cost:  85 card 2000
```

Query stmt:

```
select tab.k1, tab.k2, stddev_samp(tab.v) from tab tab where
(tab.k2 > ? : 0 ) group by tab.k1, tab.k2

/* ---> skip GROUP BY */
```

다중 키 범위 최적화

대부분의 질의가 **LIMIT** 절을 포함하고 있기 때문에 **LIMIT** 절을 최적화하는 것이 질의 성능에 매우 중요한데, 이에 해당하는 대표적인 최적화가 다중 키 범위 최적화(multiple key range optimization)이다.

다중 키 범위 최적화는 결과 생성에 필요한 인덱스 범위 전체를 스캔하지 않고, 인덱스 내의 일부 키 범위만 스캔하면서 Top N 정렬 방식을 통해 질의 결과를 생성한다. Top N 정렬은 전체 결과를 생성한 후에 이를 정렬하여 결과를 얻는 것이 아니라, 항상 최적의 N 개의 결과를 유지하는 방식으로 질의를 처리하기 때문에 매우 뛰어난 성능을 보인다.

예를 들어 내 친구들이 쓴 글 중에서 가장 최근 글을 10 개만 검색하는 경우, 다중 키 범위 최적화가 적용되면 내 전체 친구가 쓴 글을 모두 찾아서 정렬한 후에 결과를 찾지 않고 각 친구가 쓴 최근 글 10 개씩만을 찾아서 정렬을 유지하고 있는 인덱스를 스캔하여 결과를 찾는다.

다중 키 범위 최적화를 사용할 수 있는 예는 다음과 같다.

```
CREATE TABLE t (a int, b int);
CREATE INDEX i_t_a_b ON t (a,b);

-- Multiple key range optimization
SELECT *
FROM t
WHERE a IN (1,2,3)
ORDER BY b
LIMIT 2;
```

```
Query plan:
iscan
class: t node[0]
index: i_t_a_b term[0] (covers) (multi_range_opt)
sort: 1 asc, 2 asc
cost: 1 card 0
```

단일 테이블에서는 다음과 같은 조건들이 만족되었을 경우에 다중 키 범위 최적화가 수행된다.

```
SELECT /*+ hints */ ...
FROM table
WHERE col_1 = ? AND col_2 = ? AND ... AND col(j-1) = ?
AND col_(j) IN (?, ?, ...)
AND col_(j+1) = ? AND ... AND col_(p-1) = ?
AND key_filter_terms
ORDER BY col_(p) [ASC|DESC], col_(p+1) [ASC|DESC], ... col_(p+k-1)
[ASC|DESC]
LIMIT n;
```

먼저 **LIMIT** 절을 통해서 지정된 최종 결과의 상한 크기(n)가 **multi_range_optimization_limit** 시스템 파라미터 값보다 작거나 같아야 한다.

또한 다중 키 범위 최적화에 적합한 인덱스가 필요한데, **ORDER BY** 절에 명시된 모든 k 개의 칼럼을 커버해야 한다. 즉, 인덱스 상에서 **ORDER BY** 절에 명시된 칼럼들을 모두 포함해야 하고, 칼럼들의 순서와 정렬 방향이 일치해야 한다. 또한 **WHERE** 절에서 사용되는 모든 칼럼을 포함해야 한다.

인덱스를 구성하는 칼럼들 중

- 범위 조건(예를 들어, IN 조건) 앞의 칼럼들은 동일(=) 조건으로 표현된다.
- 범위 조건을 가진 칼럼이 하나만 존재한다.
- 범위 조건 이후의 칼럼들은 키 필터로 존재한다.
- 데이터 필터 조건이 없어야 한다. 다시 말해, 인덱스는 **WHERE** 절에서 사용되는 모든 칼럼을 포함해야 한다.

- 키 필터 이후의 칼럼들은 **ORDER BY** 절에 존재한다.
- 키 필터 조건의 칼럼들은 반드시 **ORDER BY** 절의 칼럼이 아니어야 한다.
- 상관 부질의(correlated subquery)를 포함한 키 필터 조건이 포함되어 있다면, 이에 연관된 칼럼은 범위 조건이 아닌 조건으로 WHERE 절에 포함되어야 한다.

다음과 같은 예에 다중 키 범위 최적화가 수행된다.

```
CREATE TABLE t (a INT, b INT, c INT, d INT, e INT);
CREATE INDEX i_t_a_b ON t (a,b,c,d,e);

SELECT *
FROM t
WHERE a = 1 AND b = 3 AND c IN (1,2,3) AND d = 3
ORDER BY e
LIMIT 2;
```

다중 테이블을 포함하는 JOIN 질의에서는 다음의 경우 최적화가 수행된다.

```
SELECT /*+ hints */ ...
FROM table_1, table_2, ... table_(sort), ...
WHERE col_1 = ? AND col_2 = ? AND ...
AND col_(j) IN (?, ?, ... )
AND col_(j+1) = ? AND ... AND col_(p-1) = ?
AND key_filter_terms
AND join_terms
ORDER BY col_(p) [ASC|DESC], col_(p+1) [ASC|DESC], ... col_(p+k-1)
[ASC|DESC]
LIMIT n;
```

JOIN 질의에 대해서 다중 키 범위 최적화가 적용되기 위해서는 다음과 같은 조건을 만족해야 한다.

- **ORDER BY** 절에 존재하는 칼럼들은 하나의 테이블에만 존재하는 칼럼들이며, 이 테이블은 단일 테이블 질의에서 다중 키 범위 최적화에 의해 요구되는 조건을 모두 만족해야 한다. 이 테이블을 정렬 테이블(sort table)이라고 하자.
- 정렬 테이블과 외부 테이블들(outer tables) 간의 JOIN 조건에 명시된 정렬 테이블의 칼럼들은 모두 인덱스에 포함되어야 한다. 즉, 데이터 필터링 조건이 없어야 한다.
- 정렬 테이블과 내부 테이블들(inner tables) 간의 JOIN 조건에 명시된 정렬 테이블의 칼럼들은 범위 조건이 아닌 조건으로 WHERE 절에 포함되어야 한다.

참고

다중 키 범위 최적화가 적용될 수 있는 대부분의 경우에 다중 키 범위 최적화가 가장 좋은 성능을 보장하지만, 특정한 상황에서 최적화를 원하지 않는다면 질의에 **NO_MULTI_RANGE_OPT** 힌트를 명시하면 된다. 힌트를 지정하는 방법은 **SQL 힌트**를 참고하면 된다.

Index Skip Scan

Index Skip Scan(이하 ISS)은 인덱스의 첫 번째 칼럼이 조건에 명시되지 않아도 뒤따라오는 칼럼이 조건(주로 =)에 명시되면 해당 인덱스를 활용하여 질의를 처리하는 최적화 방식이다.

질의문 힌트를 통해 특정 테이블에 대한 **INDEX_SS**가 입력되고 다음의 경우를 만족할 때 ISS의 적용이 고려된다.

1. 복합 인덱스의 두번째 칼럼부터 조건에 명시된다.
2. 사용되는 인덱스가 필터링된 인덱스가 아니어야 한다.
3. 인덱스의 첫 번째 칼럼이 범위 필터나 키 필터가 아니어야 한다.
4. 계층 질의는 지원하지 않는다.
5. 집계 함수가 포함된 경우는 지원하지 않는다.

INDEX_SS 힌트에는 ISS의 적용을 고려할 테이블 리스트를 입력할 수 있으며, 테이블 리스트가 생략되는 경우 모든 테이블에 대해 ISS의 적용이 고려된다.

```
/** INDEX_SS */
/** INDEX_SS(tbl1) */
/** INDEX_SS(tbl1, tbl2) */
```

참고

“INDEX_SS”를 입력하면 모든 테이블에 ISS 힌트가 적용되지만, “INDEX_SS()”를 입력하면 힌트가 무시된다.

```
CREATE TABLE t1 (id INT PRIMARY KEY, a INT, b INT, c INT);
CREATE TABLE t2 (id INT PRIMARY KEY, a INT, b INT, c INT);
CREATE INDEX i_t1_ac ON t1(a,c);
CREATE INDEX i_t2_ac ON t2(a,c);

INSERT INTO t1 SELECT rownum, rownum, rownum, rownum
FROM db_class x1, db_class x2, db_class LIMIT 10000;
```

```
INSERT INTO t2 SELECT id, a%5, b, c FROM t1;
```

```
SELECT /*+ INDEX_SS */ *
FROM t1, t2
WHERE t1.b<5 AND t1.c<5 AND t2.c<5
USING INDEX i_t1_ac, i_t2_ac limit 1;
```

```
SELECT /*+ INDEX_SS(t1) */ *
FROM t1, t2
WHERE t1.b<5 AND t1.c<5 AND t2.c<5
USING INDEX i_t1_ac, i_t2_ac LIMIT 1;
```

```
SELECT /*+ INDEX_SS(t1, t2) */ *
FROM t1, t2
WHERE t1.b<5 AND t1.c<5 AND t2.c<5
USING INDEX i_t1_ac, i_t2_ac LIMIT 1;
```

일반적으로 ISS는 여러 개의 칼럼들(C1, C2, ..., Cn) 중에서 고려되어야 하는데, 여기에서 질의는 연속된 칼럼들에 대한 조건을 가지고 있고 이 조건들은 인덱스의 두 번째 칼럼(C2)부터 시작한다.

```
INDEX (C1, C2, ..., Cn);
```

```
SELECT ... WHERE C2 = x AND C3 = y AND ... AND Cp = z; -- p <= n
SELECT ... WHERE C2 < x AND C3 >= y AND ... AND Cp BETWEEN (z AND
w); -- other conditions than equal
```

질의 최적화기는 궁극적으로 비용에 따라 ISS가 최적의 접근 방식인지 비용을 감안하여 결정한다. ISS는 인덱스의 첫 번째 칼럼이 레코드 개수에 비해 고유한(**DISTINCT**) 값의 개수가 적은 경우와 같이 특정한 상황에서 적용되며, 이 경우 인덱스 전체 검색(index full scan)보다 더 우수한 성능을 발휘한다. 예를 들어, 인덱스 칼럼 중에 첫 번째 칼럼이 남성/여성의 값 또는 수백만 건의 레코드가 1~100 사이의 값을 가지는 것처럼 DISTINCT 값의 개수가 매우 낮고(값의 중복도가 높고), 이 칼럼 조건이 질의 조건에 명시되지 않은 경우에 질의 최적화기는 ISS 적용을 검토하게 된다.

인덱스 전체 검색은 인덱스 리프 전체를 모두 다 읽어야 하지만, ISS는 동적으로 재조정되는 범위 검색(range search)을 사용하여 대부분의 인덱스 페이지 읽기를 생략하면서 질의를 처리한다. 값의 중복도가 높을수록 읽기를 생략할 수 있는 인덱스 페이지가 많아질 수 있기 때문에 ISS의 효율이 높아질 수 있다. 하지만 ISS가 많이 적용된다는 것은 인덱스 생성이 적절하지 않다는 것을 의미하기 때문에, DBA들은 인덱스 재조정이 필요하지 않은지 검토해볼 필요가 있다.

```
CREATE TABLE tbl (name STRING, gender CHAR (1), birthday DATETIME);
```

```
INSERT INTO tbl
SELECT ROWNUM, CASE (ROWNUM MOD 2) WHEN 1 THEN 'M' ELSE 'F' END,
SYSDATETIME
FROM db_class a, db_class b, db_class c, db_class d, db_class e
```



```
LIMIT 360000;
```

```
CREATE INDEX idx_tbl_gen_name ON tbl (gender, name);
-- Note that gender can only have 2 values, 'M' and 'F' (low cardinality)
```

```
UPDATE STATISTICS ON ALL CLASSES;
```

```
-- csql>;plan simple
-- this will qualify to use Index Skip Scanning
SELECT /*+ RECOMPILE INDEX_SS */ *
FROM tbl
WHERE name = '1000';
```

Query plan:

```
Index scan(tbl tbl, idx_tbl_gen_name, tbl.[name]= ?:0 (index skip scan))
```

```
-- csql>;plan simple
-- this will qualify to use Index Skip Scanning
SELECT /*+ RECOMPILE INDEX_SS */ *
FROM tbl
WHERE name between '1000' and '1050';
```

Query plan:

```
Index scan(tbl tbl, idx_tbl_gen_name, (tbl.[name]>= ?:0 and tbl.[name]<= ?:1 ) (index skip scan))
```

Loose Index Scan

GROUP BY 절 또는 **DISTINCT**의 칼럼이 인덱스 부분 키(subkey)를 포함할 때, loose index scan은 부분 키를 구성하는 칼럼 각각의 고유(unique) 값에 대해 동적으로 범위를 조정하여 B-트리 검색을 시작한다. 따라서 B-트리의 스캔 영역을 상당 부분 줄일 수 있다.

Loose index scan은 그룹핑되는 칼럼의 카디널리티가 전체 데이터량이 비해 매우 작을 때 적용하는 것이 유리하다.

질의문 힌트를 통해 **INDEX_LS**가 입력되고 다음의 경우에 loose index scan의 적용이 고려된다.

1. 인덱스가 **SELECT** 리스트의 모든 부분을 커버하는 경우. 즉, 커버링 인덱스가 적용되는 경우

2. **SELECT DISTINCT, SELECT ... GROUP BY** 또는 단일 튜플 **SELECT** 문인 경우
 3. **MIN/MAX** 함수를 제외한 모든 집계 함수가 **DISTINCT** 를 포함하는 경우
 4. **COUNT(*)** 가 사용되어선 안 됨
 5. 부분 키(subkey)의 카디널리티(고유 값의 개수)가 전체 인덱스의 카디널리티보다 100배 작은 경우
- 부분 키는 복합 인덱스(composite index)에서 앞 쪽 부분에 해당하는 것으로, 예를 들어 INDEX(a, b, c, d)로 구성되어 있는 경우 (a), (a, b) 또는 (a, b, c)가 부분 키에 해당한다.

이상과 같이 구성된 테이블에 대해 다음 질의를 수행하는 경우,

```
SELECT /*+ INDEX_LS */ a, b FROM tbl GROUP BY a;
```

칼럼 a에 대한 조건이 없으므로 부분 키를 사용할 수 없다. 그러나 다음과 같이 부분 키의 조건이 명시되면 loose index scan이 적용될 수 있다.

```
SELECT /*+ INDEX_LS */ a, b FROM tbl WHERE a > 10 GROUP BY a;
```

다음과 같이 그룹핑 칼럼이 앞에, WHERE 조건 칼럼이 뒤에 오는 경우에도 부분 키를 사용할 수 있으므로 loose index scan이 적용될 수 있다.

```
SELECT /*+ INDEX_LS */ a, b FROM tbl WHERE b > 10 GROUP BY a;
```

다음은 loose index scan이 적용되는 예이다.

```
CREATE TABLE tbl1 (
    k1 INT,
    k2 INT,
    k3 INT,
    k4 INT
);

INSERT INTO tbl1
SELECT ROWNUM MOD 2, ROWNUM MOD 400, ROWNUM MOD 80000, ROWNUM
FROM db_class a, db_class b, db_class c, db_class d, db_class e
LIMIT 360000;

CREATE INDEX idx ON tbl1 (k1, k2, k3);

CREATE TABLE tbl2 (
    k1 INT,
    k2 INT
);
```

```
INSERT INTO tbl2 VALUES (0, 0), (1, 1), (0, 2), (1, 3), (0, 4), (0, 100), (1000, 1000);
```

```
UPDATE STATISTICS ON ALL CLASSES;
```

```
-- csql>;plan simple
-- add a condition to the grouped column, k1 to enable loose index scan
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1
FROM tbl1
WHERE k1 > -1000000 LIMIT 20;
```

Query plan:

```
Sort(distinct)
  Index scan(tbl1 tbl1, idx, (tbl1.k1> ?:0 ) (covers) (loose index scan on prefix 1))
```

```
-- csql>;plan simple
-- different key ranges/filters
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1
FROM tbl1
WHERE k1 >= 0 AND k1 <= 1;
```

Query plan:

```
Sort(distinct)
  Index scan(tbl1 tbl1, idx, (tbl1.k1>= ?:0 and tbl1.k1<= ?:1 ) (covers) (loose index scan on prefix 1))
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1, k2
FROM tbl1
WHERE k1 >= 0 AND k1 <= 1 AND k2 > 3 AND k2 < 11;
```

Query plan:

```
Sort(distinct)
  Index scan(tbl1 tbl1, idx, (tbl1.k1>= ?:0 and tbl1.k1<= ?:1 ), [(tbl1.k2> ?:2 and tbl1.k2< ?:3 )] (covers) (loose index scan on prefix 2))
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1, k2
FROM tbl1
WHERE k1 >= 0 AND k1 + k2 <= 10;
```

Query plan:

```
Sort(distinct)
  Index scan(tbl1 tbl1, idx, (tbl1.k1>= ?:0 ),
[tbl1.k1+tbl1.k2<=10] (covers) (loose index scan on prefix 2))
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ tbl1.k1, tbl1.k2
FROM tbl2 INNER JOIN tbl1
ON tbl2.k1 = tbl1.k1 AND tbl2.k2 = tbl1.k2
GROUP BY tbl1.k1, tbl1.k2;
```

```
Sort(group by)
  Nested loops
    Sequential scan(tbl2 tbl2)
    Index scan(tbl1 tbl1, idx, tbl2.k1=tbl1.k1 and
tbl2.k2=tbl1.k2 (covers) (loose index scan on prefix 2))
```

```
SELECT /*+ RECOMPILE INDEX_LS */ MIN(k2), MAX(k2)
FROM tbl1;
```

Query plan:

```
Index scan(tbl1 tbl1, idx (covers) (loose index scan on prefix 2))
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ SUM(DISTINCT k1), SUM(DISTINCT k2)
FROM tbl1;
```

Query plan:

```
Index scan(tbl1 tbl1, idx (covers) (loose index scan on prefix 2))
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1
```

```
FROM tbl1
WHERE k2 > 0;
```

Query plan:

```
Sort(distinct)
  Index scan(tbl1 tbl1, idx, [(tbl1.k2> ?:0 )] (covers) (loose
index scan on prefix 2))
```

다음은 loose index scan이 적용되지 않는 경우이다.

```
-- csql>;plan simple
-- not enabled when full key is used
SELECT /*+ RECOMPILE INDEX_LS */ DISTINCT k1, k2, k3
FROM tbl1
ORDER BY 1, 2, 3 LIMIT 10;
```

Query plan:

```
Sort(distinct)
  Sequential scan(tbl1 tbl1)
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE INDEX_LS */ k1, k2, k3
FROM tbl1
WHERE k1 > -10000 GROUP BY k1, k2, k3 LIMIT 10;
```

Query plan:

```
Index scan(tbl1 tbl1, idx, (tbl1.k1> ?:0 ) (covers))
skip GROUP BY
```

```
-- csql>;plan simple
-- not enabled when using count star
SELECT /*+ RECOMPILE INDEX_LS */ COUNT(*), k1
FROM tbl1
WHERE k1 > -10000 GROUP BY k1;
```

Query plan:

```
Index scan(tbl1 tbl1, idx, (tbl1.k1> ?:0 ) (covers))
skip GROUP BY
```

```
-- csql>;plan simple
-- not enabled when index is not covering
SELECT /*+ RECOMPILE INDEX_LS */ k1, k2, SUM(k4)
FROM tbl1
WHERE k1 > -1 AND k2 > -1 GROUP BY k1, k2 LIMIT 10;
```

Query plan:

```
Index scan(tbl1 tbl1, idx, (tbl1.k1> ?:0 ), [(tbl1.k2> ?:1 )])
skip GROUP BY
```

```
-- csql>;plan simple
-- not enabled for non-distinct aggregates
SELECT /*+ RECOMPILE INDEX_LS */ k1, SUM(k2)
FROM tbl1
WHERE k1 > -1 GROUP BY k1;
```

Query plan:

```
Index scan(tbl1 tbl1, idx, (tbl1.k1> ?:0 ) (covers))
skip GROUP BY
```

```
-- csql>;plan simple
SELECT /*+ RECOMPILE */ SUM(k1), SUM(k2)
FROM tbl1;
```

Query plan:

```
Sequential scan(tbl1 tbl1)
```

인-메모리 정렬

인-메모리 정렬(in memory sort, IMS) 기능은 **ORDER BY** 절을 명시한 **LIMIT** 질의에 적용되는 최적화이다. 일반적으로, **ORDER BY** 와 **LIMIT** 절을 명시한 질의를 수행할 때, CUBRID는 전체 정렬 결과셋(full sorted resultset) 생성하고 나서 이 결과 셋에 **LIMIT** 연산을 적용한다. 전체 결과셋을 생성하는 대신 IMS 최적화를 사용하면, CUBRID는 **ORDER BY ... LIMIT** 절을 만족하는 튜플만 정렬 버퍼(sort

buffer)에 저장하는 인-메모리 바이너리 힙을 사용한다. 이 최적화는 정렬되지 않은 결과셋 전체를 정렬할 필요가 없게 하여 성능을 향상시킨다.

이 최적화의 적용 여부는 사용자가 제어하는 것이 아니며, CUBRID는 다음 상황에서 IMS를 사용할 것을 결정한다.

- **ORDER BY** 와 **LIMIT** 절을 명시한 질의
- **LIMIT** 절 적용 이후 최종 결과의 크기가 외부 정렬(external sort)에 의해 사용되는 메모리 양보다 작을 때(**메모리 관련 파라미터**의 **sort_buffer_size** 참고).

IMS는 결과 행의 개수가 아닌 결과의 실제 크기를 고려함에 주의한다. 예를 들어, 기본 정렬 버퍼 크기(2MB)에 대해, 4바이트 **INTEGER** 하나로 구성된 레코드는 524288개 행의 **LIMIT** 까지 IMS가 적용되지만 **CHAR** (1024) 하나로 구성된 레코드는 2048개 행의 **LIMIT** 까지만 IMS가 적용된다. 이 최적화는 **DISTINCT** 정렬 결과 셋을 요구하는 질의에는 적용되지 않는다.

SORT-LIMIT 최적화

SORT-LIMIT 최적화는 **ORDER BY** 절과 **LIMIT** 절을 명시한 질의에 적용되는 최적화이다. 이 최적화는 조인하는 동안의 카디널리티(cardinality)를 줄이기 위해 질의 계획에서 가능한 빨리 **LIMIT** 연산자를 평가하고자 한다.

다음 조건이 만족될 때 SORT-LIMIT 계획이 고려될 수 있다.

- **ORDER BY** 절에서 참조되는 모든 테이블은 SORT-LIMIT 계획에 속한다.
- JOIN 테이블 중 다음 테이블이 SORT-LIMIT 계획에 포함된다.
 - 외래 키/기본 키 관계에서 외래 키를 가지는 테이블
 - **LEFT JOIN** 시 왼쪽 테이블
 - **RIGHT JOIN** 시 오른쪽 테이블
- **LIMIT** 행은 **sort_limit_max_count** 시스템 파라미터(기본값: 1000) 값보다 작아야 한다.
- 질의가 **CROSS JOIN**을 하지 않는다.
- 질의가 최소한 2개의 릴레이션으로 조인한다.
- 질의는 **GROUP BY** 절을 가지지 않는다.
- 질의는 **DISTINCT** 를 명시하지 않는다.
- **ORDER BY** 표현식이 스캔하는 동안 평가될 수 있다.

예를 들어, 아래와 같은 질의는 SORT-LIMIT 최적화가 적용될 수 없는데, **SUM** 은 스캔하는 동안 평가될 수 없기 때문이다.

```
SELECT SUM(u.i) FROM u, t where u.i = t.i ORDER BY 1 LIMIT 5;
```

다음은 SORT-LIMIT을 계획하는 예이다.

```
CREATE TABLE t(i int PRIMARY KEY, j int, k int);
CREATE TABLE u(i int, j int, k int);
ALTER TABLE u ADD constraint fk_t_u_i FOREIGN KEY(i) REFERENCES t(i);
CREATE INDEX i_u_j ON u(j);

INSERT INTO t SELECT ROWNUM, ROWNUM, ROWNUM FROM _DB_CLASS a,
_DB_CLASS b LIMIT 1000;
INSERT INTO u SELECT 1+(ROWNUM % 1000), RANDOM(1000), RANDOM(1000)
FROM _DB_CLASS a, _DB_CLASS b, _DB_CLASS c LIMIT 5000;

SELECT /** RECOMPILE */ * FROM u, t WHERE u.i = t.i AND u.j > 10
ORDER BY u.j LIMIT 5;
```

위의 **SELECT** 질의 계획이 아래와 같이 출력될 때 “(sort limit)”을 확인할 수 있다.

Query plan:

```
idx-join (inner join)
  outer: temp(sort limit)
    subplan: iscan
      class: u node[0]
      index: i_u_j term[1]
      cost: 1 card 0
    cost: 1 card 0
  inner: iscan
    class: t node[1]
    index: pk_t_i term[0]
    cost: 6 card 1000
  sort: 2 asc
  cost: 7 card 0
```

재작성을 활용한 최적화

조인 변환 최적화

조인 최적화는 질의 결과에 영향을 주지 않는 테이블과의 조인을 변경하여 조인 연산을 줄이고, 질의 성능을 향상시키는 방법이다.

조인 최적화는 아래와 같은 동작이 있다:

1. **OUTER JOIN** 을 **INNER JOIN** 으로 변환
2. **INNER JOIN** 제거
3. **LEFT OUTER JOIN** 제거

조인 제거와 관련된 최적화를 하지 않으려면 **NO_ELIMINATE_JOIN** 힌트를 사용해야 한다.

OUTER JOIN을 INNER JOIN으로 변환

OUTER JOIN은 조인순서가 선행 테이블과 후행 테이블로 이미 정해져 있어 조인 순서 최적화시 순서가 제한된다. **CUBRID**는 성능을 위하여 후행 테이블에 **NULL** 불가한 조회 조건이 있는 경우 **OUTER JOIN**을 **INNER JOIN**으로 변환한다. **NULL** 가능한 조회 조건은 이 변환에서 제외되며, 아래 조건을 만족해야 한다.

1. **ON**절에 조건절이 작성된 경우
2. **NULL** 변환 함수 (**COALESCE ()**, **NVL ()**, **NVL2 ()**, **DECODE ()**, **IF ()**, **IFNULL ()**, **CONCAT_WS ()**)
3. **IS NULL**, **CASE** 문

다음은 **OUTER JOIN**을 **INNER JOIN**으로 변환하는 예제이다.

```
--create table
CREATE TABLE t1 (col1 int, col2 int);
INSERT INTO t1 values (1,1),(2,2),(3,2);

-- csql> ;plan simple
SELECT /** recompile */ *
FROM t1 a LEFT OUTER JOIN t1 b on a.col1 = b.col2
WHERE b.col1 = 2;
```

```
Query Plan:
  NESTED LOOPS (inner join)
    TABLE SCAN (a)
    TABLE SCAN (b)
```

다음은 **NULL** 가능한 조회조건 때문에 **INNER JOIN**으로 변환되지 않는 예제이다.

```
-- csql> ;plan simple
SELECT /** recompile */ *
```

```
FROM t1 a LEFT OUTER JOIN t1 b on a.col1 = b.col2
WHERE nvl(b.col1,2) = 2;
```

```
Query Plan:
  NESTED LOOPS (left outer join)
    TABLE SCAN (a)
    TABLE SCAN (b)
```

INNER JOIN 제거

기본키(**PK**)가 있는 테이블과 그 기본키(**PK**)를 참조하는 외래키(**FK**)가 있는 테이블 간의 관계를 부모-자식 관계라고 한다. 부모-자식 관계의 **INNER JOIN**에서 조인 조건 외에 부모 테이블에 대한 참조가 없는 경우에는 부모 테이블과의 조인을 제거해도 쿼리 결과에 영향을 주지 않는다. 이러한 경우에는 부모 테이블과의 조인을 제거하여 조인 연산을 줄이고, 쿼리 성능을 향상시킨다.

INNER JOIN을 제거하기 위해서는 다음 조건을 만족해야 한다:

1. 부모-자식 관계의 **INNER JOIN**을 한다.
2. 기본키(**PK**)와 외래키(**FK**)의 모든 컬럼은 조인 조건에 사용되어야 한다.
3. 기본키(**PK**) 컬럼은 해당 기본키(**PK**) 컬럼을 참조하는 외래키(**FK**) 컬럼과 동일한 조인 조건에 사용되어야 한다.
4. 모든 조인 조건은 동등(=) 비교를 사용해야 한다.
5. 조인 조건 외에 부모 테이블에 대한 참조가 없어야 한다.
6. 상속 관계가 있는 테이블을 조회하는 경우에는 **ALL** 키워드를 사용하지 않아야 한다.

부모 테이블과의 조인을 제거할 때 자식 테이블의 외래키(**FK**) 컬럼에 **NOT NULL** 제약 조건이 없는 경우에는 해당 컬럼에 대한 **IS NOT NULL** 조건이 추가된다.

다음은 **INNER JOIN**을 제거하는 예제이다.

```
call login ('public') on class db_user;

/* current_user: public */
drop table if exists child_tbl, parent_tbl;

create table parent_tbl (
  id int,
  sub_id int,
  name varchar (100),
  filter int,
  primary key (id, sub_id)
```

```

);

insert into parent_tbl
select
    ((rownum - 1) / 10) + 1 as id,
    (((rownum - 1) % 10) + 1) * 10 as sub_id,
    'Parent-' || lpad (rownum, 3) as name,
    ((rownum - 1) % 10) + 1 as filter
from db_root
connect by level <= 100;

create table child_tbl (
    id int,
    name varchar (100),
    filter int,
    parent_id int not null,
    parent_sub_id int,
    primary key (id),
    foreign key (parent_id, parent_sub_id) references parent_tbl
(id, sub_id)
);

insert into child_tbl
select
    rownum as id,
    'Child-' || lpad (rownum, 5) as name,
    ((rownum - 1) % 100) + 1 as filter,
    id as parent_id,
    sub_id as parent_sub_id
from parent_tbl, (select level from db_root connect by level <= 100);

update statistics on parent_tbl, child_tbl with fullscan;

```

```

/* current_user: public */
set optimization level 513;

select /*+ recompile */
    c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
    parent_tbl p
    inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
    c.filter = 1 and c.parent_id = 1
order by
    c.name;

```

parent_tbl 테이블과의 조인이 제거되었고, *child_tbl* 테이블의 *parent_sub_id* 컬럼에 대한 **IS NOT NULL** 조건이 추가되었다.

Query plan:

```
temp(order by)
  subplan: iscan
            class: c node[0]
            index: fk_child_tbl_parent_id_parent_sub_id term[1]
            filtr: term[2]
            sargs: term[0]
            sort: 5 asc
            cost: 8 card 1
    sort: 2 asc
    cost: 14 card 1
```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
child_tbl c where c.parent_id=?:0 and c.filter=?:1 and
c.parent_sub_id is not null order by 2
```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	1	'Child-	1	
1	10			
	101	'Child-	1	
1	20			
	201	'Child-	1	
1	30			
	301	'Child-	1	
1	40			
	401	'Child-	1	
1	50			
	501	'Child-	1	
1	60			
	601	'Child-	1	
1	70			
	701	'Child-	1	
1	80			
	801	'Child-	1	
1	90			
	901	'Child-	1	
1	100			

다음은 **NO_ELIMINATE_JOIN** 힌트를 사용하는 예제이다.

```
/* current_user: public */
set optimization level 513;
```

```

select /*+ recompile no_eliminate_join */
       c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
       parent_tbl p
       inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
       c.filter = 1 and c.parent_id = 1
order by
       c.name;

```

NO_ELIMINATE_JOIN 힌트를 사용했기 때문에 *parent_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```

temp(order by)
  subplan: idx-join (inner join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[4]
(covers)
      cost:  1 card 10
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[1] AND term[3]
      sargs: term[2]
      cost:  3 card 1
    cost:  4 card 1
  sort:  2 asc
  cost: 10 card 1

```

Query stmt:

```

select /*+ NO_ELIMINATE_JOIN */ c.id, c.[name], c.filter,
c.parent_id, c.parent_sub_id from parent_tbl p, child_tbl c where
p.sub_id=c.parent_sub_id and p.id=?:0  and c.parent_id=?:1  and
c.filter=?:2  and p.id=c.parent_id order by 2

```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	1	'Child-	1'	1
1	10			
	101	'Child-	101'	1
1	20			
	201	'Child-	201'	1

```

1          30
          301 'Child- 301'          1
1          40
          401 'Child- 401'          1
1          50
          501 'Child- 501'          1
1          60
          601 'Child- 601'          1
1          70
          701 'Child- 701'          1
1          80
          801 'Child- 801'          1
1          90
          901 'Child- 901'          1
1         100

```

기본키(PK)와 외래키(FK)의 모든 컬럼은 조인 조건에 사용되어야 한다.

다음은 *parent_tbl* 테이블의 *id* 컬럼과 *child_tbl* 테이블의 *parent_id* 컬럼만 조인 조건으로 사용된 예제이다.

```

/* current_user: public */
set optimization level 513;

select /** recompile */
       c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from   parent_tbl p
       inner join child_tbl c on p.id = c.parent_id
where  c.filter = 2 and c.parent_id = 2 and c.parent_sub_id = 20
order by
       c.name;

```

parent_tbl 테이블의 *sub_id* 컬럼과 *child_tbl* 테이블의 *parent_sub_id* 컬럼이 조인 조건으로 사용되지 않았기 때문에 *parent_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```

temp(order by)
  subplan: nl-join (cross join)
    outer: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[2] AND term[3]
  sargs: term[1]
  cost:  3 card 1

```

```

            inner: iscan
                    class: p node[0]
                    index: pk_parent_tbl_id_sub_id term[4]
(covers)
            cost: 1 card 10
            cost: 4 card 10
            sort: 2 asc
            cost: 10 card 10

```

Query stmt:

```

select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
parent_tbl p, child_tbl c where p.id=?:0 and c.parent_sub_id=?:1
and c.parent_id=?:2 and c.filter=?:3 and p.id=c.parent_id order
by 2

```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			
	1102	'Child- 1102'	2	
2	20			

기본키(PK) 컬럼은 해당 기본키(PK)를 참조하는 외래키(FK) 컬럼과 동일한 조인 조건에 사용되어야 한다.

다음은 조인 조건에서 *id* 컬럼을 *parent_sub_id* 컬럼과 비교하고, *sub_id* 컬럼을 *parent_id* 컬럼과 비교하는 예제이다.

```

/* current_user: public */
set optimization level 513;

select /*+ recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
      parent_tbl p
      inner join child_tbl c on p.id = c.parent_sub_id and p.sub_id =
c.parent_id
where
      c.filter = 3 and c.parent_id = 3
order by
      c.name;

```

id 컬럼이 *parent_id* 컬럼과 비교되지 않았고, *sub_id* 컬럼이 *parent_sub_id* 컬럼과 비교되지 않았기 때문에 *parent* 테이블과의 조인이 제거되지 않았다.

Query plan:

```

temp(order by)
  subplan: idx-join (inner join)
    outer: sscan
      class: p node[0]
      sargs: term[3]
      cost: 1 card 1
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[0] AND term[4]
      sargs: term[2]
      cost: 3 card 1
    cost: 5 card 1
  sort: 2 asc
  cost: 11 card 1

```

Query stmt:

```

select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
parent_tbl p, child_tbl c where p.id=c.parent_sub_id and p.sub_id= ?
:0 and c.parent_id= ? :1 and c.filter= ? :2 and p.sub_id=c.parent_id
order by 2

```

There are no results.
0 row selected.

모든 조인 조건은 동등(=) 비교를 사용해야 한다.

다음은 조인 조건으로 동등(=) 비교를 사용하지 않는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /*+ recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
      parent_tbl p
      inner join child_tbl c on p.id = c.parent_id and p.sub_id <
c.parent_sub_id
where
      c.filter = 4 and c.parent_id = 4 and c.parent_sub_id = 40
order by
      c.name;
```

조인 조건에서 *parent_tbl* 테이블의 *sub_id* 컬럼과 *child_tbl* 테이블의 *parent_sub_id* 컬럼을 비교할 때 동등(=) 비교를 사용하지 않았기 때문에 *parent* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: nl-join (cross join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[5]
AND term[6] (covers)
      cost: 1 card 1
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[3] AND term[4]
      sargs: term[2]
      cost: 3 card 1
    cost: 4 card 1
  sort: 2 asc
  cost: 10 card 1
```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
parent_tbl p, child_tbl c where (p.sub_id< ?:0 ) and p.id= ?:1 and
c.parent_sub_id= ?:2 and c.parent_id= ?:3 and c.filter= ?:4 and
p.id=c.parent_id and (p.sub_id<c.parent_sub_id) order by 2
```

id	name	filter	parent_id
parent_sub_id			

```
=====
=====
      3304  'Child- 3304'                4
4          40
      3304  'Child- 3304'                4
4          40
      3304  'Child- 3304'                4
4          40
```

조인 조건 외에 부모 테이블에 대한 참조가 없어야 한다.

다음은 *parent_tbl* 테이블의 *name* 컬럼을 조회하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /*+ recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id, p.name
from
      parent_tbl p
      inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
      c.filter = 5 and c.parent_id = 5
order by
      c.name;
```

parent_tbl 테이블의 *name* 컬럼의 값을 조회하고 있기 때문에 *parent_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (inner join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[4]
      cost:  1 card 10
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[1] AND term[3]
      sargs: term[2]
      cost:  3 card 1
    cost:  4 card 1
  sort:  2 asc
  cost: 10 card 1
```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id, p.
[name] from parent_tbl p, child_tbl c where p.sub_id=c.parent_sub_id
and p.id=?:0 and c.parent_id=?:1 and c.filter=?:2 and
p.id=c.parent_id order by 2
```

	id	name	filter	parent_id
parent_sub_id		name		
	4005	'Child- 4005'	5	
5	10	'Parent- 41'		
	4105	'Child- 4105'	5	
5	20	'Parent- 42'		
	4205	'Child- 4205'	5	
5	30	'Parent- 43'		
	4305	'Child- 4305'	5	
5	40	'Parent- 44'		
	4405	'Child- 4405'	5	
5	50	'Parent- 45'		
	4505	'Child- 4505'	5	
5	60	'Parent- 46'		
	4605	'Child- 4605'	5	
5	70	'Parent- 47'		
	4705	'Child- 4705'	5	
5	80	'Parent- 48'		
	4805	'Child- 4805'	5	
5	90	'Parent- 49'		
	4905	'Child- 4905'	5	
5	100	'Parent- 50'		

다음은 *parent* 테이블의 *filter* 컬럼에 대한 조건이 있는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
      parent_tbl p
      inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
      c.filter = 5 and c.parent_id = 5 and p.filter = 5
order by
      c.name;
```

parent_tbl 테이블의 *filter* 컬럼의 값이 5인 레코드를 조회하고 있기 때문에 *parent_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (inner join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[5]
      sargs: term[3]
      cost: 1 card 1
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[1] AND term[4]
      sargs: term[2]
      cost: 3 card 1
    cost: 4 card 1
  sort: 2 asc
  cost: 10 card 1
```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
parent_tbl p, child_tbl c where p.sub_id=c.parent_sub_id and p.id= ? :
0 and p.filter= ? :1 and c.parent_id= ? :2 and c.filter= ? :3 and
p.id=c.parent_id order by 2
```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	4405	'Child- 4405'	5	
5		50		

다음은 *parent_tbl* 테이블의 *name* 컬럼을 기준으로 결과를 정렬하는 예제이다.

```
select /*+ recompile */
  c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
  parent_tbl p
  inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
  c.filter = 5 and c.parent_id = 5
```

```
order by
  p.name desc;
```

parent_tbl 테이블의 *nmae* 컬럼을 기준으로 결과를 내림차순 정렬하고 있기 때문에 *parent_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (inner join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[4]
      cost: 1 card 10
    inner: iscan
      class: c node[1]
      index:
fk_child_tbl_parent_id_parent_sub_id term[1] AND term[3]
      sargs: term[2]
      cost: 3 card 1
    cost: 4 card 1
  sort: 6 desc
  cost: 10 card 1
```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id, p.
[name] from parent_tbl p, child_tbl c where p.sub_id=c.parent_sub_id
and p.id=?:0 and c.parent_id=?:1 and c.filter=?:2 and
p.id=c.parent_id order by 6 desc
```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	4905	'Child- 4905'	5	
5	100			
	4805	'Child- 4805'	5	
5	90			
	4705	'Child- 4705'	5	
5	80			
	4605	'Child- 4605'	5	
5	70			
	4505	'Child- 4505'	5	
5	60			
	4405	'Child- 4405'	5	
5	50			
	4305	'Child- 4305'	5	
5	40			

```

5      4205  'Child- 4205'      5
      30
5      4105  'Child- 4105'      5
      20
5      4005  'Child- 4005'      5
      10

```

묵시적인 동등(=)

WHERE절에서 조인 조건 외에 부모 테이블에 대한 조건이 있더라도 조인 조건에 사용된 컬럼이 해당 조건에 사용된 경우에는 묵시적인 동등(=)에 의해 부모 테이블과의 조인을 제거할 수 있다. 해당 조건은 동등(=) 비교를 사용해야 하고, 상수와 비교되어야 한다.

다음은 *parent_tbl* 테이블의 *sub_id* 컬럼에 대한 동등(=) 비교 조건이 있는 예제이다.

```

/* current_user: public */
set optimization level 513;

select /** recompile */
       c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from   parent_tbl p
       inner join child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where  c.filter = 6 and c.parent_id = 6 and p.sub_id = 60
order by
       c.name;

```

parent_tbl 테이블의 *sub_id* 컬럼의 값이 60인 레코드를 조회하고 있지만 조인 조건에 의해 해당 조건은 *child_tbl* 테이블의 *parent_sub_id* 컬럼의 값이 60인 조건과 동등하기 때문에 *parent_tbl* 테이블과의 조인이 제거되었다.

Query plan:

```

temp(order by)
  subplan: iscan
            class: c node[0]
            index: fk_child_tbl_parent_id_parent_sub_id term[1]
AND term[2]
            filtr: term[3]
            sargs: term[0]
            cost:  3 card 1
      sort:  2 asc
      cost:  9 card 1

```

Query stmt:

```
select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
child_tbl c where c.parent_sub_id=?:0 and c.parent_id=?:1 and
c.filter=?:2 and c.parent_sub_id is not null order by 2
```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	5506	'Child- 5506'	6	
6		60		

상속 관계가 있는 테이블을 조회하는 경우에는 **ALL** 키워드를 사용하지 않아야 한다. 상속 관계가 있는 부모 또는 자식 테이블에 **ALL** 키워드를 사용하면 부모 테이블과의 조인을 제거할 수 없다.

다음은 *child_tbl* 테이블에 **ALL** 키워드를 사용하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
      parent_tbl p
      inner join all child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
      c.filter = 7 and c.parent_id = 7
order by
      c.name;
```

parent_tbl 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (inner join)
    outer: iscan
      class: p node[0]
      index: pk_parent_tbl_id_sub_id term[4]
(covers)
      cost: 1 card 10
    inner: iscan
      class: c node[1]
```

```

                                index:
fk_child_tbl_parent_id_parent_sub_id term[1] AND term[3]
                                sargs: term[2]
                                cost:  3 card 1
                                cost:  4 card 1
                                sort:  2 asc
                                cost: 10 card 1

```

Query stmt:

```

select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
parent_tbl p,  all child_tbl c where p.sub_id=c.parent_sub_id and
p.id=?:0 and c.parent_id=?:1 and c.filter=?:2 and
p.id=c.parent_id order by 2

```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	6007	'Child- 6007'	7	
7	10			
	6107	'Child- 6107'	7	
7	20			
	6207	'Child- 6207'	7	
7	30			
	6307	'Child- 6307'	7	
7	40			
	6407	'Child- 6407'	7	
7	50			
	6507	'Child- 6507'	7	
7	60			
	6607	'Child- 6607'	7	
7	70			
	6707	'Child- 6707'	7	
7	80			
	6807	'Child- 6807'	7	
7	90			
	6907	'Child- 6907'	7	
7	100			

다음은 *parent_tbl* 테이블과 *child_tbl* 테이블에 **ONLY** 키워드를 사용하는 예제이다.

```

/* current_user: public */
set optimization level 513;

select /** recompile */
      c.id, c.name, c.filter, c.parent_id, c.parent_sub_id
from
      only parent_tbl p

```



```

    inner join only child_tbl c on p.id = c.parent_id and p.sub_id =
c.parent_sub_id
where
    c.filter = 7 and c.parent_id = 7
order by
    c.name;

```

parent_tbl 테이블과의 조인이 제거되었다.

Query plan:

```

temp(order by)
  subplan: iscan
            class: c node[0]
            index: fk_child_tbl_parent_id_parent_sub_id term[1]
            filtr: term[2]
            sargs: term[0]
            sort: 5 asc
            cost: 8 card 1
    sort: 2 asc
    cost: 14 card 1

```

Query stmt:

```

select c.id, c.[name], c.filter, c.parent_id, c.parent_sub_id from
child_tbl c where c.parent_id= ?:0 and c.filter= ?:1 and
c.parent_sub_id is not null order by 2

```

	id	name	filter	parent_id
parent_sub_id				
=====				
=====				
	6007	'Child- 6007'	7	
7	10			
	6107	'Child- 6107'	7	
7	20			
	6207	'Child- 6207'	7	
7	30			
	6307	'Child- 6307'	7	
7	40			
	6407	'Child- 6407'	7	
7	50			
	6507	'Child- 6507'	7	
7	60			
	6607	'Child- 6607'	7	
7	70			
	6707	'Child- 6707'	7	
7	80			
	6807	'Child- 6807'	7	

```

7          90
        6907 'Child- 6907'          7
7          100

```

LEFT OUTER JOIN 제거

N:1 관계의 **LEFT OUTER JOIN**에서 조인 조건 외에 오른쪽 테이블에 대한 참조가 없는 경우에는 오른쪽 테이블과의 조인을 제거해도 쿼리 결과에 영향을 주지 않는다. 이러한 경우에는 오른쪽 테이블과의 조인을 제거하여 조인 연산을 줄이고, 쿼리 성능을 향상시킨다.

LEFT OUTER JOIN 제거를 하기 위해서는 다음 조건을 만족해야 한다:

1. **LEFT OUTER JOIN**을 한다.
2. 오른쪽 테이블은 기본키(**PK**) 또는 **UNIQUE** 제약조건이 있어야 한다.
3. 기본키(**PK**) 또는 **UNIQUE** 제약조건의 모든 컬럼은 조인 조건에 사용되어야 한다.
4. 모든 조인 조건은 동등(=) 비교를 사용해야 한다.
5. 조인 조건 외에 오른쪽 테이블에 대한 참조가 없어야 한다.
6. 상속 관계가 있는 테이블을 조회하는 경우에는 **ALL** 키워드를 사용하지 않아야 한다.

다음은 **LEFT OUTER JOIN** 제거를 하는 예제이다.

```

call login ('public') on class db_user;

/* current_user: public */
drop table if exists left_tbl, right_tbl;

create table left_tbl (
  id int,
  sub_id int,
  other_id int,
  name varchar (100),
  filter int,
  primary key (id, sub_id, other_id)
);

insert into left_tbl
select
  ((rownum - 1) / 100) + 1 as id,
  (((rownum - 1) % 10) + 1) * 10 as sub_id,
  (((rownum - 1) % 100) + 1) * 100 as other_id,
  'Left-' || lpad (rownum, 4) as name,
  ((rownum - 1) % 100) + 1 as filter
from db_root

```

```

connect by level <= 1000;

create table right_tbl (
    id int,
    sub_id int,
    name varchar (100),
    filter int,
    primary key (id, sub_id)
);

insert into right_tbl
select
    id as id,
    sub_id as sub_id,
    'Right-' || lpad (rownum, 3) as name,
    ((rownum - 1) % 10) + 1 as filter
from (select distinct id, sub_id from left_tbl);

update statistics on left_tbl, right_tbl with fullscan;

```

```

/* current_user: public */
set optimization level 513;

select /** recompile */
    l.id, l.sub_id, l.other_id, l.name, l.filter
from
    left_tbl as l
    left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
    l.filter = 1
order by
    l.name;

```

right_tbl 테이블과의 조인이 제거되었다.

Query plan:

```

temp(order by)
  subplan: sscan
            class: l node[0]
            sargs: term[0]
            cost:  6 card 1
    sort:    4 asc
    cost:    13 card 1

```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l where l.filter= ?:0 order by 4
```

filter	id	sub_id	other_id	name
=====				
=====				
1'	1	10	100	'Left-
101'	2	10	100	'Left-
201'	3	10	100	'Left-
301'	4	10	100	'Left-
401'	5	10	100	'Left-
501'	6	10	100	'Left-
601'	7	10	100	'Left-
701'	8	10	100	'Left-
801'	9	10	100	'Left-
901'	10	10	100	'Left-

다음은 **NO_ELIMINATE_JOIN** 힌트를 사용하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /*+ recompile no_eliminate_join */
l.id, l.sub_id, l.other_id, l.name, l.filter
from
left_tbl as l
left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
l.filter = 1
order by
l.name;
```

NO_ELIMINATE_JOIN 힌트를 사용했기 때문에 *right_tbl* 테이블과의 조인을 제거되지 않았다.

Query plan:

```

temp(order by)
  subplan: idx-join (left outer join)
    outer: sscan
      class: l node[0]
      sargs: term[2]
      cost: 6 card 1
    inner: iscan
      class: r node[1]
      index: pk_right_tbl_id_sub_id term[0]
AND term[1] (covers)
      cost: 1 card 100
      cost: 9 card 1
  sort: 4 asc
  cost: 15 card 1

```

Query stmt:

```

select /*+ NO_ELIMINATE_JOIN */ l.id, l.sub_id, l.other_id, l.
[name], l.filter from left_tbl l left outer join right_tbl r on
l.sub_id=r.sub_id and l.id=r.id where l.filter=?:0 order by 4

```

filter	id	sub_id	other_id	name
=====				
=====				
1'	1	10	100	'Left-
101'	2	10	100	'Left-
201'	3	10	100	'Left-
301'	4	10	100	'Left-
401'	5	10	100	'Left-
501'	6	10	100	'Left-
601'	7	10	100	'Left-
701'	8	10	100	'Left-
801'	9	10	100	'Left-
901'	10	10	100	'Left-

오른쪽 테이블은 기본키(**PK**) 또는 **UNIQUE** 제약조건이 있어야 한다.

오른쪽 테이블에 기본키(**PK**) 또는 **UNIQUE** 제약조건이 없으면 N:1 관계의 **LEFT OUTER JOIN**이 아니기 때문에 오른쪽 테이블과의 조인을 제거할 수 없다.

다음은 *right_tbl* 테이블의 기본키(**PK**)를 제거하고 *id* 컬럼과 *sub_id* 컬럼으로 구성된 일반 인덱스를 생성해놓은 예제이다.

```
/* current_user: public */
set optimization level 513;

alter table right_tbl drop primary key;
alter table right_tbl add index i_right_id_sub_id (id, sub_id);
update statistics on right_tbl with fullscan;

select /** recompile */
      l.id, l.sub_id, l.other_id, l.name, l.filter
from
      left_tbl as l
    left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
      l.filter = 2
order by
      l.name;

drop index i_right_id_sub_id on right_tbl;
alter table right_tbl add primary key (id, sub_id);
update statistics on right_tbl with fullscan;
```

right_tbl 테이블에 기본키(**PK**) 또는 **UNIQUE** 제약조건이 없기 때문에 *right_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (left outer join)
    outer: sscan
      class: l node[0]
      sargs: term[2]
      cost: 6 card 1
    inner: iscan
      class: r node[1]
      index: i_right_id_sub_id term[0] AND
term[1] (covers)
      cost: 1 card 100
    cost: 9 card 1
  sort: 4 asc
  cost: 15 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l left outer join right_tbl r on l.sub_id=r.sub_id and l.id=r.id
where l.filter=?:0 order by 4
```

filter	id	sub_id	other_id	name
=====				
=====				
	1	20	200	'Left-
2'		2		
	2	20	200	'Left-
102'		2		
	3	20	200	'Left-
202'		2		
	4	20	200	'Left-
302'		2		
	5	20	200	'Left-
402'		2		
	6	20	200	'Left-
502'		2		
	7	20	200	'Left-
602'		2		
	8	20	200	'Left-
702'		2		
	9	20	200	'Left-
802'		2		
	10	20	200	'Left-
902'		2		

다음은 *right_tbl* 테이블의 기본키(PK)를 제거하고 *id* 컬럼과 *sub_id* 컬럼으로 구성된 **UNIQUE** 제약 조건을 생성해놓은 예제이다.

```
/* current_user: public */
set optimization level 513;

alter table right_tbl drop primary key;
alter table right_tbl add constraint unique (id, sub_id);
update statistics on right_tbl with fullscan;

select /** recompile */
l.id, l.sub_id, l.other_id, l.name, l.filter
from
left_tbl as l
left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
l.filter = 2
```

```

order by
    l.name;

alter table right_tbl drop constraint u_right_tbl_id_sub_id;
alter table right_tbl add primary key (id, sub_id);
update statistics on right_tbl with fullscan;

```

right_tbl 테이블에 기본키(PK) 또는 **UNIQUE** 제약조건이 없기 때문에 *right_tbl* 테이블과의 조인이 제거되었다.

Query plan:

```

temp(order by)
  subplan: sscan
            class: l node[0]
            sargs: term[0]
            cost:  6 card 1
    sort:    4 asc
    cost:    13 card 1

```

Query stmt:

```

select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l where l.filter= ? :0 order by 4

```

filter	id	sub_id	other_id	name
=====				
=====				
	1	20	200	'Left-
2'		2		
	2	20	200	'Left-
102'		2		
	3	20	200	'Left-
202'		2		
	4	20	200	'Left-
302'		2		
	5	20	200	'Left-
402'		2		
	6	20	200	'Left-
502'		2		
	7	20	200	'Left-
602'		2		
	8	20	200	'Left-
702'		2		
	9	20	200	'Left-
802'		2		

902'	10	20	200	'Left-
		2		

기본키(PK) 또는 **UNIQUE** 제약조건의 모든 컬럼을 조인 조건에 사용해야 한다.
 다음은 *left_tbl* 테이블의 *id* 컬럼과 *right_tbl* 테이블의 *id* 컬럼만 조인 조건으로 사용된 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
  l.id, l.sub_id, l.other_id, l.name, l.filter
from
  left_tbl as l
  left outer join right_tbl as r on l.id = r.id
where
  l.filter = 3 and l.id = 3
order by
  l.name;
```

left_tbl 테이블의 *sub_id* 컬럼과 *right_tbl* 테이블의 *sub_id* 컬럼이 조인 조건으로 사용되지 않았기 때문에 *right_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (left outer join)
    outer: iscan
      class: l node[0]
      index: pk_left_tbl_id_sub_id_other_id
term[2]
      sargs: term[1]
      cost: 3 card 1
    inner: iscan
      class: r node[1]
      index: pk_right_tbl_id_sub_id term[0]
(covers)
      filtr: term[3]
      cost: 1 card 10
    sort: 2 asc, 3 asc
    cost: 4 card 1
  sort: 4 asc
  cost: 10 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
```

```
l left outer join right_tbl r on l.id=r.id and r.id=?:0 where
l.id=?:1 and l.filter=?:2 order by 4
```

filter	id	sub_id	other_id	name
=====				
=====				
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		
203'	3	30	300	'Left-
		3		

모든 조인 조건은 동등(=) 비교를 사용해야 한다.

다음은 조인 조건으로 동등(=) 비교를 사용하지 않는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
  l.id, l.sub_id, l.other_id, l.name, l.filter
from
  left_tbl as l
  left outer join right_tbl as r on l.id = r.id and l.sub_id <
r.sub_id
where
  l.filter = 4 and l.id = 4
order by
  l.name;
```


조인 조건 외에 오른쪽 테이블에 대한 참조가 없어야 한다.

다음은 *right_tbl* 테이블의 *name* 컬럼을 조회하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /*+ recompile */
    l.id, l.sub_id, l.other_id, l.name, l.filter, r.name
from
    left_tbl as l
    left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
    l.filter = 5
order by
    l.name;
```

right_tbl 테이블의 *name* 컬럼의 값을 조회하고 있기 때문에 *right_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (left outer join)
    outer: sscan
      class: l node[0]
      sargs: term[2]
      cost: 6 card 1
    inner: iscan
      class: r node[1]
      index: pk_right_tbl_id_sub_id term[0]
AND term[1]
      cost: 1 card 100
    cost: 9 card 1
  sort: 4 asc
  cost: 15 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter, r.[name] from
left_tbl l left outer join right_tbl r on l.sub_id=r.sub_id and
l.id=r.id where l.filter=?:0 order by 4
```

filter	id	sub_id	other_id	name
=====				
=====				
	1	50	500	'Left-

5'		5	'Right- 5'	
	2	50	500	'Left-
105'		5	'Right- 15'	
	3	50	500	'Left-
205'		5	'Right- 25'	
	4	50	500	'Left-
305'		5	'Right- 35'	
	5	50	500	'Left-
405'		5	'Right- 45'	
	6	50	500	'Left-
505'		5	'Right- 55'	
	7	50	500	'Left-
605'		5	'Right- 65'	
	8	50	500	'Left-
705'		5	'Right- 75'	
	9	50	500	'Left-
805'		5	'Right- 85'	
	10	50	500	'Left-
905'		5	'Right- 95'	

다음은 **LEFT OUTER JOIN**을 하기 전에 *right_tbl* 테이블의 *filter* 컬럼에 대한 조건이 있는 예제이다.

```

/* current_user: public */
set optimization level 513;

select /*+ recompile */
    l.id, l.sub_id, l.other_id, l.name, l.filter
from
    left_tbl as l
    left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id and r.filter = 5
where
    l.filter = 5
order by
    l.name;

```

right_tbl 테이블의 *filter* 컬럼의 값이 5인 레코드를 조회하고 있지만 **LEFT OUTER JOIN**을 하기 전이기 때문에 *right_tbl* 테이블과의 조인이 제거되었다.

Query plan:

```

temp(order by)
  subplan: sscan
            class: l node[0]
            sargs: term[0]
            cost: 6 card 1
  sort: 4 asc
  cost: 13 card 1

```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l where l.filter=?:0 order by 4
```

filter	id	sub_id	other_id	name
5'	1	50	500	'Left-
105'	2	50	500	'Left-
205'	3	50	500	'Left-
305'	4	50	500	'Left-
405'	5	50	500	'Left-
505'	6	50	500	'Left-
605'	7	50	500	'Left-
705'	8	50	500	'Left-
805'	9	50	500	'Left-
905'	10	50	500	'Left-

다음은 *right_tbl* 테이블의 *name* 컬럼으로 결과를 정렬하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /*+ recompile */
  l.id, l.sub_id, l.other_id, l.name, l.filter, r.name
from
  left_tbl as l
  left outer join right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
  l.filter = 5
order by
  r.name desc;
```

right_tbl 테이블의 *name* 컬럼을 기준으로 결과를 내림차순 정렬하고 있기 때문에 *right_tbl* 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (left outer join)
    outer: sscan
      class: l node[0]
      sargs: term[2]
      cost: 6 card 1
    inner: iscan
      class: r node[1]
      index: pk_right_tbl_id_sub_id term[0]
AND term[1]
      cost: 1 card 100
      cost: 9 card 1
  sort: 6 desc
  cost: 15 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter, r.[name] from
left_tbl l left outer join right_tbl r on l.sub_id=r.sub_id and
l.id=r.id where l.filter= ?:0 order by 6 desc
```

filter	id	sub_id	other_id	name
905'	10	50	500	'Left-
		5	'Right-	95'
805'	9	50	500	'Left-
		5	'Right-	85'
705'	8	50	500	'Left-
		5	'Right-	75'
605'	7	50	500	'Left-
		5	'Right-	65'
505'	6	50	500	'Left-
		5	'Right-	55'
405'	5	50	500	'Left-
		5	'Right-	45'
305'	4	50	500	'Left-
		5	'Right-	35'
205'	3	50	500	'Left-
		5	'Right-	25'
105'	2	50	500	'Left-
		5	'Right-	15'
5'	1	50	500	'Left-
		5	'Right-	5'

상속 관계가 있는 테이블을 조회하는 경우에는 **ALL** 키워드를 사용하지 않아야 한다. 상속 관계가 있는 오른쪽 테이블에 **ALL** 키워드를 사용하면 부모 테이블과의 조인을 제거할 수 없다.

다음은 *right_tbl* 테이블에 **ALL** 키워드를 사용하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
    l.id, l.sub_id, l.other_id, l.name, l.filter
from
    left_tbl as l
    left outer join all right_tbl as r on l.id = r.id and l.sub_id =
r.sub_id
where
    l.filter = 6
order by
    l.name;
```

right_tbl 테이블과의 조인이 제거되지 않았다.

Query plan:

```
temp(order by)
  subplan: idx-join (left outer join)
    outer: sscan
      class: l node[0]
      sargs: term[2]
      cost: 6 card 1
    inner: iscan
      class: r node[1]
      index: pk_right_tbl_id_sub_id term[0]
  AND term[1] (covers)
    cost: 1 card 100
  cost: 9 card 1
  sort: 4 asc
  cost: 15 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l left outer join all right_tbl r on l.sub_id=r.sub_id and
l.id=r.id where l.filter=?:0 order by 4
```

```
          id          sub_id      other_id  name
filter
=====
=====
```


6'	1	60	600	'Left-
		6		
	2	60	600	'Left-
106'		6		
	3	60	600	'Left-
206'		6		
	4	60	600	'Left-
306'		6		
	5	60	600	'Left-
406'		6		
	6	60	600	'Left-
506'		6		
	7	60	600	'Left-
606'		6		
	8	60	600	'Left-
706'		6		
	9	60	600	'Left-
806'		6		
	10	60	600	'Left-
906'		6		

다음은 *left_tbl* 테이블과 *right_tbl* 테이블에 **ONLY** 키워드를 사용하는 예제이다.

```
/* current_user: public */
set optimization level 513;

select /** recompile */
  l.id, l.sub_id, l.other_id, l.name, l.filter
from
  only left_tbl as l
 left outer join only right_tbl as r on l.id = r.id and l.sub_id
= r.sub_id
where
  l.filter = 6
order by
  l.name;
```

right 테이블과의 조인이 제거되었다.

Query plan:

```
temp(order by)
  subplan: sscan
            class: l node[0]
            sargs: term[0]
            cost: 6 card 1
  sort: 4 asc
  cost: 13 card 1
```

Query stmt:

```
select l.id, l.sub_id, l.other_id, l.[name], l.filter from left_tbl
l where l.filter= ?:0 order by 4
```

filter	id	sub_id	other_id	name
=====				
=====				
6'	1	60	600	'Left-
106'	2	60	600	'Left-
206'	3	60	600	'Left-
306'	4	60	600	'Left-
406'	5	60	600	'Left-
506'	6	60	600	'Left-
606'	7	60	600	'Left-
706'	8	60	600	'Left-
806'	9	60	600	'Left-
906'	10	60	600	'Left-

View Merging 최적화

View Merging은 질의 처리 시간과 오버헤드를 줄이는데 초점을 둔다. 질의에서 뷰(인라인 뷰 포함)를 사용할 때, 시스템은 일반적으로 새로운 임시 테이블을 생성한다. 이렇게 생성된 임시 테이블은 인덱스를 사용할 수 없을 뿐만 아니라, 임시 테이블을 생성하는 과정 자체가 시스템에 불필요한 부하를 발생하게 된다. 따라서 **View Merging**은 뷰를 원래 테이블로 대체하여 뷰 수행시 발생하는 오버헤드를 피하고, 원래 테이블의 인덱스를 활용하여 더 효율적인 질의 처리를 가능하게 한다.

아래 질의문 1 처럼 인라인 뷰를 사용하면, 질의 내용을 파악하기 쉽다.

```
/* 질의문 1 */
SELECT *
FROM (SELECT * FROM athlete WHERE nation_code = 'USA') a,
     (SELECT * FROM record WHERE medal = 'G') b
WHERE a.code = b.athlete_code;
```

하지만, View Merging 최적화가 처리 되지 않는다면, 인라인 뷰 *a*와 인라인 뷰 *b*를 각각 수행후 결과를 임시 저장 공간에 보관하고, 임시 저장된 결과를 바탕으로 조인을 수행한다.

이 경우, 임시 저장 공간에 저장된 결과는 인덱스 사용이 불가능하여 많은 성능 저하가 발생한다.

```
/* 질의문 2 */
SELECT *
FROM athlete a, record b
WHERE a.code = b.athlete_code
AND a.nation_code = 'USA'
AND b.medal = 'G'
```

따라서 질의문 1의 뷰 질의들은 뷰를 참조하는 질의와의 머지(merge) 과정을 거쳐 질의문 2와 같은 형태로 변환된다. 이를 **View Merging**이라고 한다.

큐브리드에서는 질의가 다음 조건에 해당하면 **View Merging**을 수행할 수 없다.

1. 뷰에 **NO_MERGE** 힌트가 사용된 경우
2. 뷰가 **CONNECT BY**를 포함한 경우
3. 뷰가 **DISTINCT**문을 포함한 경우
4. 뷰가 **CTE**(Common Table Expressions)인 경우
5. 뷰를 **OUTER JOIN**하는 경우
6. 뷰에 집계함수나 분석함수를 사용하는 경우
7. 뷰에 **ROWNUM**, **LIMIT**또는 **GROUPBY_NUM ()**, **INST_NUM ()**, **ORDERBY_NUM ()**을 사용하는 경우
8. **Correlated Subquery**를 사용하여 작성된 경우
9. 뷰가 메소드를 포함한 경우
10. 뷰가 **RANDOM ()**, **DRANDOM ()**, **SYS_GUID ()**를 포함한 경우

다음은 뷰에 **NO_MERGE** 힌트가 사용된 예시이다.

```
SELECT *
FROM (SELECT /*+ NO_MERGE*/ * FROM athlete WHERE nation_code =
'USA') a,
(SELECT /*+ NO_MERGE*/ * FROM record WHERE medal = 'G') b
WHERE a.code = b.athlete_code;
```

뷰에 **NO_MERGE** 힌트를 사용할 경우, 해당 뷰는 **View Merging**의 대상이 되지 않는다.

다음은 뷰가 **CONNECT BY**를 포함한 예시이다.

```
-- Creating tree table and then inserting data
CREATE TABLE tree(id INT, mgrid INT, name VARCHAR(32), birthyear
INT);

INSERT INTO tree VALUES (1,NULL,'Kim', 1963);
INSERT INTO tree VALUES (2,NULL,'Moy', 1958);
INSERT INTO tree VALUES (3,1,'Jonas', 1976);
INSERT INTO tree VALUES (4,1,'Smith', 1974);
INSERT INTO tree VALUES (5,2,'Verma', 1973);
INSERT INTO tree VALUES (6,2,'Foster', 1972);
INSERT INTO tree VALUES (7,6,'Brown', 1981);

-- Executing a hierarchical query with CONNECT BY clause
SELECT *
FROM (SELECT * FROM tree WHERE birthyear = 1973) t
CONNECT BY PRIOR t.id=t.mgrid;
```

위 질의문에서는 **CONNECT BY**절을 사용하기 때문에 **View Merging**을 수행할 수 없다.

다음은 뷰가 **DISTINCT**문을 포함한 예시이다.

```
SELECT * FROM (SELECT DISTINCT host_year FROM record) T;
```

위 질의문의 뷰 내부에 **DISTINCT**문이 사용되어 **View Merging**을 수행할 수 없다.

다음은 뷰가 **CTE**(Common Table Expressions)인 예시이다.

```
WITH cte AS (SELECT * FROM athlete WHERE gender = 'M')
SELECT * FROM cte WHERE cte.nation_code = 'USA';
```

위와 같이 **CTE**를 포함한 질의문은 **View Merging**을 수행할 수 없다.

다음은 뷰를 **OUTER JOIN**하는 예시이다.

```
SELECT *
FROM athlete a
LEFT OUTER JOIN (SELECT * FROM record WHERE host_year = 2020) b
ON a.code = b.athlete_code;
```

위와 같이 **OUTER JOIN**을 수행하는 경우에는 **View Merging**을 수행할 수 없다.

다음은 뷰에 집계함수나 분석함수를 사용하는 예시이다.

```
SELECT *
FROM (SELECT AVG(host_year) FROM record WHERE medal = 'G') a;
```

뷰에 집계함수나 분석함수를 포함한 질의문의 경우 **View Merging**의 대상이 되지 않는다.

다음은 뷰에 **ROWNUM**, **LIMIT** 또는 **GROUPBY_NUM ()**, **INST_NUM ()**, **ORDERBY_NUM ()**을 사용하는 예시이다.

```
SELECT *
FROM (SELECT gender, rownum FROM athlete WHERE rownum < 15) a
WHERE gender = 'M';
```

뷰에 **ROWNUM**, **LIMIT** 또는 **GROUPBY_NUM ()**, **INST_NUM ()**, **ORDERBY_NUM ()**을 사용한 질의문의 경우 **View Merging**이 불가능하다. 하지만 부분범위 처리를 위해 **ROWNUM**을 사용하는 경우 **ROWNUM** 등이 있더라도 **View Merging**을 수행한다. 아래 두가지 조건을 만족하는 경우이다.

1. **WHERE** 절에 **ROWNUM**을 포함한 조회조건만 있다.
2. **FROM** 절에 한개의 부질의만 있다.

```
--csql> ;plan detail
SELECT *
FROM (SELECT name, rownum rn
      FROM (SELECT name FROM athlete WHERE nation_code = 'KOR') a
      WHERE rownum < 15) b
WHERE rn > 5;
```

Query stmt:

```
select b.[name], (rownum) from athlete b where ((rownum)> ? :0 and
(rownum)< ? :1 ) and (b.nation_code= ? :2 )
```

다음은 **Correlated Subquery**를 사용하여 작성된 예시이다

```
SELECT COUNT(*)
FROM athlete a,
(SELECT * FROM record r WHERE a.code = r.athlete_code) b;
```

Correlated Subquery를 사용한 질의문의 경우 **View Merging**이 불가능하다.

다음은 뷰가 **RANDOM ()**, **DRANDOM ()**, **SYS_GUID ()**를 포함한 예시이다.

```

SELECT *
FROM (SELECT RANDOM (), code FROM athlete WHERE nation_code =
'USA') a,
      (SELECT SYS_GUID (), athlete_code FROM record WHERE medal =
'G') b
WHERE a.code = b.athlete_code;

```

뷰가 **RANDOM ()**, **DRANDOM ()**, **SYS_GUID ()**를 포함한 질의문의 경우 **View Merging**이 불가능하다.

View Merging 최적화를 수행할 수 없는 경우, **CUBRID**는 부질의의 **SELECT-LIST** 항목을 최적화 한다. 다음은 부질의의 **SELECT-LIST**가 어떻게 최적화 되는지 보여주는 예제이다.

```

--csql> ;plan detail
SELECT /** recompile */ aa.host_year, aa.rn
FROM (select host_year,
            (select name from event where code = a.event_code)
      event_name,
            (select name from athlete where code = a.athlete_code)
      athlete_name,
            rownum rn
      from game a
      where medal = 'G') aa
WHERE host_year = '2004';

```

Query stmt:

```

select aa.host_year, aa.rn from (select a.host_year, (rownum) as
[ rn ] from game a where a.medal= ? :1 ) aa (host_year, rn) where
aa.host_year= ? :0

```

Predicate Push

Predicate Push는 외부의 조건절을 뷰 안으로 밀어 넣는 최적화이다.

이를 통해 뷰 수행시 추가된 조건에 의해 더 적은 양의 데이터만 조회할 수 있어 전체 처리량을 줄일 수 있다.

예를 들어 아래 질의 처럼 인라인 뷰와 테이블을 $a.code = r.athlete\_code$ 의 조인 조건으로 조인을 처리할 때, 조건절 중 인라인 뷰에 해당되는 조건절들을 인라인 뷰 안쪽에 밀어 넣을 수 있다면 조인해야 할 데이터양을 줄일 수 있다.

```

SELECT a.name, r.score
FROM (SELECT name, nation_code, code, count(*) cnt FROM athlete
GROUP BY name, nation_code) a, record r

```

```
WHERE a.code = r.athlete_code
AND a.nation_code = 'KOR';
```

위 질의 중 인라인 뷰 내부에는 조건절이 없다. 만약 쿼리 변환이 수행되지 않았다면, *athlete* 테이블을 full scan하여 생성된 임시 결과와 record 테이블을 조인 수행후에 *a.nation_code* = 'KOR' 조건을 필터링했을 것이다.

하지만 **Predicate Push**를 통해서 다음과 같이 질의가 변환된다면, 더 적은 양의 데이터만 조회되도록 최적화할 수 있다.

```
SELECT a.name, r.score
FROM (SELECT name, nation_code, code, count(*) cnt FROM athlete
WHERE nation_code = 'KOR' GROUP BY name, nation_code ) a, record r
WHERE a.code = r.athlete_code;
```

아래는 여러개의 부질의가 있는 상황에서 **Predicate Push**가 어떻게 동작하는지 보여주는 예시이다.

```
--csql> ;plan detail
SELECT a.host_year
FROM (select distinct host_year, event_code, athlete_code from game
where host_year = 2004) a,
      (select distinct host_year, event_code, athlete_code from game
where event_code = 20021) b
WHERE a.host_year = b.host_year
AND a.event_code = b.event_code
AND a.athlete_code = b.athlete_code;
```

Query stmt:

```
select a.host_year
from (select distinct game.host_year, game.event_code,
game.athlete_code from game game where game.host_year=? :0 and
game.event_code=20021) a (host_year, event_code, athlete_code),
      (select distinct game.host_year, game.event_code,
game.athlete_code from game game where game.event_code=? :1 and
game.host_year=2004) b (host_year, event_code, athlete_code)
where a.athlete_code=b.athlete_code and a.host_year=b.host_year and
a.event_code=b.event_code
```

조희조건 **game.event_code=20021** 와 **game.host_year=2004** 가 주질의의 조인조건에 의해서 각각의 부질의에 추가된다.

다음의 경우에는 **Predicate Push**가 수행되지 않는다.

1. 주 질의에 **NO_PUSH_PRED** 힌트가 사용된 경우

2. 주 질의가 **CONNECT BY**를 포함한 경우
3. 뷰에 집계함수나 분석함수를 사용하는 경우
4. 뷰에 **ROWNUM**, **LIMIT** 또는 **GROUPBY_NUM ()**, **INST_NUM ()**, **ORDERBY_NUM ()**을 사용하는 경우
5. **Correlated Subquery**를 사용하여 작성된 경우
6. 조건절에 부질의가 사용된 경우
7. 뷰가 메소드를 포함한 경우
8. 푸시될 조건절이나 뷰 내부의 **Predicate Push** 대상에 **RANDOM ()**, **DRANDOM ()**, **SYS_GUID ()**를 사용하는 경우
9. **OUTER JOIN**을 수행할 때 푸시될 조건절이나 뷰 내부의 **Predicate Push** 대상에 다음을 사용하는 경우:
 - **ON**절에 조건절이 작성된 경우
 - **NULL** 변환 함수 (**COALESCE ()**, **NVL ()**, **NVL2 ()**, **DECODE ()**, **IF ()**, **IFNULL ()**, **CONCAT_WS ()**)
 - **IS NULL**, **CASE** 문

다음은 주 질의에 **NO_PUSH_PRED** 힌트가 사용된 예시이다.

```
SELECT /*+ NO_PUSH_PRED*/ a.name, r.score
FROM (SELECT name, nation_code, code, count(*) cnt FROM athlete
GROUP BY name, nation_code) a, record r
WHERE a.code = r.athlete_code
AND a.nation_code = 'KOR';
```

주 질의에 **NO_PUSH_PRED** 힌트가 사용된 경우, **Predicate Push**가 수행되지 않는다.

다음은 질의가 **OUTER JOIN**을 수행할 때 **ON**절의 조건에 푸시될 조건절이 있는 예시이다.

```
SELECT a.name, r.score
FROM (SELECT * FROM athlete WHERE gender = 'M') a
LEFT OUTER JOIN record r ON a.code = r.athlete_code AND
a.nation_code = 'KOR';
```

이 경우, *a.nation_code = 'KOR'*는 **LEFT OUTER JOIN** 수행 시 **ON** 절에 있는데, 이러한 형태로 **ON** 절의 조건절은 **Predicate Push** 대상이 아니다.

다음 질의는 **OUTER JOIN**을 수행할 때 푸시될 조건절이나 뷰 내부의 **Predicate Push** 대상에 **NULL** 변환 함수를 사용하는 예시이다.


```

SELECT a.name, r.score
FROM athlete a
      LEFT OUTER JOIN (SELECT * FROM record WHERE medal = 'G') r
ON a.code = r.athlete_code
WHERE NVL(r.score, '0') = '0';

```

OUTER JOIN을 수행할 때 푸시될 조건절이나 뷰 내부의 **Predicate Push** 대상에 **NULL** 변환 함수를 사용한 경우 **Predicate Push** 대상이 아니다.

부질의 중첩 제거

부질의 중첩 제거는 **WHERE** 절의 부질의를 매번 수행하여 필터하는 방식에서 동일한 결과를 보장하는 조인방식으로 변환하여 성능을 개선한다. **WHERE** 절의 부질의는 연산자에 따라서 다른 성질을 가질 수 있으며, 연산자 **IN**이나 **EXISTS** 처럼 여러 행을 결과로 가질 수 있는 부질의가 최적화 대상이다. **CUBRID**는 **IN** 연산자에 대해서만 부분적으로 지원한다.

아래는 **IN** 연산자에 대해서 조인 질의로 변환하는 예시이다.

```

--csql> set optimization level 513;
SELECT game_date
FROM game
WHERE (host_year,event_code,athlete_code) IN (select
host_year,event_code,athlete_code
                                           from game
                                           where nation_code =
'KOR' and medal = 'G');

```

Query plan:

```

idx-join (inner join)
  outer: sscan
        class: av1861 node[1]
        cost:  1 card 25
  inner: iscan
        class: game node[0]
        index: pk_game_host_year_event_code_athlete_code
term[0] AND term[1] AND term[2]
        cost:  3 card 8653
cost:  17 card 1

```

Query stmt:

```

select game.game_date from game game, (select distinct
game.host_year, game.event_code, game.athlete_code from game game
where game.medal= ?:0  and game.nation_code= ?:1 ) av1861 (av_1,

```

```
av_2, av_3) where game.host_year=av1861.av_1 and
game.event_code=av1861.av_2 and game.athlete_code=av1861.av_3
```

참고

CUBRID 11.0 부터 **IN** 연산자의 인자에 여러개의 컬럼을 조회하는 부질의가 허용 된다.

조건절 이행

조건절 이행은 논리적으로 생성될 수 있는 조건절을 추가 함으로써 질의 성능을 개선한다.

아래는 조건절 이행 최적화로 **b.col1 = 3** 조회 조건이 추가된 예시이다.

```
CREATE TABLE t1 (col1 int, col2 int, col3 int);
CREATE TABLE t2 (col1 int, col2 int, col3 int);

--csql> ;plan detail
SELECT /*+ recompile */ *
FROM t1 a, t2 b
WHERE a.col1 = b.col1
AND a.col1 = 3;
```

Query stmt:

```
select a.col1, a.col2, a.col3, b.col1, b.col2, b.col3 from t1 a, t2
b where b.col1= ?:0 and a.col1= ?:1 and a.col1=b.col1
```

쿼리 캐시

QUERY_CACHE 힌트는 반복적으로 실행되는 쿼리의 성능을 향상시키는 데 사용할 수 있으며, 쿼리는 전용 메모리 영역에 캐시되고 그 결과도 별도의 디스크 공간에 캐시된다. 단, 힌트는 SELECT 쿼리에만 적용된다. 그러나 다음과 같은 경우에는 쿼리에 힌트를 적용 할 수 없으며 힌트는 의미가 없어진다.

- 아래와 같이 질의에 시스템 시간 또는 날짜 관련 속성이 포함된 경우.

예) SELECT SYSDATE, ADDDATE (SYSDATE, INTERVAL -24 HOUR), ADDDATE (SYSDATE, -1);

- SERIAL 관련 속성이 포함된 경우.
- 컬럼 경로 관련 속성이 포함된 경우.

- 메서드가 포함된 경우.
- 저장 프로 시저 또는 저장 함수가 포함된 경우.
- dual, _db_attribute 등과 같은 시스템 테이블이 포함된 경우.
- sys_guid ()와 같은 시스템 함수가 포함된 경우.

힌트가 설정되고 새 SELECT 질의가 처리될 때 질의가 캐시에서 발견되면 쿼리 캐시를 사용한다. 동일한 데이터베이스에서 동일한 질의 텍스트와 동일한 바인딩 값을 사용하는 경우 질의는 동일한 것으로 간주된다. 캐시된 질의를 찾을 수 없는 경우 질의가 처리된 다음 결과와 함께 캐시가 생성된다. 질의가 캐시에서 발견되면 캐시된 영역에서 결과를 가져온다. CSQL에서는 아래 예제와 같이 COUNT 함수를 사용하여 질의를 반복적으로 실행할 때 개선된 성능을 쉽게 측정 할 수 있다. 첫번째 질의에 대한 결과는 캐시가 되어 있지 않아서 느리지만, 두 번째 질의의 결과는 캐시 된 영역에서 가져오므로 응답 시간이 이전 질의보다 훨씬 빠르다.

```
csql> SELECT /*+ QUERY_CACHE */ count(*) FROM game;

=== <Result of SELECT Command in Line 1> ===

      count(*)
=====
      8653

1 row selected. (0.107082 sec) Committed.

1 command(s) successfully processed.

csql> SELECT /*+ QUERY_CACHE */ count(*) FROM game;

=== <Result of SELECT Command in Line 1> ===

      count(*)
=====
      8653

1 row selected. (0.003932 sec) Committed.

1 command(s) successfully processed.
```

다음과 같이 CSQL에 세션 명령 `;info qcache` 를 입력하여 쿼리가 캐시되는지 여부를 확인할 수 있다.

```
csql> ;info qcache

LIST_CACHE {
  n_hts 1010
  n_entries 1  n_pages 1
  lookup_counter 1
```

```

hit_counter 1
miss_counter 0
full_counter 0
}

list_hts[0] 0x6a74d10
HTABLE NAME = list file cache (DB_VALUE list), SIZE = 211, REHASH_AT
= 147,
NENTRIES = 1, NPREALLOC = 0, NCOLLISIONS = 0

HASH AT 0
LIST_CACHE_ENTRY (0x6c46d18) {
  param_values = [ ]
  list_id = { type_list { 1 integer/1 } tuple_cnt 1 page_cnt 1
first_vpid { 65 32766 } last_vpid { 65 32766 } lasttpl_len 24
query_id 2
  temp_vfid { 64 32766 } }
  uncommitted_marker = false
  tran_isolation = 4
  tran_index_array = [ ]
  last_ta_idx = 0
  query_string = select /*+ QUERY_CACHE */ count(*) from [game]
[game]?193="en_US";194="en_US";249="Asia/Seoul";user=0|833|1
  time_created = 11/23/20 16:07:12.779703
  time_last_used = 11/23/20 16:07:22.772330
  ref_count = 1
  deletion_marker = false
}

```

캐시 된 질의는 결과 화면 중간에 **query_string** 으로 표시되며 각 **n_entries** 및 **n_pages** 는 캐시된 질의 수와 캐시 된 결과의 페이지 수를 나타낸다. **n_entries** 는 파라미터 **max_QUERY_CACHE_entries** 의 값으로 제한되고 **n_pages** 는 **QUERY_CACHE_size_in_pages** 의 값으로 제한된다. **n_entries** 가 초과하거나 **n_pages** 가 초과하면 캐시 항목 중 일부가 삭제될 후보로 선택되어 삭제되고, 삭제되는 캐시는 **max_QUERY_CACHE_entries** 값과 **QUERY_CACHE_size_in_pages** 값의 약 20%이다.

11.4 버전부터는 부질의 대상으로 QUERY_CACHE 힌트를 사용할 수 있으며, 힌트를 사용할 수 있는 부질의는 아래와 같다.

1) CTE 쿼리 WITH절에 포함되는 비재귀적 부분에 QUERY_CACHE 힌트를 지정한 경우

```

WITH
  of_drones AS (SELECT /*+ QUERY_CACHE */ item, 'drones' FROM
products WHERE parent_id = 1),
  of_cars AS (SELECT /*+ QUERY_CACHE */ item, 'cars' FROM products
WHERE parent_id = 5)
SELECT * FROM of_drones UNION ALL SELECT * FROM of_cars ORDER BY 1;

```

WITH절에 포함되는 재귀 부분 중 다른 CTE를 참조하는 부질의에 QUERY_CACHE 힌트를 지정한 경우

```
WITH
  RECURSIVE cars (id, parent_id, item, price) AS (
    SELECT /** QUERY_CACHE */ id, parent_id, item,
    price
      FROM products WHERE item LIKE 'Car%'
    UNION ALL
    SELECT /** QUERY_CACHE */ p.id, p.parent_id,
    p.item, p.price
      FROM products p
    INNER JOIN cars rec_cars ON p.parent_id =
    rec_cars.id)
  SELECT item, price FROM cars ORDER BY 1;
```

위 UNION 쿼리에서 첫 번째 쿼리는 캐시 되지만, 두 번째 쿼리는 다른 테이블을 참조하기 때문에 캐시 되지 않는다.

1. 다른 테이블을 참조하지 않는 부질의에 QUERY_CACHE 힌트를 지정한 경우

```
SELECT h.host_year, (SELECT /** QUERY_CACHE */ host_nation FROM
olympic o WHERE o.host_year > 1994 limit 1) AS host_nation,
      h.event_code, h.score, h.unit
FROM history h;

SELECT name, capital, list(SELECT /** QUERY_CACHE */ host_city FROM
olympic WHERE host_nation like 'K%') AS host_cities
FROM nation;
```

단, 아래와 같이 부질의 내에서 다른 테이블을 참조하는 경우에는 QUERY_CACHE 힌트를 지정해도 캐시 되지 않는다.

```
SELECT h.host_year, (SELECT /** QUERY_CACHE */ host_nation FROM
olympic o WHERE o.host_year=h.host_year) AS host_nation,
      h.event_code, h.score, h.unit
FROM history h;

SELECT name, capital, list(SELECT /** QUERY_CACHE */ host_city FROM
olympic WHERE host_nation = name) AS host_cities
FROM nation;
```

캐시된 질의는 결과 화면 중간에 **query_string** 으로 표시되며 각 **n_entries** 및 **n_pages** 는 캐시된 질의 수와 캐시 된 결과의 페이지 수를 나타낸다. **n_entries** 는 파라미터 **max_query_cache_entries** 의 값으로 제한되고 **n_pages** 는 **query_cache_size_in_pages** 의 값으로 제한된다. **n_entries** 가 초과되거나 **n_pages** 가 초과되면 캐시 항목 중 일부가 삭제될 후보로 선택되어 삭제되고, 삭제되는 캐시는 **max_query_cache_entries** 값과 **query_cache_size_in_pages** 값의 약 20% 이다.

질의 프로파일링 요청과 함께 질의 수행시 비상관 서브쿼리(uncorrelated subquery) 캐시가 적용된 부질의 하위 정보로 비상관 서브쿼리 캐시에 대한 프로파일링 정보가 출력된다.

다음은 부질의 수행시 서브 쿼리 캐시 프로파일링 정보가 표시되는 예시이다.

```
SELECT name, capital, list (
    SELECT /** QUERY_CACHE */ host_city
    FROM olympic
    WHERE host_nation like 'K%'
) AS host_cities
FROM nation;

Trace Statistics:
  SELECT (time: 13, fetch: 2584, fetch_time: 0, ioread: 0)
    SCAN (table: public.nation), (heap time: 2, fetch: 864, ioread:
0, readrows: 215, rows: 215)
    SUBQUERY (uncorrelated)
      SELECT (time: 0, fetch: 0, fetch_time: 0, ioread: 0)
        RESULT CACHE (reference count: 1)
```

비상관 서브쿼리 캐시의 프로파일링 항목에 대한 설명은 다음과 같다.

- **reference count**: 부질의에서 참조한 캐시의 참조 횟수.

서브 쿼리 캐시 (correlated)

서브 쿼리 캐시 최적화는 상관 부질의(correlated subquery)를 포함한 질의 성능을 향상시키는데 사용할 수 있으며, 부질의 결과는 부질의마다 독립적인 공간에 캐시된다. 서브 쿼리 캐시 최적화는 기본적으로 적용되므로 질의 수행시 캐시를 사용하지 않으려면 **NO_SUBQUERY_CACHE** 힌트를 해당 부질의에 사용해야 한다.

서브 쿼리 캐시 최적화는 상관 부질의가 SELECT 절에 있는 경우 적용되며, 처리 절차는 다음과 같다. 재 실행되는 상관 부질의의 검색 조건 컬럼 값이 동일한 경우 부질의 수행 대신 캐시된 결과를 사용한다. 만약, 캐시된 값을 찾을 수 없는 경우 부질의 수행 후 참조한 컬럼 값과 결과를 서브 쿼리 캐시에 저장한다.

질의 프로파일링 요청과 함께 질의 수행시 서브 쿼리 캐시 최적화가 적용된 부질의의 하위 정보로 서브 쿼리 캐시에 대한 프로파일링 정보가 출력된다.

다음은 상관 부질의 수행시 서브 쿼리 캐시 프로파일링 정보가 표시되는 예시이다

```
csql> SELECT /** RECOMPILE NO_MERGE */
    (SELECT t1_pk FROM t1 b WHERE b.t1_pk = a.c3)
FROM t1 a
WHERE a.c2 >= 1;
```

```
Trace Statistics:
  SELECT (time: 108, fetch: 103639, fetch_time: 13, ioread: 0)
    SCAN (table: dba.t1), (heap time: 70, fetch: 100384, ioread: 0,
readrows: 100000, rows: 99000)
    SUBQUERY (correlated)
      SELECT (time: 4, fetch: 2970, fetch_time: 0, ioread: 0)
        SCAN (index: dba.t1.pk_t1), (btree time: 2, fetch: 1980,
ioread: 0, readkeys: 990, filteredkeys: 0, rows: 990, covered: true)
        SUBQUERY_CACHE (hit: 98010, miss: 990, size: 269384, status:
enabled)
```

서브 쿼리 캐시의 프로파일링 항목에 대한 설명은 다음과 같다.

- **hit**: 질의 실행 대신 캐시된 영역에서 결과를 가져온 횟수.
- **miss**: 질의를 실행한 후 결과를 캐시한 횟수.
- **size**: 서브 쿼리 캐시에 사용된 메모리 크기.
- **status**: 질의 종료 시 서브 쿼리 캐시의 활성화 여부.

size가 설정해둔 값을 초과하는 경우에는 서브 쿼리 캐시가 질의 수행 도중 비활성화되며 SQL 트레이스 정보에서 **status**가 disabled로 출력된다. 또한, **hit** 비율이 90% 미만일 경우, 서브 쿼리 캐시의 메모리 크기가 설정해둔 값을 초과하지 않더라도 캐시 miss가 높다고 판단해서 질의 수행 도중 캐시가 비활성화 될 수 있다.

참고

hit 비율(hit ratio)은 다음과 같이 계산한다.

$$\text{hit ratio} = \frac{\text{hit}}{\text{hit} + \text{miss}}$$

다음은 서브 쿼리 캐시의 사용 유무에 따른 성능 차이를 확인하기 위해 반복 수행되는 상관 부질의 결과를 count하는 예제 질의이다. 다음은 예제 질의를 수행하기 위한 데이터를 준비하는 질의이다.

```
# Prepare data
csql> DROP TABLE IF EXISTS t1;

csql> CREATE TABLE t1 AS
  SELECT
    ROWNUM AS t1_pk,
    MOD(ROWNUM, 10) AS c1,
    MOD(ROWNUM, 100) AS c2,
    MOD(ROWNUM, 1000) AS c3
  FROM
    db_class a, db_class b, db_class c, db_class d,
```

```

db_class e, db_class f
    LIMIT 100000;

csql> ALTER TABLE t1 ADD CONSTRAINT PRIMARY KEY pk_t1 (t1_pk);

csql> CREATE TABLE t2 AS
    SELECT
        ROWNUM as c1,
        1 as c2,
        TO_CHAR(ROWNUM * 1000, '0999') as code
    FROM
        db_class a, db_class b
    LIMIT 10;

csql> update statistics on t1 with fullscan;

csql> ;trace on

```

질의 #1는 서브 쿼리 캐시 최적화를 사용하여 질의를 수행했고, 질의 #2는 **NO_SUBQUERY_CACHE** 힌트를 사용하여 서브 쿼리 캐시를 비 활성화하여 질의를 수행했다. 두 질의의 결과를 비교해보면 서브 쿼리 캐시 최적화를 사용한 것이 응답 수행이 개선된 것을 확인할 수 있다.

Query #1

Query #2

```

csql> SELECT COUNT(*)
    FROM (
        SELECT /**
RECOMPILE NO_MERGE */
        (SELECT
t1_pk FROM t1 b WHERE b.t1_pk
= a.c3)
        FROM t1 a
        WHERE a.c2 >= 1
    );

=== <Result of SELECT Command
in Line 2> ===

count(*)
=====
          99000

1 row selected. (0.128251 sec)
Committed. (0.000010 sec)

```

```

csql> SELECT COUNT(*)
    FROM (
        SELECT /**
RECOMPILE NO_MERGE */
        (SELECT /
** NO_SUBQUERY_CACHE */ t1_pk
FROM t1 b WHERE b.t1_pk =
a.c3)
        FROM t1 a
        WHERE a.c2 >= 1
    );

=== <Result of SELECT Command
in Line 2> ===

count(*)
=====
          99000

1 row selected. (0.626199
sec) Committed. (0.000011 sec)

```


Query #1	Query #2
1 command(s) successfully processed.	1 command(s) successfully processed.

서브 쿼리 캐시는 다음과 같은 상황에 동작하지 않는다:

- 상관 부질의가 상관 부질을 포함한 경우. (단, 다른 상관 하위 쿼리가 포함되지 않은 최 하단의 상관 부질의는 서브 쿼리 캐시 최적화 작업이 수행됨)
- 부질의가 단일 행 부질의가 아닌 경우.
- 부질의가 CONNECT BY 절을 포함한 경우.
- 부질의에 OID 관련 기능이 포함된 경우.
- 부질의가 **NO_SUBQUERY_CACHE** 힌트를 포함한 경우.
- 새로운 결과를 저장할 때 설정한 서브 쿼리 캐시 크기(기본값: 2MB)를 초과하는 경우.
- random (), sys_guid ()와 같이 실행될 때마다 결과가 바뀌는 함수가 포함된 경우.

다음 예시는 상관 부질의가 상관 부질을 포함한 경우, 최하단의 상관부질의만 서브 쿼리 캐시가 사용되고, 상단의 상관 부질의에는 서브 쿼리 캐시가 적용되지 않는 예시이다.

```
csql> SELECT /*+ recompile */
      (
        SELECT
          (
            SELECT c.code
            FROM t2 c
            WHERE c.c1 = b.c1
          )
        FROM t1 b
        WHERE b.t1_pk = a.c1
      ) s
FROM t1 a
WHERE a.c3 = 1;
```

Trace Statistics:

```
SELECT (time: 56, fetch: 100785, fetch_time: 10, ioread: 0)
SCAN (table: dba.t1), (heap time: 55, fetch: 100384, ioread:
0, readrows: 100000, rows: 100)
SUBQUERY (correlated)
SELECT (time: 0, fetch: 401, fetch_time: 0, ioread: 0)
SCAN (index: dba.t1.pk_t1), (btree time: 0, fetch: 300,
ioread: 0, readkeys: 100, filteredkeys: 0, rows: 100) (lookup time:
```

```
0, rows: 100)
  SUBQUERY (correlated)
    SELECT (time: 0, fetch: 1, fetch_time: 0, ioread: 0)
      SCAN (table: dba.t2), (heap time: 0, fetch: 1,
ioread: 0, readrows: 10, rows: 1)
      SUBQUERY_CACHE (hit: 99, miss: 1, size: 150704,
status: enabled)
```

다음은 상관 부질의에 random () 와 같이 실행될 때마다 결과가 바뀌는 함수가 포함된 경우 서브 쿼리 캐시가 비활성화되는 예시이다.

```
csql> WITH cte_1 AS
      (SELECT
        DISTINCT (SELECT random(1) FROM t2 b WHERE b.c1 = a.c1
AND b.c2 = 1) v
        FROM t1 a
        WHERE a.c2 = 1
      ) SELECT count(*) FROM cte_1;

Trace Statistics:
  SELECT (time: 65, fetch: 101384, fetch_time: 9, ioread: 0)
    SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 1000,
rows: 1000)
    SUBQUERY (uncorrelated)
      CTE (non_recursive_part)
        SELECT (time: 65, fetch: 101384, fetch_time: 9, ioread:
0)
          SCAN (table: dba.t1), (heap time: 59, fetch: 100384,
ioread: 0, readrows: 100000, rows: 1000)
          ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
          SUBQUERY (correlated)
            SELECT (time: 4, fetch: 1000, fetch_time: 0, ioread:
0)
              SCAN (table: dba.t2), (heap time: 3, fetch: 1000,
ioread: 0, readrows: 10000, rows: 1000)
```

MEMOIZE

MEMOIZE 최적화는 중첩 루프 조인(nested loop join)에서 inner 테이블의 스캔 결과를 조인 키 값별로 캐시하여, 같은 조인 키 값이 반복되는 조인의 성능을 향상시킨다. outer 테이블에서 읽은 조인 키 값이 캐시에 있으면 inner 테이블의 스캔과 조건 평가를 생략하고 캐시된 결과를 사용하며, 캐시에 없으면 inner 테이블을 스캔한 후 그 결과를 캐시에 저장한다.

MEMOIZE 최적화는 질의 실행 시점에 자동으로 적용되며, 별도의 구문이나 힌트가 없다. 최적화를 사용하지 않으려면 `memoize_memory_limit` 파라미터를 0으로 설정해야 한다.

MEMOIZE 최적화는 다음 조건을 모두 만족하는 스캔에 적용된다.

- 중첩 루프 조인의 inner 테이블 스캔이다. outer 테이블에는 적용되지 않으며, 해시 조인과 정렬 병합 조인에도 적용되지 않는다.
- 조인 조건에서 outer 테이블의 컬럼 값(조인 키)을 찾을 수 있다. 조인 조건이 없는 크로스 조인에는 적용되지 않는다.
- 행 잠금이 필요 없는 읽기이다. **UPDATE** 문이나 **DELETE** 문의 대상 검색, **SELECT ... FOR UPDATE** 질의에는 적용되지 않는다.

inner 쪽 입력이 테이블이 아니라 부질의의 임시 결과 리스트인 경우에도 적용된다.

다음과 같이 캐시된 결과의 재사용이 안전하지 않거나 캐시 키를 만들 수 없는 경우에는 적용되지 않는다.

- inner 테이블에 컬렉션 타입(**SET**, **MULTISET**, **LIST**) 컬럼이 있는 경우
- 시스템 카탈로그 클래스나 MVCC 대상이 아닌 클래스(**db_serial** 등)를 조회하는 경우
- inner 쪽이 DBLink 원격 테이블, **JSON_TABLE**, **SHOW** 문, 집합-메서드 표현식인 경우
- inner 쪽 스캔이 순차 스캔이나 인덱스 스캔이 아닌 경우(스키마 조회 등)
- inner 쪽에 스캔 대상이 둘 이상 있는 경우

캐시가 사용하는 메모리가 `memoize_memory_limit`에 설정한 값을 초과하거나, 조인 키 조회가 1,000회를 넘어선 시점에 **hit** 비율이 50% 미만이면 캐시 유지 비용이 더 크다고 판단하여 캐시를 즉시 해제하고, 이후에는 캐시 없이 inner 테이블을 스캔하는 방식으로 질의 실행을 계속한다. 이때 질의는 실패하지 않으며, 에러나 경고도 발생하지 않는다.

참고

hit 비율(hit ratio)은 다음과 같이 계산한다.

$$\text{hit ratio} = \frac{\text{hit}}{\text{hit} + \text{miss}}$$

질의 프로파일링 요청과 함께 질의 수행시 MEMOIZE 최적화가 적용된 스캔의 하위 정보로 MEMOIZE 캐시에 대한 프로파일링 정보가 출력된다.

다음은 예제 질의를 수행하기 위한 데이터를 준비하는 질의이다. `orders` 테이블의 `customer_id` 컬럼에는 100개의 값만 반복해서 등장한다.

```
CREATE TABLE customers (id INT PRIMARY KEY, grade INT);

INSERT INTO customers
```

```

SELECT ROWNUM, MOD(ROWNUM, 5)
FROM db_class a, db_class b
LIMIT 100;

CREATE TABLE orders (id INT PRIMARY KEY, customer_id INT, amount
INT);

INSERT INTO orders
SELECT ROWNUM, MOD(ROWNUM, 100) + 1, MOD(ROWNUM, 1000)
FROM db_class a, db_class b, db_class c, db_class d
LIMIT 1000000;

UPDATE STATISTICS ON customers, orders WITH FULLSCAN;

```

다음은 중첩 루프 조인 수행시 MEMOIZE 프로파일링 정보가 표시되는 예시이다. 예제 질의는 중첩 루프 조인을 유도하기 위해 **ORDERED** 와 **USE_NL** 힌트를, 병렬 스캔의 영향을 배제하기 위해 **NO_PARALLEL_SCAN** 힌트를 사용한다.

```

csql> ;trace on

SELECT /*+ RECOMPILE ORDERED USE_NL NO_PARALLEL_SCAN */ SUM(o.amount)
FROM orders o INNER JOIN customers c ON c.id = o.customer_id
WHERE c.grade = 1;

```

```

Trace Statistics:
  SELECT (time: 491, fetch: 5132, fetch_time: 1, ioread: 0)
    SCAN (table: dba.orders), (heap time: 198, fetch: 4931, ioread:
0, readrows: 1000000, rows: 1000000)
      SCAN (index: dba.customers.pk_customers_id), (btree time: 0,
fetch: 200, ioread: 0, readkeys: 100, filteredkeys: 100, rows: 100)
(lookup time: 0, rows: 20)
      MEMOIZE (time: 184, hit: 999900, miss: 100, size: 18KB,
enabled: true)

```

100개의 조인 키 값이 각각 처음 조회될 때만 *customers* 테이블의 인덱스 스캔이 수행되고(**miss**: 100), 나머지 999,900회의 조회는 캐시에서 처리되었다(**hit**: 999900).

MEMOIZE 캐시의 프로파일링 항목에 대한 설명은 다음과 같다.

- **time**: 캐시 조회와 저장에 사용한 누적 시간(밀리초).
- **hit**: inner 테이블 스캔 대신 캐시된 결과를 사용한 조인 키 조회 횟수.
- **miss**: 캐시에서 조인 키를 찾지 못해 inner 테이블을 스캔한 조인 키 조회 횟수.
- **size**: 캐시에 사용된 메모리 크기.
- **enabled**: 질의 종료 시 캐시의 활성화 여부.

참고

MEMOIZE 항목은 **hit**가 1 이상인 경우에만 출력된다. 프로파일링 결과에 **MEMOIZE** 항목이 없다고 해서 MEMOIZE 최적화가 시도되지 않았다는 뜻은 아니다.

참고

hit와 **miss**는 결과 행 수가 아니라 조인 키 조회 수를 기준으로 집계된다.

병렬 실행과 결합되면 워커 스레드마다 캐시가 별도로 생성되며, 프로파일링 정보에는 워커들의 통계를 합산한 값이 출력된다. 이 경우 하나의 워커에서라도 캐시가 해제되면 **MEMOIZE** 항목 전체가 출력되지 않는다.

질의 #1은 MEMOIZE 최적화를 사용하여 질의를 수행했고, 질의 #2는 **memoize_memory_limit**를 0으로 설정하여 MEMOIZE 최적화를 비활성화하고 질의를 수행했다. 두 질의의 결과를 비교해보면 MEMOIZE 최적화를 사용한 것이 응답 시간이 개선된 것을 확인할 수 있다.

Query #1

```

csql> SELECT /** RECOMPILE
ORDERED USE_NL
NO_PARALLEL_SCAN */
SUM(o.amount)
      FROM orders o INNER
JOIN customers c ON c.id =
o.customer_id
      WHERE c.grade = 1;

=== <Result of SELECT Command
in Line 2> ===

      sum(o.amount)
=====
      99500000

1 row selected. (0.407000 sec)
Committed. (0.000000 sec)

```

Query #2

```

csql> SET SYSTEM PARAMETERS
'memoize_memory_limit=0';

csql> SELECT /** RECOMPILE
ORDERED USE_NL
NO_PARALLEL_SCAN */
SUM(o.amount)
      FROM orders o INNER
JOIN customers c ON c.id =
o.customer_id
      WHERE c.grade = 1;

=== <Result of SELECT Command
in Line 3> ===

      sum(o.amount)
=====
      99500000

1 row selected. (1.407000
sec) Committed. (0.000000 sec)

```

병렬 실행

병렬 실행

CUBRID는 대량의 데이터를 효율적으로 처리하기 위해 병렬 질의 실행 기능을 제공한다. 병렬 질의 실행은 하나의 질의를 여러 개의 작업 단위로 분할하고, 이를 여러 워커 스레드가 동시에 수행함으로써 응답 시간을 획기적으로 단축한다.

개요

병렬 질의는 다음과 같은 주요 기능을 제공한다:

- **병렬 스캔(Parallel Scan)**: 여러 워커 스레드가 입력 데이터(힙, 임시 리스트, 인덱스)를 나누어 스캔하여 대용량 탐색의 성능을 향상시킨다. 스캔 종류에 따라 다음 세 가지로 세분화된다.
 - **병렬 힙 스캔(Parallel Heap Scan)**: 테이블의 힙 페이지를 섹터(sector) 단위로 분할하여 워커들이 동시에 스캔한다.
 - **병렬 리스트 스캔(Parallel List Scan)**: 부질의 등으로 생성된 임시 결과 리스트(list file)를 섹터(sector) 단위로 분할하여 워커들이 동시에 스캔한다.
 - **병렬 인덱스 스캔(Parallel Index Scan)**: B+트리 인덱스의 리프 페이지 체인을 워커들이 공유 커서로 협력하여 좌→우(또는 우→좌)로 진행하면서 스캔한다.
- **병렬 부질의 실행(Parallel Uncorrelated Subquery Execution)** : 서로 독립적인 부질의(uncorrelated subquery)들을 워커들이 각자 맡아 동시에 처리하여 질의의 응답 속도를 개선한다.
- **병렬 해시 조인(Parallel Hash Join)**: 빌드(build) 단계와 프로브(probe) 단계를 병렬화하여, 해시 조인 연산시 응답 속도를 개선한다.
- **병렬 정렬(Parallel Sort)**: 정렬할 데이터를 분할하여 여러 워커 스레드를 통해 정렬한 후 병합하는 과정을 병렬로 수행하여 정렬 응답 속도를 개선한다.

설정 방법

병렬 질의 실행은 시스템 파라미터와 SQL 힌트를 통해 제어할 수 있다.

- **parallelism** 파라미터를 2 이상으로 설정하면, 옵티마이저가 질의 수행시 병렬 질의 실행 여부를 판단할 수 있게 활성화된다.
- **PARALLEL (degree)** 힌트를 사용하여 질의별로 병렬 처리 수준을 명시적으로 지정할 수 있다. *degree* 는 사용할 워커 수이며 0 이상의 정수이다. **0** 또는 **1**로 지정하면 병렬 실행이 비활성화되고, **2 이상**으로 지정하면 힌트 값이 **parallelism** 파라미터 설정보다 우선한다. 시스템의 CPU 코어 수를 초과하는 값은 코어 수로 낮추어 적용된다.
- **max_parallel_workers** 파라미터는 서버 전체에서 동시에 실행 가능한 병렬 워커 스레드의 최대 개수를 설정한다(기본값: 100).
- **NO_PARALLEL_SCAN** 힌트는 해당 질의 블록의 모든 병렬 스캔(힙/리스트/인덱스)을 비활성화한다. **PARALLEL** 힌트와 같이 사용하는 경우에는 **NO_PARALLEL_SCAN** 이 우선 적용된다.
- 병렬 해시 조인과 부질의 병렬 실행은 각각 **NO_PARALLEL_HASH_JOIN**, **NO_PARALLEL_SUBQUERY** 힌트로 비활성화할 수 있다. 자세한 내용은 각 절을 참고한다.

i 참고

max_parallel_workers와 parallelism 파라미터는 기본값이 각각 100과 4로 설정되어 있어 별도 설정 없이도 병렬 질의를 사용할 수 있다. 단, 시스템의 CPU 코어 수가 2 이하인 경우에는 병렬 질의 실행이 전면 비활성화된다.

i 참고

병렬 실행에서는 워커들의 처리 완료 순서에 따라 결과 행의 순서가 실행할 때마다 달라질 수 있다. 결과 순서가 중요한 경우에는 ORDER BY 절을 명시해야 한다.

병렬 스캔

병렬 스캔(Parallel Scan)은 단일 스캔 입력을 여러 워커 스레드가 분할하여 동시에 처리하는 기능이다. CUBRID는 다음 세 가지 입력에 대해 병렬 스캔을 지원하며, 모두 동일한 병렬 실행 프레임워크를 공유한다.

- **힙(Heap)**: 테이블의 힙 페이지 — 섹터 단위로 워커들에게 사전 분할
- **리스트(List)**: 임시 결과 리스트 파일 — 섹터 단위로 워커들에게 사전 분할
- **인덱스(Index)**: B+트리 리프 페이지 — 공유 커서를 통해 워커들이 협력적으로 좌→우(또는 우→좌) 진행

각 워커 스레드는 할당된 영역을 독립적으로 스캔하면서 필터링 조건(predicate)을 평가하고, 처리한 결과는 결과 큐를 통해 메인 스레드에 전달되어 최종 결과로 통합된다.

i 참고

실제 병렬 처리 수준은 사용자가 설정한 상한값 내에서 처리량 규칙에 의해 자동으로 최적화된다. 자세한 내용은 **병렬 질의 처리량 규칙**을 참고한다.

공통 제약 조건

다음 조건 중 하나라도 해당되면 스캔 종류와 무관하게 병렬 스캔이 적용되지 않으며, 단일 스레드 방식으로 실행된다.

- 동시성 처리를 지원하지 않는 구문이 포함된 경우
 - 저장프로시저(JavaSP, PL/CSQL), Serial 사용시
 - 세션 변수를 참조시
 - Recursive CTE 또는 Connect By 구문 사용시
 - CUBRID 오브젝트 DBMS 전용 기능 사용시
- 배타적 잠금(X-LOCK) 획득이 필요한 경우
 - SELECT ... FOR UPDATE 구문
 - incr() 함수 사용시
 - update, delete, merge 구문
- 중첩 루프 조인(nested loop join)에서 드라이빙(첫번째) 테이블이 아닌 테이블의 스캔인 경우. 중첩 루프 조인의 병렬 처리는 드라이빙 테이블의 병렬 스캔을 통해 이루어지므로, 드라이빙 테이블이 병렬 스캔 조건을 만족하지 않으면 뒤따르는 테이블이 크더라도 조인 전체가 단일 스레드로 실행된다. 이 경우 큰 테이블이 드라이빙되도록 조인 순서를 조정하거나(**ORDERED** 힌트), 해시 조인을 힌트로 강제하면(**USE_HASH** 힌트) 병렬 실행을 유도할 수 있다.
- 상관 부질의(correlated subquery) 내부의 스캔인 경우. 비상관(uncorrelated) 부질의 내부의 스캔은 병렬 실행이 가능하다.
- 소트 머지 조인의 외부/내부 입력으로 사용되는 스캔 (모든 스캔 종류)
- 시스템 카탈로그 클래스(**db_class**, **db_serial**, **db_partition** 등)나 MVCC 대상이 아닌 클래스(루트 클래스 등)를 스캔하는 경우
- **NO_PARALLEL_SCAN** 힌트가 명시된 경우

스캔 종류별로 추가 제약이 있다. 아래 각 절을 참고한다.

```
-- 병렬 스캔이 적용되지 않는 예

-- 힌트로 비활성화 (모든 스캔 종류)
SELECT /** NO_PARALLEL_SCAN */ *
FROM large_table;

-- SELECT FOR UPDATE
SELECT /** PARALLEL(4) */ *
FROM large_table
```

```

WHERE id <= 10
FOR UPDATE;

-- 세션 변수 사용
SET @user_id = 123;
SELECT /** PARALLEL(4) */ *
FROM orders
WHERE cust_id = @user_id;

-- SERIAL 사용
SELECT /** PARALLEL(4) */ id, order_seq.NEXT_VALUE
FROM large_table
WHERE id <= 3;

```

병렬 힙 스캔

병렬 힙 스캔(Parallel Heap Scan)은 테이블의 힙 페이지를 섹터(sector) 단위로 워커들에게 정적으로 분배하여 동시 스캔하는 기능이다. 특히 선택도(selectivity)가 낮은 경우(일반적으로 0.05 이하) 단일 스레드 방식의 힙 스캔보다 응답 속도가 크게 향상될 수 있다.

힙 스캔에 한정되는 추가 제약은 없다. 위에서 설명한 **공통 제약**만 적용된다.

```

-- 병렬 힙 스캔 예
SELECT /** PARALLEL(4) */ id, category
FROM large_table
WHERE status = 'active';

-- 파티션 테이블의 병렬 힙 스캔
SELECT /** PARALLEL(4) */ *
FROM sales_partitioned
WHERE sale_date BETWEEN '2024-01-01' AND '2024-12-31';

-- INSERT SELECT 문 (대용량 데이터 복사)
INSERT INTO archive_orders
SELECT /** PARALLEL(4) */ *
FROM orders
WHERE amount < 100;

```

참고

파티션 테이블의 스캔은 파티션별로 병렬화가 결정되며, 활성화 조건(**병렬 질의 처리량 규칙**)도 개별 파티션의 페이지 수로 평가된다. 페이지 수가 활성화 조건에 미달하는 파티션은 단일 스레드로 스캔된다.

병렬 리스트 스캔

병렬 리스트 스캔(Parallel List Scan)은 부질의나 derived table 등 중간 단계에서 디스크 임시 파일(temp file)로 저장된 결과 리스트를 섹터(sector) 단위로 워커들에게 정적으로 분배하여 동시 스캔하는 기능이다. 분할 메커니즘 자체는 힙 스캔과 동일하며, 입력이 테이블 힙 대신 임시 파일이라는 점만 다르다. 상위 연산이 큰 임시 결과를 다시 스캔해야 할 때 효과적이다.

리스트 스캔 추가 제약

다음 조건 중 하나라도 해당되면 병렬 리스트 스캔이 적용되지 않으며, 단일 스레드 리스트 스캔으로 실행된다.

- 임시 리스트가 메모리 버퍼에만 존재하고 디스크 임시 파일로 저장되지 않은 경우 (분할할 섹터가 없어 작은 리스트는 자동으로 단일 스레드로 실행)
- 상위 연산이 결과를 한 행씩 받아가는 row-by-row 모드로 동작하는 경우 (mergeable list 와 BUILDVALUE 모두 적용 불가능한 형태의 질의 — [스캔 결과 수집 모드](#) 참고)
- ROWNUM 또는 LIMIT 조건절이 리스트 스캔에 걸려 있는 경우
- 소트 머지 조인의 보조 입력 트리(부질의/CTE 등) 안에 위치한 리스트 스캔

```
-- 병렬 리스트 스캔이 적용되는 전형적인 패턴
-- View Merging이 수행되지 않는 derived table의 임시 결과 리스트를 외부에서 다시 집계
SELECT /*+ PARALLEL(4) */ COUNT(*)
FROM (SELECT DISTINCT id, pad FROM large_table) t;
```

참고

단순 프로젝션만 있는 derived table은 **View Merging** 최적화에 의해 상위 질의로 머지(merge)되므로 임시 결과 리스트 자체가 만들어지지 않는다. 임시 결과 리스트는 DISTINCT, GROUP BY, UNION 등이 포함되어 **View Merging**을 수행할 수 없는 derived table에서 생성되며, **NO_MERGE** 힌트로 **View Merging**을 막아 강제로 생성할 수도 있다. 해시 조인의 입력 리스트를 상위에서 다시 스캔하는 경우도 대표적인 병렬 리스트 스캔 대상이다. **View Merging**에 대한 자세한 내용은 [View Merging 최적화](#)를 참고한다.

병렬 인덱스 스캔

병렬 인덱스 스캔(Parallel Index Scan)은 B+트리 인덱스의 리프 페이지 체인을 여러 워커가 공유 커서를 통해 협력적으로 진행하면서 읽는 기능이다. 인덱스 진입(루트→리프 수직 탐색)은 메인 스레드가 단일 스레드로 수행하며, 그 이후의 리프 순회와 OID 페치/필터 평가가 워커들에 의해 병렬로 수

행된다. 워커는 리프 한 페이지를 잡으면 그 안의 키들을 독립적으로 처리하고, 다음 리프를 얻기 위해서만 짧게 동기화한다.

인덱스 스캔 추가 제약

다음 조건 중 하나라도 해당되면 병렬 인덱스 스캔이 적용되지 않으며, 단일 스레드 인덱스 스캔으로 실행된다.

- 리프 순서·진입 방식에 의존하는 인덱스 최적화가 적용된 경우
 - ISS(Index Skip Scan)
 - ILS(Index Loose Scan)
 - KEYLIMIT 절
 - ORDERBY_SKIP / GROUPBY_SKIP 계열 최적화 (ORDER BY / GROUP BY를 인덱스 순서로 대체하는 최적화)
 - MIN/MAX 단일 키 조회 (min_max scan)
- ROWNUM 조건을 워커별로 다시 계산할 수 없는 형태로 사용한 경우, 또는 분석 함수(analytic function)의 SKIP SORT/LIMIT 최적화가 적용된 경우
- 상위 연산이 결과를 한 행씩 받아가는 row-by-row 모드로 동작하는 경우 (mergeable list 와 BUILDVALUE 모두 적용 불가능한 형태의 질의 — [스캔 결과 수집 모드](#) 참고)
- 소트 머지 조인의 보조 입력 트리(부질의/CTE 등) 안에 위치한 인덱스 스캔

```
-- 병렬 인덱스 스캔이 적용되는 전형적인 예
-- (넓은 범위 조건의 커버링 인덱스 스캔)
CREATE INDEX idx_large_id_pad ON large_table(id, pad);

SELECT /*+ PARALLEL(4) */ COUNT(pad)
FROM large_table
WHERE id BETWEEN 1 AND 900000 USING INDEX idx_large_id_pad;

-- 병렬 인덱스 스캔이 적용되지 않는 예
-- (KEYLIMIT 절 → 단일 스레드 인덱스 스캔)
SELECT /*+ PARALLEL(4) */ id, category
FROM large_table
WHERE id BETWEEN 1 AND 900000 USING INDEX idx_large_id_pad KEYLIMIT
100;
```

참고

병렬 인덱스 스캔은 메인 스레드의 인덱스 진입과 워커들의 리프 동기화 비용이 존재하기 때문에, 인덱스의 리프 페이지 수가 충분히 많을 때 효과가 크다. 작은 인덱스에서는 동기화 비용이

절감 효과를 상쇄할 수 있으며, 이를 막기 위해 처리량 규칙(**병렬 질의 처리량 규칙**)이 함께 적용된다.

i 참고

PARALLEL 힌트 없이 인덱스 스캔이 자동으로 병렬화되려면 옵티마이저 단계에서 다음 두 조건을 추가로 만족해야 한다.

- 모든 키 범위(key range) 조건의 선택도가 **히스토그램 통계**로부터 산출될 수 있어야 한다. 대상 테이블의 히스토그램 통계가 없으면 옵티마이저는 병렬 인덱스 스캔을 선택하지 않는다. 통계는 **UPDATE STATISTICS** 문으로 갱신할 수 있다.
- 선택도와 인덱스 페이지 수의 곱이 일정 기준(기본 32페이지) 이상이어야 한다.

PARALLEL 힌트를 지정하면 위 옵티마이저 조건은 우회하지만, 서버가 실측한 인덱스 페이지 수가 활성화 조건(**병렬 질의 처리량 규칙**)에 미달하면 단일 스레드로 실행된다.

성능 고려사항

병렬 스캔은 다음과 같은 경우에 성능 향상 효과가 크다.

- 대용량 입력(테이블/리스트/인덱스)을 스캔해야 하는 경우 (페이지 수가 많을수록 효과적)
- 힙/인덱스 스캔에서 선택도가 낮은 경우 (약 0.05 이하)
- CPU 코어가 충분히 사용 가능한 경우
- 디스크 I/O보다 CPU 처리가 병목인 경우

반면, 다음과 같은 경우에는 오히려 성능이 저하될 수 있다.

- 소량의 입력만 스캔하는 경우
- 인덱스 스캔이 단일 스레드로 충분히 효율적인 경우(예: 짧은 범위 점 조회)
- 시스템 리소스(CPU, 메모리)가 부족한 경우

병렬 질의 사용 시에는 **max_parallel_workers** 파라미터를 적절히 설정하여 시스템 리소스 경쟁을 방지해야 한다. 일반적으로 실제 물리 CPU 코어 수 수준으로 설정하는 것을 권장한다.

스캔 결과 수집 모드

병렬 스캔이 활성화된 경우, 워커가 처리한 결과를 메인 스레드가 수집하는 방식은 질의 형태에 따라 다음 세 가지 중 하나로 결정된다. 모드는 SQL 트레이스의 **gather** 항목으로 확인할 수 있다.

- **mergeable list**: 각 워커가 자신의 임시 결과 리스트를 만들고, 메인 스레드는 워커별 리스트를 병합 없이 그대로 출력에 사용한다. 워커 간 동기화 비용이 가장 적어 일반적으로 가장 빠르다.
- **buildvalue**: 워커들이 부분 집계값을 계산하여 메인 스레드에 전달하면, 메인 스레드가 최종 집계값을 결합하여 반환한다. 단순 집계 질의에 특화된 모드이다(**BUILDVALUE 최적화** 참고).
- **row-by-row**: 메인 스레드가 한 행씩 순서대로 받아 처리한다. 다른 두 모드를 적용할 수 없는 경우에 사용된다. 적용 가능 범위가 가장 넓지만 동기화 비용이 가장 크다.

참고

row-by-row 모드는 **힙 스캔에서만** 나타난다. 리스트 스캔과 인덱스 스캔은 row-by-row 모드가 필요한 질의 형태에서는 단일 스레드로 회귀하므로(위 추가 제약 참고), 트레이스의 **gather: row by row** 표기는 힙 스캔에서만 관찰된다.

mergeable list 가 적용되지 않는 조건

다음 조건 중 하나라도 해당되면 mergeable list 가 아닌 다른 모드(buildvalue 또는 row-by-row)로 처리된다.

- 대상 입력을 스캔하면서 평가할 수 없는 조건절(상위 단계로 미루어진 predicate)이 포함되는 경우
- select-list에 저장프로시저(JavaSP 또는 PL/CSQL)가 있는 경우
- ROWNUM 또는 LIMIT 조건을 워커별로 다시 계산할 수 없는 형태로 사용한 경우
- 스캔이 테이블의 컬럼을 직접 출력하지 않는 경우(상수만 출력하는 질의 등)
- result_cache가 활성화되어 있는 경우

참고

ORDER BY와 LIMIT을 함께 사용하는 상위 N개 추출 질의도 mergeable list 로 병렬 실행될 수 있다. 이때는 각 워커가 스캔 중에 자신의 상위 N개를 유지하며, 트레이스의 병렬 처리 상세 정보에 **topnsort: true** 가 표시된다. GROUP BY가 결합된 질의에서는 워커들이 부분 해시 집계를 수행하며, 트레이스의 **GROUPBY** 항목에 **hash: partial** 이 표시된다.

BUILDVALUE 최적화

GROUP BY 없이 집계 함수를 계산하는 질의에서 사용된 집계 함수가 모두 지원 목록에 포함되면, 병렬 스캔은 **BUILDVALUE 최적화**를 적용한다. 이 모드에서는 각 워커가 자신이 스캔한 범위의 부분 집계값을 계산하여 메인 스레드에 전달하고, 메인 스레드는 부분 집계들을 결합하여 최종값을 만든다. 워커 간 데이터 전달량이 가장 적어 집계 질의에서 가장 빠른 동작 모드이다.

지원 집계 함수

다음 집계 함수를 사용한 질의에 BUILDVALUE 최적화가 적용된다. SELECT 리스트에서 집계 함수들을 산술식 등으로 조합하는 것도 가능하다.

- COUNT(*), COUNT(column), COUNT(DISTINCT column)
- MIN, MAX
- SUM, AVG
- STDDEV, STDDEV_POP, STDDEV_SAMP
- VARIANCE, VAR_POP, VAR_SAMP
- BIT_AND, BIT_OR, BIT_XOR
- GROUP_CONCAT, JSON_ARRAYAGG, JSON_OBJECTAGG
- MEDIAN, PERCENTILE_CONT, PERCENTILE_DISC
- CUME_DIST, PERCENT_RANK

적용 조건

위 집계 함수 사용에 더해 다음 조건을 모두 만족해야 한다.

- GROUP BY 절이 없을 것
- ROWNUM을 사용하지 않을 것

적용 범위

BUILDVALUE 최적화는 스캔 종류와 무관하게 적용 가능하다.

- 병렬 힙 스캔
- 병렬 리스트 스캔
- 병렬 인덱스 스캔

예제

```
-- COUNT 계열
SELECT /** PARALLEL(8) */ COUNT(*)
FROM large_table
WHERE status = 'active';

SELECT /** PARALLEL(8) */ COUNT(DISTINCT customer_id)
FROM orders;

-- 산술 집계 (HEAP/LIST/INDEX 모두 가능)
SELECT /** PARALLEL(8) */ SUM(amount), AVG(amount), MAX(amount)
FROM orders
WHERE order_date > '2024-01-01';

-- 분산/표준편차
SELECT /** PARALLEL(8) */ STDDEV(price), VARIANCE(price)
FROM products;

-- UPDATE STATISTICS도 내부적으로 BUILDVALUE 최적화의 혜택을 받는다
UPDATE STATISTICS ON large_table WITH FULLSCAN;
```

참고

미지원 집계 함수가 하나라도 포함되거나 GROUP BY가 결합되면 BUILDVALUE 최적화는 적용되지 않으며, mergeable list 또는 row-by-row 모드로 처리된다.

스캔 SQL 트레이스

병렬 스캔이 수행되면 **SQL 트레이스** 결과에 병렬 처리 상세 정보가 추가로 출력된다.

```
csql> ;trace on

SELECT /** PARALLEL(4) RECOMPILE */ id, category
FROM large_table
WHERE status = 'active';
```

```
Trace Statistics:
  SELECT (time: 110, fetch: 19717, fetch_time: 15, ioread: 2101)
    SCAN (table: dba.large_table), (heap time: 110, fetch: 19714,
ioread: 2100, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 108..110, readrows:
248704..252416, rows: 248704..252416, gather: mergeable list)
```


병렬 스캔의 트레이스 출력 항목은 다음과 같다.

- **parallel workers**: 사용된 워커 스레드의 수
- **heap time / temp time / index time**: 각 워커의 스캔 소요 시간 범위 (최소..최대, 밀리초). 항목 이름은 힙 스캔이면 **heap time**, 리스트 스캔이면 **temp time**(입력이 임시 리스트 파일), 인덱스 스캔이면 **index time**이다.
- **readrows**: 각 워커가 읽은 행 수 범위 (최소..최대)
- **rows**: 각 워커가 반환한 행 수 범위 (최소..최대)
- **gather**: 결과 수집 방식
 - **mergeable list**: 워커별 리스트를 병합 없이 직접 사용
 - **buildvalue**: 워커별 부분 집계를 결합 (구 `count` 표시를 대체)
 - **row by row**: 한 건씩 수집하여 병합 (힙 스캔에서만 나타남)

gather 항목에 **mergeable list** 또는 **buildvalue** 가 표시된 경우, 동기화 비용이 적은 최적 경로로 실행되었음을 의미한다.

참고

병렬 워커들의 시간과 행 수가 범위(최소..최대)로 표시되며, 이상적으로는 모든 워커가 비슷한 양의 작업을 수행해야 한다. 범위가 크게 벌어진다면 데이터 분포나 시스템 리소스 경합 문제를 의심해볼 수 있다.

BUILDVALUE 최적화 추적 정보 예제

BUILDVALUE 최적화가 적용되면 **gather: buildvalue** 가 표시된다.

```
csql> ;trace on
```

```
SELECT /*+ PARALLEL(4) RECOMPILE */ COUNT(*)
FROM large_table
WHERE status = 'active';
```

Trace Statistics:

```
  SELECT (time: 86, fetch: 17249, fetch_time: 93, ioread: 13377)
    SCAN (table: dba.large_table), (heap time: 86, fetch: 17246,
ioread: 13377, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 81..86, readrows:
248704..252416, rows: 248704..252416, gather: buildvalue)
```

병렬 인덱스 스캔 트레이스 예제

병렬 인덱스 스캔의 병렬 처리 상세 정보에는 **index time** 과 함께 워커별 **readkeys**(읽은 키 수)와 **filteredkeys**(키 필터 통과 수) 범위가 출력된다. 커버링 인덱스 스캔이면 **covered: true** 가 표시된다.

```
csql> ;trace on
```

```
SELECT /** PARALLEL(4) RECOMPILE */ COUNT(pad)
FROM large_table
WHERE id BETWEEN 1 AND 900000 USING INDEX idx_large_id_pad;
```

Trace Statistics:

```
SELECT (time: 136, fetch: 13247, fetch_time: 77, ioread: 13239)
      SCAN (index: dba.large_table.idx_large_id_pad), (btree time:
136, fetch: 13244, ioread: 13239, readkeys: 900003, filteredkeys:
900000, rows: 900000, covered: true)
      (parallel workers: 4, index time: 136..136, readkeys:
224557..225285, filteredkeys: 224556..225284, rows: 224556..225284,
covered: true, gather: buildvalue)
```

커버링이 아닌 인덱스 스캔에서는 병렬 처리 상세 정보 뒤에 데이터 페이지 조회 통계인 **(lookup time: 최소..최대, rows: 최소..최대)** 가 덧붙는다.

```
csql> ;trace on
```

```
SELECT /** PARALLEL(4) RECOMPILE */ category
FROM large_table
WHERE id BETWEEN 1 AND 900000 USING INDEX idx_large_id_pad;
```

Trace Statistics:

```
SELECT (time: 330, fetch: 915916, fetch_time: 244, ioread: 129)
      SCAN (index: dba.large_table.idx_large_id_pad), (btree time:
330, fetch: 915913, ioread: 129, readkeys: 900002, filteredkeys:
900000, rows: 900000) (lookup time: 0, rows: 900000)
      (parallel workers: 4, index time: 330..330, readkeys:
223469..227732, filteredkeys: 223468..227732, rows: 223468..227732,
gather: mergeable list) (lookup time: 0..0, rows: 223468..227732)
```

부질의 병렬 실행

부질의 병렬 실행(Parallel Subquery Execution)은 서로 독립적으로 실행 가능한 부질의(subquery)들을 여러 워커 스레드를 사용하여 동시에 실행함으로써 쿼리 성능을 향상시키는 기능이다.

부질의 실행 개요

부질의는 다른 부질의와 독립적으로 실행될 수 있으므로, 여러 부질의가 있는 경우 병렬로 실행하여 전체 쿼리 응답 시간을 단축할 수 있다. 각 부질의는 독립적인 워커 스레드에서 실행되며, 모든 부질의의 실행이 완료되면 결과가 병합되어 최종 결과를 생성한다.

`parallelism` 파라미터가 2 이상으로 설정되어 있거나, **PARALLEL** (*degree*) 힌트를 사용하여 병렬 정도를 2 이상으로 지정하면 부질의의 병렬 실행이 가능하다.

NO_PARALLEL_SUBQUERY 힌트를 사용하면 부질의의 병렬 실행을 비활성화할 수 있다. **PARALLEL** 힌트와 같이 사용하는 경우에는 **NO_PARALLEL_SUBQUERY** 힌트가 우선된다.

실행 조건

다음 조건을 모두 만족할 때 부질의의 병렬 실행이 가능하다:

- `max_parallel_workers` 파라미터가 2 이상으로 설정되어 있고, 사용 가능한 워커 스레드가 있는 경우
- `parallelism` 파라미터가 2 이상으로 설정되어 있거나, **PARALLEL** (2) 이상의 힌트가 명시된 경우
- 부질의가 최상위 레벨 쿼리(top-level XASL)에 직접 연결되는 있는 경우
- **NO_PARALLEL_SUBQUERY** 힌트가 사용되지 않은 경우

```
-- parallelism 파라미터 설정 (cubrid.conf)
-- parallelism=4

-- 부질의 병렬 실행 예제
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);

-- 힌트를 사용한 예제
SELECT /*+ PARALLEL(4) */ *
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);
```

적용되지 않는 경우

다음 조건 중 하나라도 해당되면 부질의 병렬 실행이 적용되지 않는다:

- 부질의가 최상위 레벨 쿼리에 직접 연결되지 않은 경우 (중첩된 부질의 내부의 부질의 등)
- CTE(Common Table Expression)의 recursive 부분이 존재하거나 CTE 간 참조가 있는 경우
- derived table(인라인 뷰) 등에 의해 부질의 간 참조가 존재하는 경우
- Object DBMS 기능을 사용하는 경우 (path expression 등)
- JSON_TABLE이나 SET 타입 테이블의 스캔이 포함된 경우
- 부질의 조건절에 저장 프로시저가 포함된 경우
- 상관 부질의인 경우

```
-- 병렬 실행이 적용되지 않는 예

-- NO_PARALLEL_SUBQUERY 힌트 사용
SELECT /*+ NO_PARALLEL_SUBQUERY */ *
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);

-- CTE 간 참조가 있는 경우
WITH cte1 AS (
    SELECT * FROM table1
),
cte2 AS (
    SELECT * FROM cte1 WHERE id > 100 -- cte1 참조
)
SELECT * FROM cte2;

-- JSON_TABLE 사용
SELECT *
FROM orders,
JSON_TABLE(json_column, '$[*]' COLUMNS(id INT PATH '$.id')) AS jt
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);

-- 저장 프로시저가 조건절에 포함된 경우
SELECT *
```

```
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE check_region_sp(region)
    = 1
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);
```

부질의 성능 고려사항

부질의의 병렬 실행은 다음과 같은 경우에 성능 향상 효과가 크다:

- 복수의 독립적인 부질의가 존재하는 경우
- 각 부질의의 실행 시간이 충분히 긴 경우
- CPU 코어가 충분히 사용 가능한 경우

반면, 다음과 같은 경우에는 오히려 성능이 저하될 수 있다:

- 부질의의 실행 시간이 매우 짧은 경우 (병렬 처리 오버헤드가 더 클 수 있음)
- 부질의가 하나만 존재하는 경우
- 시스템 리소스(CPU, 메모리)가 부족한 경우
- `max_parallel_workers` 설정이 부적절한 경우

관련 파라미터

부질의의 병렬 실행을 효과적으로 사용하려면 다음 파라미터들을 적절히 설정해야 한다:

- `max_parallel_workers`: 서버 전역의 최대 병렬 워커 수
- `parallelism`: 기본 병렬 처리 정도

```
-- cubrid.conf 설정 예제
max_parallel_workers=16
parallelism=4
```

부질의 추적 정보

부질의의 병렬 실행이 수행되면 **SQL 트레이스** 결과에 병렬 처리 상세 정보가 추가로 출력된다.

```
csql> ;trace on
```

```
SELECT /*+ PARALLEL(4) RECOMPILE */ *
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);
```

```
Trace Statistics:
  SELECT (time: 1710, fetch: 51619, fetch_time: 5, ioread: 0)
    SCAN (temp time: 0, fetch: 0, ioread: 0, readrows: 125,
rows: 125)
      SCAN (table: dba.orders), (heap time: 1677, fetch:
51500, ioread: 0, readrows: 12500000, rows: 25000)
        SCAN (hash temp(m), build time: 0, time: 0, fetch:
0, ioread: 0, readrows: 350, rows: 17)
          SUBQUERY (uncorrelated)
            (parallel workers: 2, time: 0, fetch: 9, fetch_time: 0,
ioread: 0)
              SELECT (time: 0, fetch: 5, fetch_time: 0, ioread: 0)
                SCAN (table: dba.customers), (heap time: 0, fetch:
4, ioread: 0, readrows: 1000, rows: 333)
                  ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
                SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
                  SCAN (table: dba.products), (heap time: 0, fetch: 3,
ioread: 0, readrows: 500, rows: 125)
                    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```

부질의 병렬 실행 SQL 트레이스 출력 항목에 대한 설명은 다음과 같다:

- **SUBQUERY (uncorrelated)**: 부질의 실행 표시
- **parallel workers**: 병렬 실행에 사용된 워커 스레드의 수
- **time**: 병렬 실행에 소요된 시간 (밀리초)
- 각 부질의는 독립적인 SELECT로 표시되며, 각각의 실행 통계가 출력된다

위 예제에서는 2개의 부질의(customers 테이블 조회, products 테이블 조회)가 2개의 워커 스레드를 사용하여 병렬로 실행되었다.

병렬 해시 조인

병렬 해시 조인(Parallel Hash Join)은 해시 조인의 빌드(build)·프로브(probe) 단계를 여러 워커 스레드로 병렬화하여 조인 응답 속도를 개선하는 기능이다. 해시 조인은 두 입력을 임시 결과 리스트로 적재한 뒤 한쪽 입력으로 해시 테이블을 만들고(빌드) 다른 쪽 입력으로 해시 테이블을 탐색(프로브)하는데, 병렬 해시 조인은 입력 크기에 따라 다음 두 형태로 수행된다.

- **파티션 병렬**: 두 입력을 조인 키 기준으로 여러 파티션으로 분할(SPLIT)한 뒤, 워커들이 파티션별 빌드와 프로브를 동시에 수행한다. 빌드 입력이 커서 파티셔닝이 필요한 경우에 사용된다.
- **프로브 병렬**: 빌드 입력이 작아 해시 테이블을 메모리에 한 번에 만들 수 있는 경우, 빌드는 단일 스레드로 수행하고 프로브 입력만 워커들이 분할하여 동시에 탐색한다.

활성화 조건

- 두 입력 리스트 중 큰 쪽의 페이지 수가 활성화 조건(**병렬 질의 처리량 규칙**)을 만족해야 한다. 미달이면 **PARALLEL** 힌트가 있어도 단일 스레드 해시 조인으로 실행된다.
- **NO_PARALLEL_HASH_JOIN** 힌트가 지정되면 병렬화되지 않는다. 이때 해시 조인 자체는 유지되고 병렬화만 비활성화된다. 자세한 내용은 **NO_PARALLEL_HASH_JOIN**을 참고한다.

해시 조인 트레이스

해시 조인이 수행되면 **SQL 트레이스** 결과에 **HASHJOIN** 트리가 출력된다. 트리는 파티션 분할(**SPLIT**), 빌드(**BUILD**), 프로브(**PROBE**) 단계별 통계를 보여주며, 이 트리 구조 자체는 병렬 여부와 무관하게 해시 조인이 수행되면 항상 출력된다. 병렬 해시 조인이 적용되면 **HASHJOIN** 항목에 **parallel workers**가 추가되고, 파티션별 병렬 수행 구간이 **PARALLEL** 항목으로 표시된다.

```
csql> ;trace on
```

```
SELECT /*+ RECOMPILE PARALLEL(4) USE_HASH */ COUNT(*)
FROM orders o JOIN large_table t ON o.order_id = t.id;
```

Trace Statistics:

```
  SELECT (time: 915, fetch: 76794, fetch_time: 134, ioread: 497)
    SCAN (temp time: 28, fetch: 2465, ioread: 142, readrows:
1000000, rows: 1000000)
      (parallel workers: 4, temp time: 24..28, readrows:
181482..327609, rows: 181482..327609, gather: buildvalue)
    HASHJOIN (time: 886, fetch: 74305, fetch_time: 133, ioread:
349, parallel workers: 4)
      SPLIT (time: 198, fetch: 16086, ioread: 338, partitions: 8)
      PARALLEL (time: 326, fetch: 21073, ioread: 2)
        BUILD (time: 83..96, fetch: 2437, ioread: 0, rows:
1000000, method: hybrid)
        PROBE (time: 180..226, fetch: 18604, ioread: 1, readrows:
1000000, readkeys: 1000000, rows: 1000000)
```

```

SUBQUERY (uncorrelated)
  (parallel workers: 2, time: 357, fetch: 37116,
fetch_time: 86, ioread: 2)
  SELECT (time: 351, fetch: 14869, fetch_time: 40, ioread: 2)
    SCAN (table: dba.orders), (heap time: 348, fetch: 14866,
ioread: 2, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 237..348,
readrows: 245632..252096, rows: 245632..252096, gather: mergeable
list)
    SELECT (time: 356, fetch: 22247, fetch_time: 46, ioread: 0)
      SCAN (table: dba.large_table), (heap time: 356, fetch:
22244, ioread: 0, readrows: 1000000, rows: 1000000)
        (parallel workers: 4, heap time: 300..354,
readrows: 248704..252416, rows: 248704..252416, gather: mergeable
list)

```

해시 조인 트레이스의 병렬 관련 출력 항목은 다음과 같다.

- **HASHJOIN**의 **parallel workers**: 해시 조인에 사용된 워커 스레드의 수. 이 항목이 없으면 해시 조인이 단일 스레드로 수행된 것이다.
- **SPLIT**: 두 입력을 조인 키 기준으로 파티션으로 분할하는 단계. **partitions** 는 생성된 파티션 개수이다.
- **PARALLEL**: 워커들이 파티션별 빌드·프로브를 병렬로 수행한 구간. 하위 **BUILD** 와 **PROBE** 의 **time** 이 워커별 범위(최소..최대)로 표시된다.
- **SUBQUERY (uncorrelated)**: 조인의 두 입력을 임시 결과 리스트로 적재하는 단계. 두 입력의 적재 자체도 부질의 병렬 실행과 병렬 스캔의 적용 대상이다.

다음은 **NO_PARALLEL_HASH_JOIN** 힌트로 병렬화를 비활성화한 예이다. **HASHJOIN** 항목에 **parallel workers** 가 없고 **PARALLEL** 항목 없이 **BUILD** 와 **PROBE** 가 단일 스레드로 수행되지만, 조인 입력을 적재하는 스캔들은 독립적으로 병렬 수행될 수 있다.

```
csql> ;trace on
```

```

SELECT /** RECOMPILE PARALLEL(4) USE_HASH NO_PARALLEL_HASH_JOIN */
COUNT(*)
FROM orders o JOIN large_table t ON o.order_id = t.id;

```

Trace Statistics:

```

SELECT (time: 1716, fetch: 74720, fetch_time: 398, ioread: 16587)
  SCAN (temp time: 52, fetch: 2465, ioread: 1346, readrows:
1000000, rows: 1000000)
    (parallel workers: 4, temp time: 46..52, readrows:
223584..302385, rows: 223584..302385, gather: buildvalue)
  HASHJOIN (time: 1663, fetch: 72231, fetch_time: 355, ioread:
15237)

```



```

        SPLIT (time: 390, fetch: 15843, ioread: 1029, partitions: 8)
        BUILD (time: 237, fetch: 2431, ioread: 2323, rows: 1000000,
method: hybrid)
        PROBE (time: 562, fetch: 16847, ioread: 2136, readrows:
1000000, readkeys: 1000000, rows: 1000000)
        SUBQUERY (uncorrelated)
            (parallel workers: 2, time: 439, fetch: 37048,
fetch_time: 222, ioread: 9733)
            SELECT (time: 360, fetch: 14873, fetch_time: 62, ioread:
2989)
                SCAN (table: dba.orders), (heap time: 360, fetch: 14870,
ioread: 2987, readrows: 1000000, rows: 1000000)
                    (parallel workers: 4, heap time: 165..354,
readrows: 245632..252096, rows: 245632..252096, gather: mergeable
list)
                SELECT (time: 433, fetch: 22175, fetch_time: 160, ioread:
6744)
                    SCAN (table: dba.large_table), (heap time: 433, fetch:
22172, ioread: 6742, readrows: 1000000, rows: 1000000)
                        (parallel workers: 4, heap time: 215..433,
readrows: 248704..252416, rows: 248704..252416, gather: mergeable
list)

```

다음은 빌드 입력이 작아 **프로브 병렬** 형태로 수행된 예이다. **BUILD** 는 단일 스레드로 수행되고 (**method: memory**), **PROBE** 항목 아래에 워커별 범위를 보여주는 병렬 처리 상세 정보가 출력된다.

```
csql> ;trace on
```

```

SELECT /** RECOMPILE PARALLEL(4) USE_HASH */ COUNT(*)
FROM large_table t JOIN small_table s ON t.category = s.id;

```

Trace Statistics:

```

    SELECT (time: 482, fetch: 32028, fetch_time: 76, ioread: 2290)
        SCAN (temp time: 23, fetch: 2441, ioread: 64, readrows: 990000,
rows: 990000)
            (parallel workers: 4, temp time: 15..19, readrows:
197807..352823, rows: 197807..352823, gather: buildvalue)
        HASHJOIN (time: 458, fetch: 29575, fetch_time: 72, ioread:
2226)
            BUILD (time: 0, fetch: 0, ioread: 0, rows: 100, method:
memory)
                PROBE (time: 123, fetch: 14720, ioread: 0, readrows:
1000000, readkeys: 990000, rows: 990000)
                    (parallel workers: 4, time: 113..119, readrows:
197015..312576, readkeys: 195044..309450, rows: 195044..309450)
                SUBQUERY (uncorrelated)
                    (parallel workers: 2, time: 334, fetch: 22199,
fetch_time: 64, ioread: 2226)
                SELECT (time: 334, fetch: 22195, fetch_time: 64, ioread:

```

```

2223)
      SCAN (table: dba.large_table), (heap time: 333, fetch:
22192, ioread: 2221, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 288..332,
readrows: 248704..252416, rows: 248704..252416, gather: mergeable
list)
      SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 3)
      SCAN (table: dba.small_table), (heap time: 0, fetch: 2,
ioread: 1, readrows: 100, rows: 100)

```

병렬 정렬

병렬 정렬(Parallel Sort)은 정렬 입력을 여러 워커 스레드에 분배하여 각자 정렬하게 한 뒤 결과를 병합함으로써 정렬 응답 속도를 개선하는 기능이다. 다음 정렬 연산이 병렬 정렬의 대상이다.

- ORDER BY 절의 파일 정렬(filesort)
- 분석 함수(analytic function)의 파티션·정렬 처리
- 해시 집계를 적용할 수 없는 GROUP BY의 정렬 (예: DISTINCT 집계 함수가 포함된 경우)
- DISTINCT 중복 제거를 위한 내부 정렬

활성화 조건

- 정렬 입력의 페이지 수가 활성화 조건(**병렬 질의 처리량 규칙**)을 만족해야 한다.
- 입력의 행 수가 병렬 처리 수준 이하이면 단일 스레드로 정렬된다.
- ORDER BY와 LIMIT을 함께 사용하는 정렬은 입력의 행 수가 500행 미만이면 단일 스레드로 정렬된다.

참고

ORDER BY와 LIMIT을 함께 사용하는 상위 N개 추출 질의는 별도의 병렬 정렬 단계를 거치지 않고, 병렬 스캔 워커들이 스캔 중에 각자 상위 N개를 유지하는 방식으로 병렬화된다. 이때 트레이스에서 스캔의 병렬 처리 상세 정보에 **topnsort: true**가 표시되고, **ORDERBY** 단계는 최종 병합만 수행한다.

참고

GROUP BY 질의에서 해시 집계가 가능한 형태이면 정렬을 병렬화하는 대신 워커들이 부분 해시 집계를 수행하며, 트레이스의 **GROUPBY** 항목에 **hash: partial**이 표시된다.

NO_HASH_AGGREGATE 힌트로 해시 집계를 비활성화한 경우에는 GROUP BY 정렬이 병렬화되지 않는다.

i 참고

병렬 정렬은 워커별로 임시 정렬 공간을 사용하므로 단일 스레드 정렬보다 임시 볼륨 사용량이 늘어날 수 있다.

정렬 트레이스

병렬 정렬이 수행되면 해당 정렬 항목(**ORDERBY**, **GROUPBY**, **ANALYTIC**) 아래에 워커별 소요 시간, 페이지 수, I/O 읽기 수의 범위를 보여주는 병렬 처리 상세 정보가 추가로 출력된다.

```
csql> ;trace on
```

```
SELECT /*+ PARALLEL(4) RECOMPILE */ id, category, pad
FROM large_table
ORDER BY category;
```

Trace Statistics:

```
SELECT (time: 3563, fetch: 49462, fetch_time: 235, ioread: 14)
SCAN (table: dba.large_table), (heap time: 628, fetch: 49418,
ioread: 10, readrows: 1000000, rows: 1000000)
(parallel workers: 4, heap time: 497..628, readrows:
248704..252416, rows: 248704..252416, gather: mergeable list)
ORDERBY (time: 2934, sort: true, page: 0, ioread: 4)
(parallel workers: 4, time: 2732..2916, page:
6440..11364, ioread: 46952..58793)
```

병렬 정렬의 트레이스 출력 항목은 다음과 같다.

- **parallel workers**: 정렬에 사용된 워커 스레드의 수
- **time**: 각 워커의 정렬 소요 시간 범위 (최소..최대, 밀리초)
- **page, ioread**: 각 워커가 정렬 중 사용한 페이지 수와 I/O 읽기 수의 범위 (최소..최대)

다음은 분석 함수의 정렬이 병렬화된 예이다.

```
csql> ;trace on
```

```
SELECT /*+ PARALLEL(4) RECOMPILE */ order_id, amount,
```

```
SUM(amount) OVER (PARTITION BY cust_id ORDER BY order_id)
FROM orders;
```

```
Trace Statistics:
  SELECT (time: 1458, fetch: 1059149, fetch_time: 353, ioread: 50971)
    SCAN (table: dba.orders), (heap time: 139, fetch: 19880, ioread:
9910, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 135..139, readrows:
245632..252096, rows: 245632..252096, gather: mergeable list)
        ANALYTIC #1 (time: 1317, sort: true, page: 10561, ioread: 33612,
rows: 1000000)
          (parallel workers: 4, time: 64..70, page: 3656..4066,
ioread: 4..5)
```

다음은 DISTINCT 집계 함수 때문에 해시 집계를 적용할 수 없어 GROUP BY 정렬이 병렬화된 예이다.

```
csql> ;trace on
```

```
SELECT /*+ PARALLEL(4) RECOMPILE */ cust_id, COUNT(DISTINCT amount)
FROM orders
GROUP BY cust_id;
```

```
Trace Statistics:
  SELECT (time: 848, fetch: 984969, fetch_time: 114, ioread: 1742)
    SCAN (table: dba.orders), (heap time: 99, fetch: 14846, ioread:
34, readrows: 1000000, rows: 1000000)
      (parallel workers: 4, heap time: 96..99, readrows:
245632..252096, rows: 245632..252096, gather: mergeable list)
        GROUPBY (time: 749, hash: false, sort: true, page: 129, ioread:
3, readrows: 1000000, rows: 50000)
          (parallel workers: 4, time: 49..57, page: 2470..2759,
ioread: 4..90)
```

병렬 질의 처리량 규칙

처리량 규칙 개요

병렬 질의 실행은 질의 응답 시간을 획기적으로 단축하지만, 동시에 서버 자원(CPU, 메모리, I/O 등)을 많이 소모한다. 소수의 쿼리가 병렬 실행으로 서버 자원을 과도하게 선점하면 다른 다수의 쿼리 성능이 저하될 수 있다. 이를 방지하기 위해 CUBRID는 병렬 실행의 효과가 큰 쿼리만을 선별하기 위한 처리량 규칙을 적용한다.

각 병렬 연산의 실제 병렬 처리 수준은 다음 요인에 따라 결정된다:

- 테이블/리스트/인덱스 크기 등의 처리량 규칙
- **PARALLEL** 힌트로 명시적으로 지정된 값
- `parallelism` 파라미터로 설정된 상한값
- `max_parallel_workers` 파라미터로 설정된 전역 워커 풀 크기

처리량 규칙으로 계산된 병렬 처리 수준은 `parallelism` 파라미터 값을 초과할 수 없다. 힌트로 지정된 병렬 처리 수준은 `parallelism` 파라미터 값을 초과할 수 있지만 최대값(32 또는 시스템 코어 수 중 작은 값)은 초과할 수 없다.

스캔 처리량 규칙

병렬 힙 스캔과 병렬 리스트 스캔의 병렬 처리 수준은 스캔 대상의 페이지 수(힙 스캔은 테이블 힙 페이지 수, 리스트 스캔은 임시 리스트 페이지 수)에 따라 동일한 규칙으로 결정된다. 인덱스 스캔은 별도 규칙을 따른다(아래 참고).

활성화 조건

- 스캔 대상의 페이지 수가 2,048개 이상일 때 활성화된다 (약 32MB, `db_page_size`가 16K일 때)
- 이 조건을 만족하지 않으면 **PARALLEL** 힌트가 있어도 병렬 스캔이 활성화되지 않는다

처리 수준 결정

병렬 처리 수준은 페이지 수에 따라 다음과 같이 결정된다.

페이지 수	처리량	처리량 규칙 계산값
2,048	32 MB	2
4,096	64 MB	3
8,192	128 MB	4
16,384	256 MB	5
32,768	512 MB	6

페이지 수	처리량	처리량 규칙 계산값
65,536	1.0 GB	7
131,072	2.0 GB	8
262,144	4.0 GB	9
524,288	8.0 GB	10
1,048,576	16.0 GB	11
2,097,152	32.0 GB	12
4,194,304	64.0 GB	13
8,388,608	128.0 GB	14

페이지 수가 2,048개를 시작으로, 이전에 증가한 기준 페이지 수의 2배가 될 때마다 처리량 규칙으로 계산된 병렬 처리 수준이 1씩 증가한다.

처리량 규칙에 따라 결정되는 병렬 처리 수준은 **parallelism** 파라미터 값을 초과할 수 없다:

· **MIN (처리량 규칙 계산값, parallelism 파라미터 값)**

예를 들어, parallelism=4 (기본값)로 설정된 경우:

- 페이지 수 2,048개 → 처리량 규칙 계산값 2 → $\text{MIN}(2, 4) = 2$ 적용
- 페이지 수 65,536개 → 처리량 규칙 계산값 7 → $\text{MIN}(7, 4) = 4$ 적용 (parallelism 초과 불가)

참고

PARALLEL 힌트로 병렬 수준을 명시적으로 지정한 경우에도, 활성화 조건(페이지 수 2,048 이상)은 동일하게 적용된다. 활성화된 이후의 병렬 수준 결정에서는 힌트 값이 우선한다.

인덱스 스캔의 처리량 규칙

병렬 인덱스 스캔의 자동 병렬 수준은 옵티마이저가 **선택도 * 인덱스 페이지 수** 를 기준으로 계산한다. 이 값이 32 이상일 때 후보가 되며, 32에서 2배가 될 때마다 병렬 처리 수준이 2에서 1씩 증가하고 **parallelism** 파라미터 값을 초과할 수 없다. **PARALLEL** 힌트를 지정하면 이 옵티마이저 계산은 우회하지만, 서버가 실측한 인덱스 페이지 수가 2,048 이상이어야 한다는 활성화 조건은 동일하게 적용된다.

예제

```
-- 테이블 생성 및 데이터 삽입
CREATE TABLE large_table (id INT PRIMARY KEY, category INT, status
VARCHAR(10), pad VARCHAR(200));

INSERT INTO large_table
SELECT ROWNUM, MOD(ROWNUM, 100),
       CASE WHEN MOD(ROWNUM, 2) = 0 THEN 'active' ELSE 'closed' END,
       LPAD('x', 200, 'x')
FROM db_class a, db_class b, db_class c, db_class d
LIMIT 1000000;

UPDATE STATISTICS ON large_table WITH FULLSCAN;

-- 테이블 통계 확인 (SHOW HEAP CAPACITY OF large_table)
-- Num_pages: 17245 (약 269MB, db_page_size가 16K일 때)
-- Num_recs: 1000000

-- parallelism 파라미터가 4로 설정된 경우
-- 페이지 수 17245 → 처리량 규칙 계산값 5 → MIN(5, 4) = 4 적용
SELECT COUNT(*) FROM large_table WHERE status = 'active';

-- 힌트로 명시적 지정 (활성화 조건 충족 시 힌트 값이 우선)
SELECT /*+ PARALLEL(8) */ COUNT(*) FROM large_table WHERE status =
'active';
```

해시 조인 처리량 규칙

병렬 해시 조인은 두 입력 리스트 중 **큰 쪽의 페이지 수**를 기준으로 스캔과 같은 형태의 규칙(활성화 조건 2,048페이지, 페이지 수가 2배가 될 때마다 병렬 처리 수준 1 증가, **parallelism** 상한)에 따라 병렬 처리 수준이 계산된다.

- 계산된 병렬 처리 수준은 파티션 개수를 초과할 수 없다.
- 활성화 조건 미달이면 **PARALLEL** 힌트가 있어도 단일 스레드 해시 조인으로 실행된다.

정렬 처리량 규칙

병렬 정렬은 정렬 입력 리스트의 페이지 수를 기준으로 스캔과 같은 형태의 규칙(활성화 조건 2,048 페이지, 페이지 수가 2배가 될 때마다 병렬 처리 수준 1 증가, **parallelism** 상한)에 따라 병렬 처리 수준이 계산된다.

- 입력의 행 수가 계산된 병렬 처리 수준 이하이면 단일 스레드로 정렬된다.
- ORDER BY와 LIMIT을 함께 사용하는 정렬은 입력의 행 수가 500행 미만이면 단일 스레드로 정렬된다.
- 활성화 조건 미달이면 **PARALLEL** 힌트가 있어도 단일 스레드로 정렬된다.

부질의 처리량 규칙

부질의(Subquery)의 병렬 실행은 복수의 부질의가 서로의 결과를 참조하지 않는 독립적인 구조일 때 활성화된다.

- 부질의 병렬 실행시 처리 수준은 2로 고정되어 적용된다. 예를 들어, 한 쿼리 내에 독립적인 부질의가 4개가 존재하더라도, 시스템은 2개의 병렬 워커를 할당하여 처리한다.
- 각 부질의의 병렬 실행 여부는 “처리량 규칙”에 의해 결정된다.
- 여러 독립적인 부질의가 존재하는 경우 병렬 실행의 효과가 크다

```
-- 병렬 부질의 실행 예제
-- parallelism=4로 설정된 경우, 2개의 부질의가 병렬로 실행됨
SELECT *
FROM orders
WHERE customer_id IN (
    SELECT customer_id FROM customers WHERE region = 'Asia'
)
AND product_id IN (
    SELECT product_id FROM products WHERE category = 'Electronics'
);
```

워커 스레드 풀 관리

병렬 스레드 풀이 부족하면 일부 연산만 병렬로 수행하거나, 병렬 실행을 전혀 수행하지 못할 수 있다.

- **max_parallel_workers** 파라미터로 전역 병렬 처리 워커 풀의 최대 스레드 개수를 설정한다
- 각 병렬 연산은 실행 전에 필요한 병렬 워커 수를 미리 병렬 워커 풀로부터 예약하고, 작업 완료 후 반환한다

- 워커 풀 경합으로 요청한 수를 모두 예약하지 못하면 실제 예약된 워커 수만큼만 병렬로 실행하며, 예약된 워커가 하나도 없으면 단일 스레드 방식으로 실행된다
- 질의 전체에서 사용되는 병렬 처리 수준의 합은 `parallelism` 파라미터 값을 넘을 수 있으나, `max_parallel_workers` 값을 초과할 수는 없다

```
-- cubrid.conf 설정 예제
max_parallel_workers=100 # 전역 워커 풀 크기
parallelism=4           # 단일 병렬 연산의 상한값
```

처리량 성능 고려사항

병렬 질의 처리량 규칙을 통한 최적화:

- 작은 입력에 대한 불필요한 병렬 실행을 방지하여 오버헤드를 줄인다
- 입력 크기에 비례하여 병렬 처리 수준을 자동으로 조정한다
- 과도한 병렬 실행으로 인한 시스템 자원 경쟁을 방지한다
- 효과가 큰 쿼리에 집중적으로 병렬 자원을 할당한다

권장 설정:

- **max_parallel_workers**: 동시 실행 가능한 병렬 질의 수와 각 쿼리의 평균 병렬 처리 수준을 고려하여 설정
- **parallelism**: 시스템의 물리 코어 수를 고려하여 설정 (보통 4~8 정도가 적절)
- 대용량 테이블이 많은 환경에서는 **max_parallel_workers** 값을 높게 설정
- 소규모 테이블이 많은 환경에서는 기본값 사용을 권장

분할

분할 기법(partitioning)은 하나의 테이블을 분할(partition)이라는 여러 독립적인 물리적 단위로 나눠 주는 기법이다. CUBRID에서 각 분할은 분할 테이블(partitioned table)의 서브클래스로 구현된 테이블이다. 각 분할은 **분할 키**와 분할 방식에 의해 분할 테이블 데이터의 일부분을 보유한다. 사용자는 분할 테이블에 질의문을 수행하여 분할된 데이터에 접근할 수 있다. 이것은 사용자가 해당 테이블에 접근하는데 사용되는 질의문이나 코드를 변경하지 않고 테이블을 분할할 수 있다는 것을 의미한다. 즉 응용 프로그램을 거의 변경하지 않고 분할의 이점을 얻을 수 있게 해준다.

분할 기법은 관리 편의성, 성능 및 가용성을 향상시킬 수 있다. 테이블을 분할하면 다음과 같은 이점이 있다.

- 대용량 테이블의 관리 편의성 향상
- 데이터 조회 시 접근 범위를 줄임으로써 성능 향상
- 디스크 I/O를 분산함으로써 성능 향상 및 물리적 부하 감소
- 여러 분할로 나눔으로써 전체 데이터의 훼손 가능성 감소 및 가용성 향상
- 스토리지 비용의 최적화

분할된 데이터는 시스템에 의해 자동으로 관리된다. 분할 테이블에서 실행되는 **INSERT** 문과 **UPDATE** 문은 실행하는 동안 레코드가 어느 분할에 위치해야 하는지 확인하기 위한 과정을 수행하게 된다. 시스템은 **UPDATE** 문이 수행되는 동안 수정된 레코드가 어떤 분할로 이동해야 할지를 확인하고 이동시켜줌으로써 분할 정의를 일관되게 유지한다. 유효하지 않은 분할에 레코드를 삽입하려고 하면 오류를 반환한다.

SELECT 문을 수행할 때 시스템은 검색 조건에 해당하는 결과를 보유하고 있는 분할만을 대상으로 하여 검색 공간을 좁히기 위해 **분할 프루닝** 작업을 적용한다. **SELECT** 문 수행 중에 대부분의 분할을 프루닝(제거)하는 작업은 성능을 크게 향상시킨다.

테이블 분할 기법은 큰 테이블에 적용할 때 가장 효과적이다. “큰” 테이블의 정확한 의미는 응용 프로그램과 테이블이 질의문에서 사용되는 방법에 달려있다. 테이블에 대해 **영역**, **리스트** 또는 **해시** 중 어느 것이 최적의 분할 방법인지는 테이블이 질의문에서 어떻게 사용되며 데이터가 어떻게 분할되는가에 따라 달라진다. 분할 테이블을 마치 일반 테이블처럼 사용할 수도 있지만, 분할 테이블을 사용할 때는 **분할에 관한 노트**를 고려해야 한다.

분할

분할 키

분할 키는 정의된 분할들에 데이터를 분배하기 위해 분할 방식에서 사용하는 표현식이다. 분할 키로 사용할 수 있는 칼럼의 데이터 타입은 다음과 같다.

- **CHAR**
- **VARCHAR**
- **SMALLINT**
- **INT**
- **BIGINT**

- **DATE**
- **TIME**
- **TIMESTAMP**
- **DATETIME**

분할 키에는 다음과 같은 제약 사항이 적용된다.

- 분할 키는 분할 테이블에서 하나의 칼럼만을 사용해야 한다.
- 집계 함수 및 분석 함수, 논리 연산자, 비교 연산자는 분할 키 표현식에 사용할 수 없다.
- 다음 함수 및 표현식은 분할 키 표현식에서 허용되지 않는다.
 - CASE
 - CHARSET()
 - CHR()
 - COALESCE()
 - SERIAL_CURRENT_VALUE()
 - SERIAL_NEXT_VALUE()
 - DECODE()
 - DECR()
 - INCR()
 - DRAND()
 - DRANDOM()
 - GREATEST()
 - LEAST()
 - IF()
 - IFNULL()
 - INSTR()
 - NVL()
 - NVL2()

- ROWNUM
- INST_NUM()
- USER
- PRIOR
- WIDTH_BUCKET()

- 각각의 고유 인덱스 키 또는 기본 키는 분할 키를 포함해야 한다. 이에 대한 자세한 내용은 [여기](#)를 참고한다.
- 분할 표현식의 길이는 1024바이트를 초과하면 안 된다.

영역 분할

영역 분할(range partitioning)은 각 분할에 대해 지정된 값의 영역으로 테이블을 분할하는 방법이다. 범위는 겹치지 않는 연속된 구간으로 정의된다. 이 분할 방법은 테이블의 데이터가 영역 구간으로 나누어질 수 있을 때 가장 유용한 방법이다. 예를 들면, 주문 정보 테이블에서 주문 날짜 또는 사용자 테이블에서 나이 영역으로 분할하는 경우이다. 영역 분할은 거의 모든 검색 조건이 영역을 매칭하는데 사용될 수 있기 때문에 **분할 프루닝** 측면에서 가장 다양하게 활용되는 분할 기법이다.

테이블은 **CREATE** 또는 **ALTER** 문에서 **PARTITION BY RANGE** 절을 사용하여 분할될 수 있다.

```
CREATE TABLE [schema_name.]table_name (
    ...
)
PARTITION BY RANGE ( <partitioning_key> ) (
    PARTITION partition_name VALUES LESS THAN ( <range_value> )
    [COMMENT 'comment_string' ] ,
    PARTITION partition_name VALUES LESS THAN ( <range_value> )
    [COMMENT 'comment_string' ] ,
    ...
)

ALTER TABLE [schema_name.]table_name
PARTITION BY RANGE ( <partitioning_key> ) (
    PARTITION partition_name VALUES LESS THAN ( <range_value> )
    [COMMENT 'comment_string' ] ,
    PARTITION partition_name VALUES LESS THAN ( <range_value> )
    [COMMENT 'comment_string' ] ,
    ...
)
```

- *partitioning_key*: **분할 키**를 지정한다.
- *partition_name*: 분할 이름을 지정한다.

- *range_value*: 분할 키의 최대 값을 지정한다. *range_value* 보다 작은 분할 키 값을 가지는 레코드들은 모두 해당 분할에 저장된다.
- *comment_string*: 각 분할의 커멘트를 지정한다.

다음은 올림픽 참가국 정보를 담은 *participant2* 테이블을 참가한 올림픽의 개최연도를 기준으로 2000년도 전의 참가국(*before_2000* 분할)과 2008년도 전의 참가국(*before_2008* 분할)로 나누는 영역 분할을 생성하는 예제이다.

```
CREATE TABLE participant2 (
    host_year INT,
    nation CHAR(3),
    gold INT,
    silver INT,
    bronze INT
)
PARTITION BY RANGE (host_year) (
    PARTITION before_2000 VALUES LESS THAN (2000),
    PARTITION before_2008 VALUES LESS THAN (2008)
);
```

분할을 생성할 때, 사용자가 제공한 영역을 가장 작은 값부터 가장 큰 값까지 정렬하고 정렬된 리스트에서 겹치지 않는 간격을 생성한다. 위 예에서 생성된 영역의 간격은 $[-\text{inf}, 2000)$ 와 $[2000, 2008)$ 이다. 분할에 대한 무제한의 최대값을 지정하고 싶으면 **MAXVALUE** 식별자를 사용한다.

```
ALTER TABLE participant2 ADD PARTITION (
    PARTITION before_2012 VALUES LESS THAN (2012),
    PARTITION last_one VALUES LESS THAN MAXVALUE
);
```

튜플을 영역 분할 테이블에 삽입할 때, 시스템은 분할 키를 평가하여 해당 튜플이 어느 분할 영역에 속하게 될 것인가를 식별한다. 분할 키 값이 **NULL**이면, 해당 튜플은 가장 작은 영역의 분할에 저장된다. 분할 키 값에 해당하는 영역이 없으면 오류를 반환한다. 또한 튜플을 업데이트할 때도 새로운 분할 키 값에 해당하는 영역이 존재하지 않으면 오류를 반환한다.

다음은 각 분할에 커멘트를 추가하는 예제이다.

```
CREATE TABLE tbl (a int, b int) PARTITION BY RANGE(a) (
    PARTITION less_1000 VALUES LESS THAN (1000) COMMENT 'less 1000 comment',
    PARTITION less_2000 VALUES LESS THAN (2000) COMMENT 'less 2000 comment'
);

ALTER TABLE tbl PARTITION BY RANGE(a) (
```

```
PARTITION less_1000 VALUES LESS THAN (1000) COMMENT 'new
partition comment');
```

분할 커멘트를 확인하는 방법은 **분할 커멘트**를 참고한다.

해시 분할

해시 분할은 지정된 개수의 분할로 데이터를 분배하기 위해 사용되는 분할 기법이다. 이 분할 기법은 테이블 데이터의 영역이나 리스트가 의미 없는 값을 포함할 때 유용하다. 예를 들어, 키워드 테이블이나 `user_id`가 가장 관심 있는 값인 사용자 테이블과 같은 경우에 해당된다. 분할 키 값이 테이블 데이터를 고르게 분배한다면, 해시 분할 기법은 정의된 분할들에 테이블 데이터를 고르게 배분해준다. 해시 분할에 대한 **분할 프루닝** 최적화는 동등 조건(=과 `IN` 조건)에만 적용될 수 있는데, 대부분의 질의가 분할 키에 대한 동등 조건으로 주어질 때에 해시 분할이 유용하다.

CREATE 또는 **ALTER** 문에서 **PARTITION BY HASH** 절을 사용하여 해시 분할을 할 수 있다.

```
CREATE TABLE [schema_name.]table_name (
    ...
)
PARTITION BY HASH ( <partitioning_key> )
PARTITIONS ( number_of_partitions )

ALTER TABLE [schema_name.]table_name
PARTITION BY HASH (<partitioning_key>)
PARTITIONS (number_of_partitions)
```

- `partitioning_key`: **분할 키**를 지정한다.
- `number_of_partitions`: 생성할 분할의 개수를 지정한다.

다음은 국가 코드와 국가 이름의 정보를 담은 `nation2` 테이블을 생성하고 `code` 값을 기준으로 4개의 해시 분할을 정의하는 예제이다. 해시 분할은 분할의 개수만 지정하고 이름은 지정하지 않는다.

```
CREATE TABLE nation2 (
    code CHAR (3),
    name VARCHAR (50)
)
PARTITION BY HASH (code) PARTITIONS 4;
```

해시 분할 테이블에 삽입될 때 데이터를 저장할 분할은 분할 키의 해시 값에 의해 결정된다. 분할 키 값이 **NULL**이면, 해당 레코드는 첫번째 분할에 저장된다.

리스트 분할

리스트 분할은 사용자가 지정한 분할 키 값의 리스트에 따라 테이블을 분할하는 기법이다. 분할을 위한 값의 리스트는 겹치는 값이 없어야 한다. 이 분할 기법은 사원 테이블의 부서 ID, 사용자 테이블의 국가 코드와 같은 경우처럼 테이블 데이터가 의미 있는 값의 리스트로 나누어질 때 유용하다. 해시 분할과 마찬가지로, 리스트 분할에 대한 **분할 프루닝** 최적화는 동등 조건(=과 IN 조건)에만 적용된다.

CREATE 또는 **ALTER** 문에서 **PARTITION BY LIST** 절을 사용하여 리스트 분할을 할 수 있다.

```
CREATE TABLE [schema_name.]table_name (
    ...
)
PARTITION BY LIST ( <partitioning_key> ) (
    PARTITION partition_name VALUES IN ( <values_list> ) [COMMENT
'comment_string'],
    PARTITION partition_name VALUES IN ( <values_list> ) [COMMENT
'comment_string'],
    ...
)

ALTER TABLE [schema_name.]table_name
PARTITION BY LIST ( <partitioning_key> ) (
    PARTITION partition_name VALUES IN ( <values_list> ) [COMMENT
'comment_string'],
    PARTITION partition_name VALUES IN ( <values_list> ) [COMMENT
'comment_string'],
    ...
)
```

- *partitioning_key*: **분할 키**를 지정한다.
- *partition_name*: 분할 명을 지정한다.
- *partition_value_list*: 분할의 기준이 되는 값의 목록을 지정한다.
- *comment_string*: 각 분할의 커멘트를 지정한다.

다음은 선수의 이름과 종목 정보를 담고 있는 *athlete2* 테이블을 생성하고 종목에 따른 리스트 분할을 정의하는 예제이다.

```
CREATE TABLE athlete2 (name VARCHAR (40), event VARCHAR (30))
PARTITION BY LIST (event) (
    PARTITION event1 VALUES IN ('Swimming', 'Athletics'),
    PARTITION event2 VALUES IN ('Judo', 'Taekwondo', 'Boxing'),
    PARTITION event3 VALUES IN ('Football', 'Basketball', 'Baseball')
);
```

리스트 분할 테이블에 튜플을 삽입할 때 분할 키 값은 분할에 정의된 리스트 값 중 하나에 속해야 한다. 리스트 분할의 경우 분할 키 값이 **NULL**일 때 자동으로 특정 분할을 할당하지 않고 오류가 발생된다. **NULL** 값을 저장하려면 다음의 예와 같이 **NULL**을 포함하는 분할을 생성해야 한다.

```
CREATE TABLE athlete2 (name VARCHAR (40), event VARCHAR (30))
PARTITION BY LIST (event) (
    PARTITION event1 VALUES IN ('Swimming', 'Athletics' ),
    PARTITION event2 VALUES IN ('Judo', 'Taekwondo', 'Boxing'),
    PARTITION event3 VALUES IN ('Football', 'Basketball',
    'Baseball', NULL)
);
```

다음은 각 분할에 커멘트를 추가하는 예제이다.

```
CREATE TABLE athlete2 (name VARCHAR (40), event VARCHAR (30))
PARTITION BY LIST (event) (
    PARTITION event1 VALUES IN ('Swimming', 'Athletics') COMMENT
    'G1',
    PARTITION event2 VALUES IN ('Judo', 'Taekwondo', 'Boxing')
    COMMENT 'G2',
    PARTITION event3 VALUES IN ('Football', 'Basketball',
    'Baseball') COMMENT 'G3');

CREATE TABLE athlete3 (name VARCHAR (40), event VARCHAR (30));
ALTER TABLE athlete3 PARTITION BY LIST (event) (
    PARTITION event1 VALUES IN ('Handball', 'Volleyball', 'Tennis')
    COMMENT 'G1');
```

분할 커멘트

분할 커멘트는 영역 분할과 리스트 분할에 대해서만 지정할 수 있으며, 해시 분할에서는 지정할 수 없다. 분할 커멘트는 다음 구문을 실행하여 확인할 수 있다.

```
SHOW CREATE TABLE [schema_name.]table_name;
SELECT class_name, partition_name, COMMENT FROM db_partition WHERE
class_name ='table_name';
```

또는 CSQL 인터프리터에서 테이블의 스키마를 출력하는 ;sc 명령으로 인덱스의 커멘트를 확인할 수 있다.

```
$ csql -u dba demodb

csql> ;sc tbl
```


분할 프루닝

분할 프루닝(Partition Pruning)은 분할 테이블 조회 시 분할 키 조건을 평가하여, 접근해야 할 분할(Partition) 범위를 최소화하는 최적화 기법이다. 결과에 포함될 가능성이 없는 분할을 미리 제외함으로써 디스크 I/O와 스캔 비용을 줄이고 질의 성능을 향상시킬 수 있다.

분할 프루닝의 적용 여부는 질의 컴파일 단계에서 결정하지 않으며, 질의 실행을 준비하는 단계에서 분할 키 조건으로 사용되는 값을 분석하여 결정한다. 따라서 동일한 형태의 질이라도 분할 키 조건으로 사용되는 값이 달라지면 분할 프루닝 적용 여부도 달라질 수 있다.

분할 프루닝 적용 여부는 질의 실행 계획에 표시되지 않지만, 질의 실행 후 프로파일링 결과를 통해 이를 확인할 수 있다. 질의 프로파일링에 대한 자세한 내용은 [질의 프로파일링](#)을 참고한다.

참고

CUBRID 9.0 미만 버전에서는 분할 프루닝 여부를 질의 컴파일 단계에서 결정되지만, 9.0 이상 버전부터는 질의 실행 준비 단계에서 결정된다.

분할 방식별 분할 프루닝 지원 비교 연산자

분할 방법	동등 조건	범위 조건
영역 분할	<code>=</code> , <code>IN</code>	<code><</code> , <code>≤</code> , <code>></code> , <code>≥</code> , <code>BETWEEN</code>
리스트 분할	<code>=</code> , <code>IN</code>	미지원
해시 분할	<code>=</code> , <code>IN</code>	미지원

분할 프루닝이 적용되지 않는 경우

- **WHERE** 절의 분할 키 조건이 분할 키로 정의한 표현식과 다르거나, 표현식이 동일하더라도 인자 순서가 다른 경우
- 분할 방식에서 분할 프루닝을 지원하지 않는 비교 연산자를 사용하는 경우
- 질의 실행 준비 단계에서 분할 키 조건의 값을 확인할 수 없는 경우

분할 직접 접근

```
... FROM [<schema_name>.]<partition_table_name>__p__<partition_name>
[AS <alias_name>] ...
```

```
... FROM [<schema_name>.<partition_table_name> PARTITION
(<partition_name>)] [AS <alias_name>] ...
```

WHERE 절에 분할 키 조건이 없거나 분할 프루닝이 적용되지 않더라도, **PARTITION** 절을 사용하거나 분할 이름을 직접 지정하여 특정 분할만 조회할 수 있다. 이 방식은 일반 테이블을 조회하는 것과 동일하게 동작하므로 분할 테이블에서는 제한되었던 일부 최적화 기법도 적용 가능하다. 다만 질의 범위가 해당 분할로 고정되기 때문에, 조건에 맞는 레코드가 다른 분할에 있더라도 검색 대상에서 제외되므로 잘못된 결과가 반환될 수 있다.

INSERT와 **UPDATE** 문에서도 특정 분할을 지정할 수 있지만, 처리하려는 레코드가 해당 분할에 속하지 않으면 오류가 발생할 수 있으니 주의해야 한다. 특히 **INSERT** 문은 분할을 지정하더라도 별도의 성능 이점이 없으므로 분할을 직접 지정하는 방식을 권장하지 않는다.

또한, 분할에 직접 접근하면 분할 테이블이 제공하는 운영상의 장점을 누리기 어렵다. 테이블 단위로 질의를 작성하면 향후 분할 구성이 변경되어도 응용 프로그램을 수정할 필요가 없지만, 분할을 직접 명시하면 이러한 유연성을 확보할 수 없기 때문이다. 따라서 특별한 목적이 있는 경우가 아니라면 분할 테이블을 통해 데이터에 접근하는 것을 권장한다.

분할 테이블에 적용할 수 없는 최적화 기법에 대한 자세한 내용은 [분할된 테이블에 대한 제약들](#)을 참고한다.

예제 1. 질의 프로파일링 결과를 통해 분할 프루닝 확인

이번 예제에서는 질의 프로파일링 결과를 통해 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 **WHERE** 절에 분할 키 조건이 존재하며, 해당 조건은 **CREATE TABLE** 문에서 정의한 분할 키 표현식과 동일하다. 또한 범위 분할은 범위 조건에 대한 분할 프루닝을 지원하므로 분할 프루닝이 적용될 수 있다. 다만 분할 프루닝은 질의 실행 준비 단계에서 결정하므로 실행 계획만으로는 적용 여부를 확인할 수 없다.

```
drop table if exists olympic_range;

create table olympic_range
partition by range (host_year) (
    partition before_1920 values less than (1920),
    partition before_1940 values less than (1940),
    partition before_1960 values less than (1960),
    partition before_1980 values less than (1980),
    partition before_2000 values less than (2000),
    partition latest values less than maxvalue
)
as (select * from olympic);

create index i_olympic_range_host_nation on olympic_range
(host_nation);
```

```
update statistics on olympic_range;
```

```
set optimization level 513;
set trace on;

select /*+ recompile */
       o.host_year, o.host_nation, o.host_city, o.mascot
from   olympic_range as o
where  o.host_year > 1990
       and o.host_nation = 'USA';
```

```
Join graph segments (f indicates final):
seg[0]: [0]
seg[1]: host_year[0] (f)
seg[2]: host_nation[0] (f)
seg[3]: host_city[0] (f)
seg[4]: mascot[0] (f)
Join graph nodes:
node[0]: public.olympic_range o(25/6) (sargs 0 1) (loc 0)
Join graph terms:
term[0]: o.host_nation='USA' (sel 0.0555556) (sarg term) (not-join
eligible) (indexable host_nation[0]) (loc 0)
term[1]: o.host_year range (1990 gt_inf max) (sel 0.1) (rank 2)
(sarg term) (not-join eligible) (loc 0)
```

Query plan:

```
iscan
  class: o node[0]
  index: i_olympic_range_host_nation term[0]
  sargs: term[1]
  cost:  3 card 1
```

Query stmt:

```
select o.host_year, o.host_nation, o.host_city, o.mascot from
olympic_range o where (o.host_year> ?:0 ) and o.host_nation= ?:1
```

host_year	host_nation	host_city	mascot
1996	'USA'	'Atlanta'	'Izzy'

질의 실행 후 프로파일링 결과를 확인하면, 분할 테이블의 각 분할에 대한 스캔이 **SCAN** 하위에 **PARTITION**으로 출력되는 것을 확인할 수 있다. **SCAN**에는 분할 테이블 전체에 대한 스캔 정보가 표시되고, **PARTITION**에는 각 분할에 대한 스캔 정보가 표시된다. **PARTITION**으로 출력되지 않은 분할은 분할 프루닝이 적용되어 스캔 대상에서 제외된 것이다.

`o.host_year > 1990` 조건을 만족하는 `before_2000` 과 `latest` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```
show trace;
```

Query Plan:

```
INDEX SCAN (o.i_olympic_range_host_nation) (key range:
o.host_nation= ?:1 )
```

```
rewritten query: select o.host_year, o.host_nation, o.host_city,
o.mascot from [public.olympic_range] o where (o.host_year> ?:0 ) and
o.host_nation= ?:1
```

Trace Statistics:

```
SELECT (time: 0, fetch: 10, fetch_time: 0, ioread: 0)
SCAN (index: public.olympic_range.i_olympic_range_host_nation),
(btrees time: 0, fetch: 6, ioread: 0, readkeys: 1, filteredkeys: 1,
rows: 2) (lookup time: 0, rows: 1)
PARTITION (index:
public.olympic_range__p__before_2000.i_olympic_range_host_nation),
(btrees time: 0, fetch: 4, ioread: 0, readkeys: 1, filteredkeys: 1,
rows: 2) (lookup time: 0, rows: 1)
PARTITION (index:
public.olympic_range__p__latest.i_olympic_range_host_nation), (btrees
time: 0, fetch: 2, ioread: 0, readkeys: 0, filteredkeys: 0, rows: 0)
(lookup time: 0, rows: 0)
```

아래 질의는 **WHERE** 절에 분할 키 조건이 없으므로 분할 프루닝이 적용되지 않는다.

```
set trace on;
```

```
select /* recompile */
o.host_year, o.host_nation, o.host_city, o.mascot
from
olympic_range as o
where
o.opening_date > '1990-12-31'
and o.host_nation = 'USA';

show trace;
```

```

      host_year  host_nation      host_city      mascot
=====
=====
          1996   'USA'           'Atlanta'       'Izzy'

```

질의 실행 후 프로파일링 결과를 확인하면, 분할 프루닝이 적용되지 않아 모든 분할에 대한 스캔 정보가 **SCAN** 하위에 **PARTITION**으로 출력된 것을 확인할 수 있다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 24, fetch_time: 1, ioread: 0)
    SCAN (index: public.olympic_range.i_olympic_range_host_nation),
    (btree time: 1, fetch: 16, ioread: 0, readkeys: 3, filteredkeys: 3,
    rows: 4) (lookup time: 1, rows: 1)
      PARTITION (index:
      public.olympic_range__p__before_1920.i_olympic_range_host_nation),
      (btree time: 0, fetch: 3, ioread: 0, readkeys: 1, filteredkeys: 1,
      rows: 1) (lookup time: 0, rows: 0)
        PARTITION (index:
        public.olympic_range__p__before_1940.i_olympic_range_host_nation),
        (btree time: 0, fetch: 3, ioread: 0, readkeys: 1, filteredkeys: 1,
        rows: 1) (lookup time: 0, rows: 0)
          PARTITION (index:
          public.olympic_range__p__before_1960.i_olympic_range_host_nation),
          (btree time: 0, fetch: 2, ioread: 0, readkeys: 0, filteredkeys: 0,
          rows: 0) (lookup time: 0, rows: 0)
            PARTITION (index:
            public.olympic_range__p__before_1980.i_olympic_range_host_nation),
            (btree time: 0, fetch: 2, ioread: 0, readkeys: 0, filteredkeys: 0,
            rows: 0) (lookup time: 0, rows: 0)
              PARTITION (index:
              public.olympic_range__p__before_2000.i_olympic_range_host_nation),
              (btree time: 1, fetch: 4, ioread: 0, readkeys: 1, filteredkeys: 1,
              rows: 2) (lookup time: 1, rows: 1)
                PARTITION (index:
                public.olympic_range__p__latest.i_olympic_range_host_nation), (btree
                time: 0, fetch: 2, ioread: 0, readkeys: 0, filteredkeys: 0, rows: 0)
                (lookup time: 0, rows: 0)

```

예제 2. 산술 표현식 분할 키의 분할 프루닝

이번 예제에서는 산술 표현식으로 정의한 분할 키를 **WHERE** 절에서 동일한 표현식으로 사용하는 경우와, 인자 순서만 달라진 경우에 대해, 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 **WHERE** 절에 산술 표현식 분할 키 조건이 존재하며, 해당 조건은 **CREATE TABLE** 문에서 정의한 분할 키 표현식과 동일하다. 또한 범위 분할은 범위 조건에 대한 분할 프루닝을 지원하므로 분할 프루닝이 적용될 수 있다.

```
drop table if exists olympic_arith_range;

create table olympic_arith_range
partition by range (host_year + 5) (
    partition before_1925 values less than (1925),
    partition before_1945 values less than (1945),
    partition before_1965 values less than (1965),
    partition before_1985 values less than (1985),
    partition before_2005 values less than (2005),
    partition latest values less than maxvalue
)
as (select * from olympic);

update statistics on olympic_arith_range;
```

```
set trace on;

select /*+ recompile */
    o.host_year, o.host_nation, o.host_city, o.mascot
from
    olympic_arith_range as o
where
    o.host_year + 5 between 1975 and 1995
order by
    o.host_year;

show trace;
```

host_year	host_nation	host_city	mascot
1972	'Germany'	'Munich'	'Waldi'
1976	'Canada'	'Montreal'	'Amik'
1980	'USSR'	'Moscow'	'Misha'
1984	'USA'	'Los Angeles'	'Sam'
1988	'Korea'	'Seoul'	'HODORI'

질의 실행 후 프로파일링 결과를 확인하면, `o.host_year + 5 between 1975 and 1995` 조건을 만족하는 `before_1985` 와 `before_2005` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```
Trace Statistics:
  SELECT (time: 4, fetch: 4, fetch_time: 0, ioread: 0)
    SCAN (table: public.olympic_arith_range), (heap time: 0, fetch:
2, ioread: 0, readrows: 10, rows: 5)
      PARTITION (table:
public.olympic_arith_range__p__before_1985), (heap time: 0, fetch:
1, ioread: 0, readrows: 5, rows: 2)
```

```

PARTITION (table:
public.olympic_arith_range__p__before_2005), (heap time: 0, fetch:
1, ioread: 0, readrows: 5, rows: 3)
ORDERBY (time: 4, sort: true, page: 0, ioread: 0)

```

산술 표현식의 결과는 같지만 인자 순서가 다르면 분할 프루닝이 적용되지 않는다.

```

set trace on;

select /* recompile */
    o.host_year, o.host_nation, o.host_city, o.mascot
from
    olympic_arith_range as o
where
    5 + o.host_year between 1975 and 1995
order by
    o.host_year;

show trace;

```

host_year	host_nation	host_city	mascot
1972	'Germany'	'Munich'	'Waldi'
1976	'Canada'	'Montreal'	'Amik'
1980	'USSR'	'Moscow'	'Misha'
1984	'USA'	'Los Angeles'	'Sam'
1988	'Korea'	'Seoul'	'HODORI'

질의 실행 후 프로파일링 결과를 확인하면, 분할 프루닝이 적용되지 않아 모든 분할에 대한 스캔 정보가 **SCAN** 하위에 **PARTITION**으로 출력된 것을 확인할 수 있다.

```

Trace Statistics:
  SELECT (time: 0, fetch: 12, fetch_time: 0, ioread: 0)
    SCAN (table: public.olympic_arith_range), (heap time: 0, fetch:
6, ioread: 0, readrows: 25, rows: 5)
      PARTITION (table:
public.olympic_arith_range__p__before_1925), (heap time: 0, fetch:
1, ioread: 0, readrows: 5, rows: 0)
        PARTITION (table:
public.olympic_arith_range__p__before_1945), (heap time: 0, fetch:
1, ioread: 0, readrows: 5, rows: 0)
          PARTITION (table:
public.olympic_arith_range__p__before_1965), (heap time: 0, fetch:
1, ioread: 0, readrows: 3, rows: 0)
            PARTITION (table:
public.olympic_arith_range__p__before_1985), (heap time: 0, fetch:

```

```

1, ioread: 0, readrows: 5, rows: 2)
    PARTITION (table:
public.olympic_arith_range__p__before_2005), (heap time: 0, fetch:
1, ioread: 0, readrows: 5, rows: 3)
    PARTITION (table: public.olympic_arith_range__p__latest),
(heap time: 0, fetch: 1, ioread: 0, readrows: 2, rows: 0)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)

```

예제 3. 함수 표현식 분할 키의 분할 프루닝

이번 예제에서는 함수 표현식으로 정의한 분할 키를 **WHERE** 절에서 동일한 표현식으로 사용하는 경우와, 함수 표현식을 사용하지 않고 원본 컬럼만 사용하는 경우에 대해, 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 **WHERE** 절에 함수 표현식 분할 키 조건이 존재하며, 해당 조건은 **CREATE TABLE** 문에서 정의한 분할 키 표현식과 동일하다. 또한 범위 분할은 범위 조건에 대한 분할 프루닝을 지원하므로 분할 프루닝이 적용될 수 있다.

```

drop table if exists olympic_func_range;

create table olympic_func_range
partition by range (YEAR (opening_date)) (
    partition before_1920 values less than (1920),
    partition before_1940 values less than (1940),
    partition before_1960 values less than (1960),
    partition before_1980 values less than (1980),
    partition before_2000 values less than (2000),
    partition latest values less than maxvalue
)
as (select * from olympic);

update statistics on olympic_func_range;

```

```

set trace on;

select /** recompile */
    YEAR (o.opening_date) as opening_year, o.host_nation,
o.host_city, o.mascot
from
    olympic_func_range as o
where
    YEAR (o.opening_date) between 1970 and 1990
order by
    opening_year;

show trace;

```


opening_year	host_nation	host_city	mascot
=====			
=====			
1972	'Germany'	'Munich'	'Waldi'
1976	'Canada'	'Montreal'	'Amik'
1980	'USSR'	'Moscow'	'Misha'
1984	'USA'	'Los Angeles'	'Sam'
1988	'Korea'	'Seoul'	'HODORI'

질의 실행 후 프로파일링 결과를 확인하면, `YEAR (o.opening_date) between 1970 and 1990` 조건을 만족하는 `before_1980` 과 `before_2000` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```
Trace Statistics:
  SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
    SCAN (table: public.olympic_func_range), (heap time: 0, fetch:
2, ioread: 0, readrows: 10, rows: 5)
      PARTITION (table:
public.olympic_func_range__p__before_1980), (heap time: 0, fetch: 1,
ioread: 0, readrows: 5, rows: 2)
        PARTITION (table:
public.olympic_func_range__p__before_2000), (heap time: 0, fetch: 1,
ioread: 0, readrows: 5, rows: 3)
          ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```

분할 키로 정의한 함수 표현식이 원본 컬럼과 동일한 정렬 순서를 보장하는 경우에는, 함수 표현식 대신 원본 컬럼을 사용하더라도 분할 프루닝이 적용될 수 있다.

```
set trace on;

select /*+ recompile */
  YEAR (o.opening_date) as opening_year, o.host_nation,
o.host_city, o.mascot
from
  olympic_func_range as o
where
  o.opening_date between '1970-01-01' and '1990-12-31'
order by
  opening_year;

show trace;
```

opening_year	host_nation	host_city	mascot
=====			
=====			
1972	'Germany'	'Munich'	'Waldi'
1976	'Canada'	'Montreal'	'Amik'

1980	'USSR'	'Moscow'	'Misha'
1984	'USA'	'Los Angeles'	'Sam'
1988	'Korea'	'Seoul'	'HODORI'

질의 실행 후 프로파일링 결과를 확인하면, 함수 표현식 분할 키 조건을 직접 사용하지 않았음에도 **YEAR** 함수가 원본 컬럼과 동일한 정렬 순서를 보장하므로 분할 프루닝이 적용되었음을 확인할 수 있다.

```
Trace Statistics:
  SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
    SCAN (table: public.olympic_func_range), (heap time: 0, fetch:
2, ioread: 0, readrows: 10, rows: 5)
      PARTITION (table:
public.olympic_func_range__p__before_1980), (heap time: 0, fetch: 1,
ioread: 0, readrows: 5, rows: 2)
        PARTITION (table:
public.olympic_func_range__p__before_2000), (heap time: 0, fetch: 1,
ioread: 0, readrows: 5, rows: 3)
          ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```

조건절에는 `o.opening_date` 가 사용되었지만 `YEAR(o.opening_date)` 와 정렬 순서가 동일하기 때문에, `YEAR(o.opening_date) between YEAR('1970-01-01') and YEAR('1990-12-31')` 조건을 만족하는 `before_1980` 과 `before_2000` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

예제 4. 리스트 분할의 분할 프루닝

이번 예제에서는 리스트 분할 방식으로 분할 테이블을 생성한 뒤, **WHERE** 절에 동등 조건을 사용하는 경우와 범위 조건을 사용하는 경우에 대해, 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 **WHERE** 절에 분할 키 조건이 존재하며, 해당 조건은 **CREATE TABLE** 문에서 정의한 분할 키 표현식과 동일하다. 또한 리스트 분할은 동등 조건에 대한 분할 프루닝을 지원하므로 분할 프루닝이 적용될 수 있다.

```
drop table if exists participant_list;

create table participant_list
partition by list (host_year) (
  partition p1988 values IN (1988),
  partition p1992 values IN (1992),
  partition p1996 values IN (1996),
  partition p2000 values IN (2000),
  partition p2004 values IN (2004)
)
as (select * from participant);

update statistics on participant_list;
```

```

set trace on;

select /*+ recompile */
      p.host_year, p.nation_code, p.gold
from
      participant_list as p
where
      p.host_year in (1988, 1996)
      and p.gold > 40
order by
      p.host_year, p.gold, p.nation_code;

show trace;

```

host_year	nation_code	gold
1988	'URS'	55
1996	'USA'	44

질의 실행 후 프로파일링 결과를 확인하면, `p.host_year in (1988, 1996)` 조건을 만족하는 `p1988` 과 `p1996` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 4, fetch_time: 0, ioread: 0)
    SCAN (table: public.participant_list), (heap time: 1, fetch: 2,
ioread: 0, readrows: 352, rows: 2)
      PARTITION (table: public.participant_list__p__p1988),
(heap time: 1, fetch: 1, ioread: 0, readrows: 156, rows: 1)
      PARTITION (table: public.participant_list__p__p1996),
(heap time: 0, fetch: 1, ioread: 0, readrows: 196, rows: 1)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)

```

리스트 분할은 동등 조건에서만 분할 프루닝을 지원하므로, 범위 조건 등 동등 조건이 아닌 경우에는 분할 프루닝이 적용되지 않는다.

```

set trace on;

select /*+ recompile */
      p.host_year, p.nation_code, p.gold
from
      participant_list as p
where
      p.host_year < 1996
      and p.gold > 40
order by
      p.host_year;

```

```
show trace;
```

```

      host_year  nation_code      gold
=====
      1988      'URS'            55
      1992      'EUN'            45

```

질의 실행 후 프로파일링 결과를 확인하면, 분할 프루닝이 적용되지 않아 모든 분할에 대한 스캔 정보가 **SCAN** 하위에 **PARTITION**으로 출력된 것을 확인할 수 있다.

각 분할의 값을 기준으로 보면 `p1996`, `p2000`, `p2004` 분할은 스캔 대상에서 제외될 것처럼 보이지만, 리스트 분할은 정렬된 순서대로 값이 배치되는 구조가 아니므로 범위 조건만으로는 특정 분할이 스캔 대상에서 제외될 수 없다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 10, fetch_time: 0, ioread: 0)
    SCAN (table: public.participant_list), (heap time: 0, fetch: 5,
ioread: 0, readrows: 916, rows: 2)
      PARTITION (table: public.participant_list__p__p1988),
(heap time: 0, fetch: 1, ioread: 0, readrows: 156, rows: 1)
      PARTITION (table: public.participant_list__p__p1992),
(heap time: 0, fetch: 1, ioread: 0, readrows: 165, rows: 1)
      PARTITION (table: public.participant_list__p__p1996),
(heap time: 0, fetch: 1, ioread: 0, readrows: 196, rows: 0)
      PARTITION (table: public.participant_list__p__p2000),
(heap time: 0, fetch: 1, ioread: 0, readrows: 197, rows: 0)
      PARTITION (table: public.participant_list__p__p2004),
(heap time: 0, fetch: 1, ioread: 0, readrows: 202, rows: 0)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)

```

예제 5. 해시 분할의 분할 프루닝

이번 예제에서는 해시 분할 방식으로 분할 테이블을 생성한 뒤, **WHERE** 절에 동등 조건을 사용하는 경우와 부정 조건을 사용하는 경우에 대해, 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 **WHERE** 절에 분할 키 조건이 존재하며, 해당 조건은 **CREATE TABLE** 문에서 정의한 분할 키 표현식과 동일하다. 또한 해시 분할은 동등 조건에 대한 분할 프루닝을 지원하므로 분할 프루닝이 적용될 수 있다.

```

drop table if exists stadium_hash;

create table stadium_hash
partition by hash (nation_code) partitions 4
as (select * from stadium);

```

```
update statistics on stadium_hash;
```

```
set trace on;

select /*+ recompile */
       s.nation_code, s.name, s.seats
from   stadium_hash as s
where  s.nation_code = 'KOR'
       and s.seats >= 100000
order by
       s.name;

show trace;
```

nation_code	name	seats
'KOR'	'Seoul Olympic Stadium'	100000

질의 실행 후 프로파일링 결과를 확인하면, `s.nation_code = 'KOR'` 조건을 만족하는 `p2` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```
Trace Statistics:
  SELECT (time: 0, fetch: 2, fetch_time: 0, ioread: 0)
    SCAN (table: public.stadium_hash), (heap time: 0, fetch: 1,
ioread: 0, readrows: 32, rows: 1)
      PARTITION (table: public.stadium_hash__p__p2), (heap
time: 0, fetch: 1, ioread: 0, readrows: 32, rows: 1)
```

해시 분할은 동등 조건에서만 분할 프루닝을 지원하므로, 부정 조건 등 동등 조건이 아닌 경우에는 분할 프루닝이 적용되지 않는다.

```
set trace on;

select /*+ recompile */
       s.nation_code, s.name, s.seats
from   stadium_hash as s
where  s.nation_code != 'KOR'
       and s.seats > 100000
order by
       s.name;
```

```
show trace;
```

```

nation_code      name                      seats
=====
'AUS'            'Olympic Stadium'        115600

```

질의 실행 후 프로파일링 결과를 확인하면, 분할 프루닝이 적용되지 않아 모든 분할에 대한 스캔 정보가 **SCAN** 하위에 **PARTITION**으로 출력된 것을 확인할 수 있다.

앞선 질의에서 `s.nation_code = 'KOR'` 조건을 만족하는 경우 p2 분할만 스캔되었으므로 부정 조건에서는 스캔 대상에서 제외될 것처럼 보이지만, 해당 분할에도 `s.nation_code ≠ 'KOR'` 조건을 만족하는 레코드가 존재할 수 있어 스캔 대상에서 제외될 수 없다.

```

Trace Statistics:
  SELECT (time: 0, fetch: 8, fetch_time: 0, ioread: 0)
    SCAN (table: public.stadium_hash), (heap time: 0, fetch: 4,
ioread: 0, readrows: 141, rows: 1)
      PARTITION (table: public.stadium_hash__p__p0), (heap
time: 0, fetch: 1, ioread: 0, readrows: 27, rows: 0)
      PARTITION (table: public.stadium_hash__p__p1), (heap
time: 0, fetch: 1, ioread: 0, readrows: 53, rows: 0)
      PARTITION (table: public.stadium_hash__p__p2), (heap
time: 0, fetch: 1, ioread: 0, readrows: 32, rows: 0)
      PARTITION (table: public.stadium_hash__p__p3), (heap
time: 0, fetch: 1, ioread: 0, readrows: 29, rows: 1)

```

예제 6. 조인에서의 분할 프루닝

이번 예제에서는 조인 질의 시 분할 테이블이 드리븐(driven) 테이블로 사용되는 경우, 분할 프루닝의 적용 여부를 확인한다.

아래 질의는 힌트를 사용하여 분할 테이블이 드리븐 테이블로 사용되도록 조인 순서를 고정하며, 이에 따라 조인 조건이 분할 키 조건으로 활용될 것으로 기대할 수 있다. 하지만 분할 프루닝은 질의 실행 준비 단계에서 확인 가능한 값을 분석하여 결정된다. 따라서 조인 실행 시점에 컬럼 값이 확정되는 조인 조건으로는 질의 실행 준비 단계에서 값을 미리 알 수 없으므로, 분할 프루닝이 적용되지 않는다.

```

drop table if exists olympic_range;

create table olympic_range
partition by range (host_year) (
  partition before_1920 values less than (1920),
  partition before_1940 values less than (1940),
  partition before_1960 values less than (1960),
  partition before_1980 values less than (1980),

```

```

    partition before_2000 values less than (2000),
    partition latest values less than maxvalue
)
as (select * from olympic);

update statistics on olympic_range;

```

```

set trace on;

select /** recompile ordered */
    p.host_year, p.nation_code, p.gold,
    o.host_nation, o.slogan
from
    participant as p
    inner join olympic_range as o on p.host_year = o.host_year
where
    p.host_year in (1988, 2004)
    and p.gold > 40
order by
    p.host_year, p.gold, p.nation_code;

show trace;

```

```

    host_year  nation_code      gold
host_nation      slogan
=====
=====
1988  'URS'      55
'Korea'      'Harmony and progress'

```

질의 실행 후 프로파일링 결과를 확인하면, 분할 프루닝이 적용되지 않아 모든 분할에 대한 스캔 정보가 **SCAN** 하위에 **PARTITION**으로 출력된 것을 확인할 수 있다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 2204, fetch_time: 0, ioread: 0)
    SCAN (index:
public.participant.pk_participant_host_year_nation_code), (btree
time: 1, fetch: 2172, ioread: 0, readkeys: 2154, filteredkeys: 2148,
rows: 2148) (lookup time: 1, rows: 6)
      SCAN (table: public.olympic_range), (heap time: 0, fetch: 31,
ioread: 0, readrows: 25, rows: 1)
        PARTITION (table:
public.olympic_range__p__before_1920), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 0)
          PARTITION (table:
public.olympic_range__p__before_1940), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 0)

```

```

PARTITION (table:
public.olympic_range__p__before_1960), (heap time: 0, fetch: 4,
ioread: 0, readrows: 3, rows: 0)
PARTITION (table:
public.olympic_range__p__before_1980), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 0)
PARTITION (table:
public.olympic_range__p__before_2000), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 1)
PARTITION (table: public.olympic_range__p__latest),
(heap time: 0, fetch: 3, ioread: 0, readrows: 2, rows: 0)
MEMOIZE (time: 0, hit: 0, miss: 1, size: 0KB, enabled: true)

```

분할 테이블이 트리본 테이블로 사용되는 경우에도, 조건절에 질의 실행 준비 단계에서 확인 가능한 값을 가진 분할 키 조건이 존재한다면 분할 프루닝이 적용될 수 있다.

```

set trace on;

select /** recompile ordered */
  p.host_year, p.nation_code, p.gold,
  o.host_nation, o.slogan
from
  participant as p
  inner join olympic_range as o on p.host_year = o.host_year
where
  o.host_year in (1988, 2004)
  and p.gold > 40
order by
  p.host_year, p.gold, p.nation_code;

show trace;

```

```

      host_year  nation_code          gold
host_nation      slogan
=====
=====
      1988   'URS'                  55
'Korea'          'Harmony and progress'

```

질의 실행 후 프로파일링 결과를 확인하면, `o.host_year in (1988, 2004)` 조건을 만족하는 `before_2000` 분할과 `latest` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 1845, fetch_time: 1, ioread: 0)
    SCAN (table: public.participant), (heap time: 1, fetch: 1838,
ioread: 0, readrows: 1832, rows: 6)
      SCAN (table: public.olympic_range), (heap time: 0, fetch: 6,

```



```

ioread: 0, readrows: 21, rows: 1)
    PARTITION (table:
public.olympic_range__p__before_2000), (heap time: 0, fetch: 3,
ioread: 0, readrows: 15, rows: 1)
    PARTITION (table: public.olympic_range__p__latest),
(heap time: 0, fetch: 3, ioread: 0, readrows: 6, rows: 0)
    MEMOIZE (time: 0, hit: 0, miss: 3, size: 0KB, enabled: true)

```

분할 키 조건에 바인드 변수가 사용되는 경우에도, 질의 실행 준비 단계에서 바인딩된 값을 확인할 수 있으므로 분할 프루닝이 적용될 수 있다. 여러 종류의 표현식이 중첩되더라도 질의 실행 준비 단계에서 상수로 평가될 수 있다면 분할 프루닝이 적용될 수 있다.

```

set trace on;

prepare q from '
select /*+ ordered */
    p.host_year, p.nation_code, p.gold,
    o.host_nation, o.slogan
from
    participant as p
    inner join olympic_range as o on p.host_year = o.host_year
where
    o.host_year = year (to_date (?, 'YYYY-MM-DD'))
    and p.gold > 40
order by
    p.host_year, p.gold, p.nation_code;
';

execute q using '1988-01-01';

show trace;

```

```

      host_year  nation_code          gold
host_nation      slogan
=====
=====
      1988   'URS'                  55
'Korea'          'Harmony and progress'

```

질의 실행 후 프로파일링 결과를 확인하면, `o.host_year = year (to_date ('1988-01-01', 'YYYY-MM-DD'))` 조건을 만족하는 `before_2000` 분할만 스캔되었으며, 그 외 나머지 분할은 스캔 대상에서 제외되었다.

```

Trace Statistics:
  SELECT (time: 1, fetch: 165, fetch_time: 0, ioread: 0)
    SCAN (index:
public.participant.pk_participant_host_year_nation_code), (btree
time: 1, fetch: 158, ioread: 0, readkeys: 157, filteredkeys: 156,

```

```

rows: 156) (lookup time: 1, rows: 1)
  SCAN (table: public.olympic_range), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 1)
    PARTITION (table:
public.olympic_range__p__before_2000), (heap time: 0, fetch: 6,
ioread: 0, readrows: 5, rows: 1)

```

예제 7. 분할 직접 접근

이번 예제에서는 분할 프루닝 외에 **PARTITION** 절을 사용하여 특정 분할을 직접 조회하는 방법을 확인한다. 이를 통해 분할 테이블에는 적용할 수 없었던 최적화 기법이 분할 단위에서는 적용되는지 그 여부를 확인한다.

아래 질의는 분할 테이블을 대상으로 조회하므로, **ORDER BY** 및 **LIMIT** 절과 적절한 인덱스가 존재하더라도 **SKIP ORDER BY** 최적화는 적용되지 않는다.

```

drop table if exists olympic_range;

create table olympic_range
partition by range (host_year) (
  partition before_1920 values less than (1920),
  partition before_1940 values less than (1940),
  partition before_1960 values less than (1960),
  partition before_1980 values less than (1980),
  partition before_2000 values less than (2000),
  partition latest values less than maxvalue
)
as (select * from olympic);

create index i_olympic_range_host_year on olympic_range (host_year);

update statistics on olympic_range;

```

```

set optimization level 513;
set trace on;

select /** recompile */
  o.host_year, o.host_nation, o.host_city, o.mascot
from
  olympic_range as o
where
  o.host_year > 2000
order by
  o.host_year desc
limit 1;

show trace;

```

```

Join graph segments (f indicates final):
seg[0]: [0]
seg[1]: host_year[0] (f)
seg[2]: host_nation[0] (f)
seg[3]: host_city[0] (f)
seg[4]: mascot[0] (f)
Join graph nodes:
node[0]: public.olympic_range o(25/6) (sargs 0) (loc 0)
Join graph terms:
term[0]: o.host_year range (2000 gt_inf max) (sel 0.1) (rank 2)
(sarg term) (not-join eligible) (indexable host_year[0]) (loc 0)

```

Query plan:

```

temp(order by)
  subplan: iscan
            class: o node[0]
            index: i_olympic_range_host_year term[0]
            cost: 3 card 2
  sort: 1 desc
  cost: 9 card 2

```

Query stmt:

```

select o.host_year, o.host_nation, o.host_city, o.mascot from
olympic_range o where (o.host_year> ?:0 ) order by 1 desc for
orderby_num(<= ?:1

```

host_year	host_nation	host_city	mascot
2004	'Greece'	'Athens'	'Athena Phevos'

질의 실행 후 프로파일링 결과를 확인하면, **SKIP ORDER BY** 최적화가 적용되지 않아 **ORDERBY**에서 **TopNSort**가 수행되었음을 확인할 수 있다.

```

Query Plan:
  SORT (order by)
    INDEX SCAN (o.i_olympic_range_host_year) (key range:
(o.host_year> ?:0 ))

  rewritten query: select o.host_year, o.host_nation, o.host_city,
o.mascot from [public.olympic_range] o where (o.host_year> ?:0 )
order by 1 desc for orderby_num(<= ?:1

```

Trace Statistics:

```

SELECT (time: 0, fetch: 6, fetch_time: 0, ioread: 0)

```

```

SCAN (index: public.olympic_range.i_olympic_range_host_year),
(btrees time: 0, fetch: 3, ioread: 0, readkeys: 1, filteredkeys: 1,
rows: 1) (lookup time: 0, rows: 1)
  PARTITION (index:
public.olympic_range__p__latest.i_olympic_range_host_year), (btrees
time: 0, fetch: 3, ioread: 0, readkeys: 1, filteredkeys: 1, rows: 1)
(lookup time: 0, rows: 1)
  ORDERBY (time: 0, topnsort: true)

```

PARTITION 절을 통해 특정 분할만 직접 조회하면 일반 테이블과 동일하게 처리되므로, **SKIP ORDER BY** 최적화가 적용될 수 있다.

```

set optimization level 513;
set trace on;

select /** recompile */
  o.host_year, o.host_nation, o.host_city, o.mascot
from
  olympic_range partition (latest) as o
where
  o.host_year > 2000
order by
  o.host_year desc
limit 1;

show trace;

```

```

Join graph segments (f indicates final):
seg[0]: [0]
seg[1]: host_year[0] (f)
seg[2]: host_nation[0] (f)
seg[3]: host_city[0] (f)
seg[4]: mascot[0] (f)
Join graph nodes:
node[0]: public.olympic_range__p__latest o(25/1) (sargs 0) (loc 0)
Join graph terms:
term[0]: o.host_year range (2000 gt_inf max) (sel 0.1) (rank 2)
(sarg term) (not-join eligible) (indexable host_year[0]) (loc 0)

```

Query plan:

```

iscan
  class: o node[0]
  index: i_olympic_range_host_year term[0] (desc_index)
  sort: 1 desc
  cost: 4 card 2

```

Query stmt:

```
select o.host_year, o.host_nation, o.host_city, o.mascot from
olympic_range__p__latest o where (o.host_year> ?:0 ) order by 1
desc for orderby_num(<= ?:1

/* ---> skip ORDER BY */
```

```
      host_year  host_nation      host_city      mascot
=====
=====
      2004    'Greece'          'Athens'          'Athena
Phevos '
```

질의 실행 후 프로파일링 결과를 확인하면, **SKIP ORDER BY** 최적화가 적용되었음을 확인할 수 있다.

```
Query Plan:
  INDEX SCAN (o.i_olympic_range_host_year) (key range:
(o.host_year> ?:0 ), desc_index: true)
  skip order by: true

  rewritten query: select o.host_year, o.host_nation, o.host_city,
o.mascot from [public.olympic_range__p__latest] o where
(o.host_year> ?:0 ) order by 1 desc for orderby_num(<= ?:1

Trace Statistics:
  SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
  SCAN (index:
public.olympic_range__p__latest.i_olympic_range_host_year), (btree
time: 0, fetch: 3, ioread: 0, readkeys: 2, filteredkeys: 1, rows: 1)
(lookup time: 0, rows: 1)
```

분할 관리

ALTER 문의 분할 지정 절을 사용하여 다음과 같이 분할 테이블을 관리할 수 있다.

1. 분할 테이블을 일반 테이블로 변경
2. 분할 재구성
3. 이미 존재하는 분할 테이블에 분할 추가
4. 분할 제거하기
5. 분할을 일반 테이블로 승격

분할 테이블을 일반 테이블로 변경

분할 테이블을 일반 테이블로 변경하려면 **ALTER TABLE** 문을 이용한다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name REMOVE PARTITIONING
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 변경하고자 하는 테이블의 이름을 지정한다.

분할 설정을 제거하면 각 분할에 있던 모든 데이터가 분할 테이블로 이동된다. 이는 비용이 많이 드는 작업으로 주의해서 계획해야 한다.

분할 재구성

분할 재구성은 하나의 분할을 더 작은 분할들로 나누거나 한 그룹의 분할들을 하나의 분할로 병합하는 작업이다. 이를 수행하려면 **ALTER** 문의 **REORGANIZE PARTITION** 절을 사용한다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
REORGANIZE PARTITION <alter_partition_name_comma_list>
INTO "(" <partition_definition_comma_list> ")"

partition_definition_comma_list ::=
PARTITION partition_name VALUES LESS THAN "(" <range_value> ")", ...
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 재정의할 테이블의 이름을 지정한다.
- *alter_partition_name_comma_list*: 재정의할 현재 분할들을 지정한다. 여러 개의 분할은 쉼표(,)로 구분된다.
- *partition_definition_comma_list*: 새 분할들을 지정한다. 여러 개의 분할은 쉼표(,)로 구분된다.

이 절은 영역 분할 및 리스트 분할에만 적용된다. 해시 분할 기법에서 데이터 분배는 영역 분할과 리스트 분할과는 의미적으로 다르므로, 해시 분할 테이블은 분할 추가 및 삭제만 허용한다. 자세한 사항은 **해시 분할 재구성** 절을 참고한다.

다음 예는 **participant2** 테이블의 *before_2000* 분할을 *before_1996* 분할과 *before_2000* 분할로 재구성하는 방법이다.

```
ALTER TABLE participant2
REORGANIZE PARTITION before_2000 INTO (
  PARTITION before_1996 VALUES LESS THAN (1996),
```

```
PARTITION before_2000 VALUES LESS THAN (2000)
);
```

다음 예는 위의 예에서 정의된 두 개의 분할을 다시 하나의 *before_2000*로 병합하는 방법이다.

```
ALTER TABLE participant2
REORGANIZE PARTITION before_1996, before_2000 INTO (
    PARTITION before_2000 VALUES LESS THAN (2000)
);
```

다음 예는 *athlete2* 테이블에서 정의된 *event2* 분할을 *event2_1* (Judo)와 *event2_2* (Taekwondo, Boxing)으로 재구성하는 방법이다.

```
ALTER TABLE athlete2
REORGANIZE PARTITION event2 INTO (
    PARTITION event2_1 VALUES IN ('Judo'),
    PARTITION event2_2 VALUES IN ('Taekwondo', 'Boxing')
);
```

다음 예는 *event2_1*과 *event2_2* 분할을 다시 *event2* 분할로 합치는 방법이다.

```
ALTER TABLE athlete2
REORGANIZE PARTITION event2_1, event2_2 INTO (
    PARTITION event2 VALUES IN ('Judo', 'Taekwondo', 'Boxing')
);
```

참고

- 영역 분할 테이블에서 인접한 분할끼리만 재구성될 수 있다.
- 분할 재구성을 수행하는 동안, 새로 분할된 스키마에 맞춰 분할 간에 데이터를 이동한다. 재구성되는 분할의 크기에 따라 시간이 많이 소요될 수 있으므로 주의 깊게 해당 작업을 계획할 필요가 있다.
- **REORGANIZE PARTITION** 절은 분할 방법을 바꾸기 위해 사용할 수 없다. 예를 들어, 영역 분할 테이블을 해시 분할 테이블로 바꿀 수 없다.
- 분할을 재구성한 후에 최소한 하나의 분할이 존재해야 한다.

분할 추가

ALTER 문의 **ADD PARTITION** 절을 사용하여 분할 테이블에 분할을 추가할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
ADD PARTITION "(" <partition_definitions_comma_list> ")"
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 분할이 추가될 테이블 이름을 지정한다.
- *partition_definitions_comma_list*: 추가될 분할 이름을 지정한다. 여러 개인 경우 쉼표(,)로 구분한다.

다음 예는 **participant2** 테이블에 *before_2012* 분할과 *last_one* 분할을 추가하는 방법이다.

```
ALTER TABLE participant2 ADD PARTITION (
  PARTITION before_2012 VALUES LESS THAN (2012),
  PARTITION last_one VALUES LESS THAN MAXVALUE
);
```

참고

- 영역 분할 테이블에서 추가할 분할에 대한 영역 값은 기존 분할의 최대 영역 값보다 커야 한다.
- 영역 분할 테이블에서 **MAXVALUE** 로 최대값이 설정되어 있으면 **ADD PARTITION** 절은 항상 오류를 반환한다. 이 경우에 대신 **REORGANIZE PARTITION** 절을 사용해야 한다.
- **ADD PARTITION** 절은 이미 존재하는 분할 테이블에 대해서만 사용할 수 있다.
- **ADD PARTITION** 절이 해시 분할 테이블에 적용될 때는 다른 의미를 가진다. 이에 대한 자세한 사항은 **해시 분할 재구성** 절을 참고한다.

분할 제거

ALTER 문의 **DROP PARTITION** 절을 이용하여 분할 테이블에서 분할을 제거(drop)할 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
DROP PARTITION partition_name_list
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.

- *table_name*: 분할 테이블 이름을 지정한다.
- *partition_name_list*: 제거할 분할 이름을 지정한다. 여러 개인 경우 쉼표(,)로 구분한다.

다음은 *participant2* 테이블에서 *before_2000* 분할을 제거하는 방법이다.

```
ALTER TABLE participant2 DROP PARTITION before_2000;
```

참고

- 분할을 제거하면 해당 분할 내에 저장된 데이터도 모두 삭제된다. 데이터를 유지한 채로 테이블의 분할을 변경하고 싶다면 **ALTER TABLE ... REORGANIZE PARTITION** 문을 사용하면 된다.
- 분할을 제거할 경우 삭제된 행의 개수를 반환하지 않는다. 테이블과 분할을 유지한 채로 데이터만 삭제하고 싶은 경우 **DELETE** 문을 사용하면 된다.

해시 분할 테이블에 대해 이 구문을 사용할 수 없다. 해시 분할 테이블의 분할을 제거하려면 해시 분할에서만 사용하는 **해시 분할 재구성** 절을 참고한다.

해시 분할 재구성

해시 분할 테이블에서 분할 간의 데이터 분배는 CUBRID에 의해 내부적으로 관리되므로, 해시 분할 재구성은 리스트 분할이나 영역 분할에서의 재구성과 다르게 동작한다. 해시 분할 테이블에 정의된 분할 개수를 증가시키거나 감소시키는 것만 허용된다. 해시 분할 테이블의 분할 개수를 수정하더라도 데이터 손실은 발생되지 않는다. 그러나 해시 함수의 영역이 수정되기 때문에, 해시 분할의 일관성을 유지하기 위해 새로운 분할들 간에 데이터가 재분배되어야 한다.

해시 분할 테이블에 정의된 분할 개수는 **ALTER** 문의 **COALESCE PARTITION** 절을 이용하여 줄일 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
COALESCE PARTITION number_of_shrinking_partitions
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 재정의할 테이블의 이름을 지정한다.
- *number_of_shrinking_partitions*: 삭제하려는 분할 개수를 지정한다.

다음은 *nation2* 테이블의 분할 개수를 4 개에서 3 개로 줄이는 예제이다.

```
ALTER TABLE nation2 COALESCE PARTITION 1;
```

ALTER 문의 **ADD PARTITION** 절을 사용하여 **ALTER** 해시 분할 테이블에 정의된 분할 개수를 늘릴 수 있다.

```
ALTER [TABLE | CLASS] [schema_name.]table_name
ADD PARTITION PARTITIONS number
```

- *schema_name*: 테이블의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *table_name*: 분할 개수를 재정의할 테이블의 이름을 지정한다.
- *number*: 추가할 분할 개수를 지정한다.

다음은 **nation2** 테이블에 3 개의 분할을 추가하는 예이다.

```
ALTER TABLE nation2 ADD PARTITION PARTITIONS 3;
```

분할 승격

분할(partition) **PROMOTE** 문은 분할 테이블에서 사용자가 지정한 분할을 일반 테이블로 승격(promote)한다. 이것은 거의 사용하지 않는 오래된 데이터를 보관할(archiving) 목적으로 유지하고자 할 때 유용하다. 해당 분할을 일반 테이블로 승격함으로써 분할 테이블에 대한 접근 부하를 줄일 수 있고, 분할 테이블에서 제거된 데이터는 승격된 테이블에 유지되므로 여전히 해당 데이터를 접근할 수 있다. 분할을 승격(promote)하는 것은 비가역적인 작업으로 승격된 분할을 분할 테이블로 다시 되돌릴 수 없다.

분할 **PROMOTE** 문은 영역 분할 테이블과 리스트 분할 테이블에만 허용된다. 해시 분할 테이블은 사용자가 해시 분할 간에 데이터 분배를 제어할 수 없으므로 승격을 허용하지 않는다.

분할이 일반 테이블로 승격될 때 승격 테이블은 데이터와 일반 인덱스만 상속받는다. 다음의 테이블 속성들은 승격된 테이블에 저장되지 않는다.

- 기본 키
- 외래 키
- 고유 인덱스
- **AUTO_INCREMENT** 속성 및 시리얼
- 트리거
- 메서드

- 상속 관계(수퍼클래스와 서브클래스)

분할을 승격하는 구문은 다음과 같다.

```
ALTER TABLE [schema_name.]table_name PROMOTE PARTITION
<partition_name_list>
```

- <partition_name_list> : 승격할 분할 이름으로, 여러 개를 쉼표(,)로 구분한다.

다음은 분할 테이블을 생성하고, 일부 튜플을 삽입한 후 이들 중 2 개의 분할을 승격하는 예이다.

```
CREATE TABLE t (i INT) PARTITION BY LIST (i) (
    PARTITION p0 VALUES IN (1, 2),
    PARTITION p1 VALUES IN (3, 4),
    PARTITION p2 VALUES IN (5, 6)
);

INSERT INTO t VALUES(1), (2), (3), (4), (5), (6);
```

테이블 *t*의 스키마와 데이터는 다음과 같다.

```
csql> ;schema t
=== <Help: Schema of a Class> ===
...
<Partitions>
    PARTITION BY LIST ([i])
    PARTITION p0 VALUES IN (1, 2)
    PARTITION p1 VALUES IN (3, 4)
    PARTITION p2 VALUES IN (5, 6)

csql> SELECT * FROM t;

=== <Result of SELECT Command in Line 1> ===
      i
=====
      1
      2
      3
      4
      5
      6
```

다음 구문은 *p0* 분할과 *p2* 분할을 승격한다.

```
ALTER TABLE t PROMOTE PARTITION p0, p2;
```

승격(promotion) 이후, 테이블 *t*는 *p1*이라는 하나의 분할만 포함하며 다음 데이터를 유지한다.

```
csql> ;schema t
=== <Help: Schema of a Class> ===
<Class Name>
    t
...
<Partitions>
    PARTITION BY LIST ([i])
    PARTITION p1 VALUES IN (3, 4)

csql> SELECT * FROM t;

=== <Result of SELECT Command in Line 1> ===
      i
=====
      3
      4
```

분할 테이블의 인덱스

분할 테이블에서 생성되는 모든 인덱스는 로컬 인덱스이다. 로컬 인덱스의 경우 각 분할에 대한 데이터가 별도의(로컬) 인덱스로 저장된다. 다른 분할의 데이터에 액세스하는 트랜잭션이 다른 로컬 인덱스에도 액세스하므로 분할 테이블 인덱스의 동시성을 향상시킨다.

고유 인덱스를 생성할 때 다음 제약 사항을 충족해야 한다.

- 고유 인덱스 키 또는 기본 키는 분할 키를 포함해야 한다.

이를 충족하지 않으면 CUBRID에서 오류가 반환된다.

```
csql> CREATE TABLE t(i INT , j INT) PARTITION BY HASH (i) PARTITIONS
4;
Execute OK. (0.142929 sec) Committed.

1 command(s) successfully processed.
csql> ALTER TABLE t ADD PRIMARY KEY (i);
Execute OK. (0.123776 sec) Committed.

1 command(s) successfully processed.
csql> CREATE UNIQUE INDEX idx2 ON t(j);

In the command from line 1,

ERROR: Partition key attributes must be present in the index key.
```

```
0 command(s) successfully processed.
```

로컬 인덱스의 이점을 이해하는 것이 중요하다. 글로벌 인덱스 스캔의 경우 프루닝(pruning)되지 않은 분할에 대해 각각 별도의 인덱스 스캔이 수행된다. 디스크에서 다른 분할에 있는 데이터(지금 스캔 중인 분할이 아닌 다른 분할에 속한 데이터)를 가져온 다음 버리기 때문에 로컬 인덱스 스캔보다 성능이 저하된다. **INSERT** 질의문도 글로벌 인덱스보다 크기가 더 작은 로컬 인덱스에서 향상된 성능을 보인다.

분할에 관한 노트

분할된 테이블은 일반적인 테이블 처럼 정상적으로 동작한다. 하지만 분할된 테이블의 장점을 충분히 살리기 위해서 적용을 고려해야하는 노트가 있다.

분할 테이블에 관한 통계

CUBRID 9.0에서 부터, **ALTER** 문의 **ANALYZE PARTITION** 절은 더 이상 사용되지 않는다. 질의를 수행하는 동안 분할을 잘라내는 것이 발생하고, 이러한 경우 이 문은 유용한 결과를 생산하지 못한다. 9.0에서 부터, CUBRID는 각 분할에 대한 통계를 분리 유지한다. 분할된 테이블의 통계는 각 분할에 대한 통계의 평균 값으로 계산된다. 이것은 일상적인 경우의 최적화, 하나를 제외하고 모든 분할이 제거된 분할에 대한 질의, 등을 위해서 진행되었다.

분할된 테이블에 대한 제약들

다음의 제약이 분할된 테이블에 적용된다:

- 하나의 테이블에 대해서 최대 1,024 까지의 분할이 정의될 수있다.
- 분할은 상속 체인의 부분이 될 수 없다. 클래스는 하나의 분할을 상속할 수 없고, 분할은 분할된 클래스(기본으로 상속한다)를 제외한 다른 클래스를 상속할 수 없다.
- 다음의 질의 최적화는 분할된 테이블에 대해서 수행되지 않는다:
 - ORDER BY skip (for details, see [ORDER BY 절 최적화](#))
 - GROUP BY skip (for details, see [GROUP BY 절 최적화](#))
 - Multi-key range optimization (for details, see [다중 키 범위 최적화](#))
 - INDEX JOIN

분할 키와 문자셋, 콜레이션

분할하는 키들과 분할의 정의는 같은 문자셋이어야 한다. 아래의 질의는 오류를 반환한다:

```
CREATE TABLE t (c CHAR(50) COLLATE utf8_bin)
PARTITION BY LIST (c) (
    PARTITION p0 VALUES IN (_utf8'x'),
    PARTITION p1 VALUES IN (_iso88591'y')
);
```

```
ERROR: Invalid codeset '_iso88591' for partition value. Expecting
'_utf8' codeset.
```

분할 키에서 비교 작업을 수행할 때 분할 테이블에 정의된 콜레이션을 사용한다. 다음 예제에서 utf8_en_ci 콜레이션의 'test'는 'TEST'와 같으므로 오류를 반환한다.

```
CREATE TABLE tbl (str STRING) COLLATE utf8_en_ci
PARTITION BY LIST (str) (
    PARTITION p0 VALUES IN ('test'),
    PARTITION p1 VALUES IN ('TEST')
);
```

```
ERROR: Partition definition is duplicated. 'p1'
```

다국어 지원

다국어 지원(Globalization)은 다양한 언어와 지역에 적용될 수 있도록 하는 국제화(Internationalization)와 언어별 구성 요소를 추가하여 특정 지역의 언어나 문화에 맞추는 현지화(Localization)를 포함한다. CUBRID는 현지화를 쉽게 하기 위해 유럽과 아시아를 포함한 여러 언어들의 콜레이션(collation)을 지원한다.

문자 데이터 설정에 대한 전반적인 사항을 알고 싶다면 [다국어 설정을 위한 고려 사항](#)을 참고한다.

문자셋, 콜레이션, 로캘이 무엇인지 알고 싶다면 [다국어 개요](#)를 참고한다.

타임존 타입 및 관련 함수들은 [타임존이 있는 날짜/시간 데이터 타입](#)을 참고한다. 타임존 관련 시스템 파라미터는 [타임존 파라미터](#)를 참고한다. 타임존 정보를 새로운 정보로 업데이트하고 싶은 경우 타임존 라이브러리를 재컴파일해야 되며, 이와 관련하여 [타임존 라이브러리 컴파일](#)를 참고한다.

원하는 언어와 지역에 따른 로캘을 데이터베이스에 반영하고 싶으면 반드시 로캘을 먼저 설정한 후 데이터베이스를 생성해야만 한다. 이와 관련하여 [로캘 설정](#)을 참고한다.

데이터베이스에 설정된 콜레이션 또는 문자셋을 변환하고 싶으면 해당 칼럼, 테이블, 표현식에 [COLLATE 수정자](#) 또는 [CHARSET 수정자](#)를 지정하고, 문자열 상수에 [COLLATE 수정자](#) 또는 [문자셋 소개자](#)를 지정한다. 이와 관련하여 [콜레이션](#)을 참고한다.

문자열 관련 함수나 연산자는 문자셋과 콜레이션에 따라 다르게 동작할 수 있다. 이와 관련하여 **문자셋과 콜레이션을 필요로 하는 연산**을 참고한다.

다국어 개요

문자 데이터

콜레이션을 적용할 수 있는 데이터의 타입은 VARCHAR, CHAR, ENUM 타입이다.

문자셋(charset, codeset)은 어떤 타입에 대해 문자가 저장되는 방식 또는 일련의 바이트가 하나의 문자를 구성하는 방식을 제어한다. CUBRID는 UTF-8, ISO-8859-1, EUC-KR 문자셋을 지원하며, UTF-8에 대해 코드포인트 10FFFF(4 바이트까지 인코딩됨)까지만 지원한다. 예를 들어, 문자 “Ç”는 코드셋 ISO-8859-1에서 1 바이트(C7)으로 인코딩되지만, UTF-8에서는 2 바이트(C3 88)로 인코딩된다. 반면, EUC-KR에서 이 문자는 사용할 수 없다.

콜레이션은 문자를 비교하는 방법을 결정한다. 사용자는 흔히 콜레이션을 통해 대소문자를 구분하지 않거나 구분한다. 예를 들어 “ABC”와 “abc”는 대소문자 구분 무시(case insensitive) 콜레이션에서는 동일한 반면, 대소문자 구분(case sensitive) 콜레이션에서는 동일하지 않다. 또한 콜레이션에 따라 비교 결과가 “ABC” > “abc”가 되거나 “ABC” < “abc”가 될 수 있다.

콜레이션은 대소문자 비교 이상의 것을 의미한다. 콜레이션은 두 문자열 간에 관계를 결정하여 LIKE와 같은 문자열 매칭에 사용되거나 인덱스 스캔에서 영역을 계산하는데 사용된다.

CUBRID에서 콜레이션은 문자셋을 포함한다. 예를 들어, “utf8_en_ci”와 “iso88591_en_ci”는 대소문자를 구분하지 않는 비교이지만 다른 문자셋에서 동작한다. ASCII 범위에서 비교 결과는 비슷하지만 UTF-8 인코딩은 가변적인 바이트 수에 따라 동작해야 하기 때문에 “utf8_en_ci” 콜레이션에서 비교 수행 속도가 더 느리다.

- “a’ COLLATE iso88591_en_ci”는 “_iso88591’a’ COLLATE iso88591_en_ci”를 나타낸다.
- “a’ COLLATE utf8_en_ci”는 “_utf8’a’ COLLATE utf8_en_ci”를 나타낸다.

모든 문자 데이터 타입은 자릿수(precision)를 지원한다. 고정 길이 문자(CHAR)는 특별히 다루어져야 한다. CHAR 타입의 값은 자릿수를 채워서 저장된다. 예를 들어 “abc”를 CHAR(5) 칼럼에 저장하면, “abc ”(2 개의 공백 문자가 패딩됨)를 저장하게 된다. 공백 문자(ASCII 32, Unicode 0020)는 대부분 문자에서 사용하는 패딩 문자이다. 하지만 EUC-KR 문자셋에서 패딩 문자는 두 개 바이트(A1 A1)로 저장된 하나의 문자로 구성된다.

관련 용어

- **문자셋(character set)**: 기호를 인코딩(어떤 기호에 특정 번호를 부여)한 집합
- **콜레이션(collation)**: 문자셋에서 문자를 비교하기 위한, 데이터 정렬을 위한 규칙의 집합

- **로캘(locale)**: 사용자의 언어 및 국가에 따라 숫자 형식, 캘린더(월의 이름, 요일의 이름), 날짜/시간 형식, 콜레이션 등을 정의하는 인자들의 집합. 로캘은 언어에 대한 지역화(localization)를 정의한다. 로캘의 문자셋은 월의 이름 및 다른 데이터가 어떻게 인코딩되는지를 나타낸다. 로캘은 최소한 하나의 언어 식별자와 영역 식별자로 구성되어 있으며 language[_territory][.codeset]으로 표현한다. (예를 들어 UTF-8 인코딩을 쓰는 오스트레일리아 영어는 en_AU.utf8로 표기한다.)

- **유니코드 정규화(Unicode normalization)**: 모양이 같은 여러 문자들이 있을 경우 기준에 따라 이를 하나로 통합하는 것. CUBRID는 입력 시에는 NFC(Normalization Form C) 형식(분해된 코드포인트에서 결합된 코드포인트로 변환)을 사용하고, 출력 시에는 NFD(Normalization Form D) 형식(결합된 코드포인트에서 분해된 코드포인트로 변환)을 사용한다. 하지만, CUBRID는 예외적으로 규범적 등가(canonical equivalence) 규칙을 적용하지 않는다.

예를 들어, 일반적인 NFC 규칙에 따르면 규범적 등가 규칙을 적용하여 코드포인트 212A(캘빈 기호 K)는 코드포인트 4B(ASCII 코드 대문자 K)로 변환된다. CUBRID는 규범적 등가 규칙에 의한 변환을 수행하지 않도록 하여 정규화 알고리즘을 더 간단하고 빠르게 하였고, 따라서 역변환도 수행하지 않는다.

- **규범적 등가(canonical equivalence)**: 시각적으로 구별이 불가능하고 텍스트 비교상 정확히 동일한 의미를 가지는 문자. 한 예로 'A'에 옹스트롬(Angstrom) 기호가 있는 'Å'가 있는데, 'Å'(유니코드 U+212B)와 라틴어 'A'(유니코드 U+00C5)는 모양이 같고 코드포인트가 다르지만 분해된 결과는 'A'와 U+030A로 같으므로 규범적 등가이다.

- **호환성 등가(compatibility equivalence)**: 동일한 문자나 문자 시퀀스의 대체 표현 문자. 예로는 숫자 '2'(유니코드 U+0032)와 위 첨자 '2'(유니코드 U+00B2)가 있는데, '²'는 숫자 '2'의 다른 형태이긴 하지만 시각적으로 구별되고 의미도 다르기 때문에 규범적 등가에 해당되지 않는다. '2²'를 NFC로 정규화하면 규범적 등가를 사용하기 때문에 '2²'가 유지되지만, NFKC 방식에서는 '2'가 호환성 등가인 '2'로 분해된 후 결합되어 '22'로 바뀔 수 있다. CUBRID의 유니코드 정규화에서는 호환성 등가 규칙도 적용하지 않는다.

유니코드 정규화에 대한 설명은 [유니코드 정규화](http://unicode.org/reports/tr15/) 를 참고하며, 보다 자세한 내용은 <http://unicode.org/reports/tr15/> 를 참고한다.

유니코드 정규화 관련 시스템 파라미터에서 기본으로 설정되는 값은 `unicode_input_normalization=no`이고 `unicode_output_normalization=no`이다. 이 파라미터에 대한 보다 자세한 설명은 [구문/타입 관련 파라미터](#) 를 참고한다.

로캘 속성

로캘은 다음과 같은 속성들로 정의된다.

- **문자셋(코드셋)**: 여러 바이트를 하나의 문자로 해석하는 방법을 정의한다. 유니코드에서는 여러 개의 바이트가 하나의 코드포인트(codepoint)를 구성하는 것으로 해석된다.

- **콜레이션(collation)**: LDML(UNICODE Locale Data Markup Language) 파일의 로캘 데이터에 여러 콜레이션을 지정할 수 있는데, 이 중에 마지막으로 명시된 것이 기본(default) 콜레이션으로 사용된다.

- **알파벳(대소문자 규칙)**: 하나의 로캘 데이터는 테이블 이름, 칼럼 이름과 같은 식별자용과 사용자 데이터용으로 최대 두 종류의 알파벳을 가질 수 있다.
- **캘린더**: 요일 이름, 월의 이름, 오전/오후(AM/PM) 표시
- **숫자 표기**: 자릿수 구분 기호
- **텍스트 변환 데이터**: CSQL 콘솔 변환용. 선택 사항
- **유니코드 정규화 데이터**: 모양이 같은 여러 문자들이 있을 경우 이를 기준에 따라 하나의 값으로 통합하는 정규화를 수행하여 변환된 데이터. 정규화 이후에는 로캘이 달라도 모양이 같은 문자는 같은 코드값을 가지며, 각 로캘은 이 정규화 기능을 활성화 또는 비활성화할 수 있다.

참고

일반적으로 한 로캘은 다양한 문자셋을 지원하지만, CUBRID 로캘은 영어와 한국어에 한해서만 ISO와 UTF-8 문자셋을 둘 다 지원한다. 그 외의 LDML 파일을 이용한 모든 사용자 정의 로캘은 UTF-8 문자셋만 지원한다.

콜레이션 속성

콜레이션(collation)은 문자열의 비교 및 정렬 규칙의 집합으로, CUBRID에서 콜레이션은 다음과 같은 속성(property)을 갖는다.

- **세기(strength)**: 기본 비교 항목들(문자들)이 어떻게 다른지 나타내는 측정 기준이다. 이것은 선택도(selectivity)에 영향을 준다. LDML 파일에서 콜레이션의 세기는 네 가지 수준(level)으로 설정할 수 있다. 예를 들어, 대소문자 구분이 없는 콜레이션은 level = “secondary” (2) 또는 “primary” (1)로 설정해야 한다.
- **확장(expansion)과 축약(contraction) 지원 여부**

각각의 칼럼이 콜레이션을 가질 수 있기 때문에, `LOWER()`, `UPPER()` 함수 등을 적용할 때 해당 콜레이션의 기본 언어에서 정의한 로캘의 대소문자 구분 규칙(casing rule)이 사용된다.

콜레이션 속성에 따라 일부 콜레이션에서 다음과 같은 특정 CUBRID 최적화가 동작하지 않을 수 있다.

- **LIKE 구문 재작성**: 같은 가중치(weight)에 여러 개의 다른 문자를 매핑하는 콜레이션, 예를 들어 대소문자 구분이 없는 콜레이션에서는 **LIKE** 구문이 재작성되지 않는다.
- **커버링 인덱스 스캔**: 같은 가중치에 여러 개의 다른 문자를 매핑하는 콜레이션에서는 커버링 인덱스 스캔이 동작하지 않는다([커버링 인덱스](#) 참고).

이에 관한 보다 자세한 설명은 [콜레이션 설정으로 인한 영향](#) 을 참고하면 된다.

콜레이션 명명 규칙

콜레이션 이름은 다음 규칙을 따른다.

```
<charset>_<lang specific>_<desc1>_<desc2>_...
```

- <charset>: 문자셋 이름. iso88591, utf8, euckr이 있다.
- <lang specific>: 지역/언어를 나타내며, en, de, es, fr, it, ja, km, ko, tr, vi, zh, ro가 있다. 특정 언어를 나타내지 않을 때는 “gen”으로 일반적인 정렬 규칙을 의미한다.
- <desc1>_<desc2>_...: 대부분 LDML 콜레이션에만 적용되며 각각 다음의 의미를 갖는다.
 - ci: 대소문자 구분 안 함. LDML 파일에서 다음을 설정하면 적용된다: strength=“secondary” caseLevel=“off” caseFirst=“off”
 - cs: 대소문자 구분. 기본적으로 모든 콜레이션은 대소문자를 구분한다. LDML 파일에서 다음을 설정하면 적용된다: strength=“tertiary”
 - bin: 정렬 순서가 코드포인트의 순서와 똑같음. 메모리의 바이트 순서를 비교하면 거의 같은 결과가 나오는데, 공백 문자와 EUC의 더블 바이트 패딩 문자는 “bin” 콜레이션에서 항상 0으로 정렬된다. LDML 파일에는 bin 콜레이션을 설정하는 방법이 없는데(bin 콜레이션은 이미 내장되어 있음), LDML 파일에서 다음을 설정하면 비슷하게 적용된다: strength=“quaternary” 또는 strength=“identical”
 - ai: 악센트 구분 안 함. 예를 들어, ‘Á’는 ‘A’와 같은 순서이다. 이는 또한 대소문자를 구분하지 않는다. LDML 파일에서 (strength=“primary”)를 설정하면 적용된다.
 - uca: UCA(Unicode Collation Algorithm) 기반 콜레이션을 의미함. 내장된 변형 콜레이션과 구별하기 위해서만 사용된다. 즉, 모든 LDML 콜레이션은 UCA를 기반으로 하지만 짧은 이름을 유지하기 위해 “_uca”가 생략되며, 예외적으로 ‘utf8_ko_cs_uca’, ‘utf8_tr_cs_uca’ 이 두 개의 콜레이션만 내장된 ‘utf8_ko_cs’, ‘utf8_tr_cs’ 콜레이션과 구별하기 위해 사용된다.
 - exp: 다른 콜레이션들이 문자 단위로 비교하는 것에 반해, **확장** 은 전체 단어 매칭/비교 알고리즘을 사용한다. 이 콜레이션은 좀더 복잡한 알고리즘을 사용하므로 훨씬 느릴 수 있지만, 알파벳 정렬에 유용할 수 있다. LDML 파일에 다음이 명시되어야 한다: CUBRIDExpansions=“use”
 - ab: 역순 악센트(accent backwards). 특히 캐나다 프랑스어에만 적용되는데, UCA 2단계(악센트 가중치를 저장)는 문자열의 끝에서부터 시작 방향으로 비교된다. 이 콜레이션 설정은 오직 **확장** 이 활성화되는 경우에만 사용될 수 있다. “ab” 설정은 다음 정렬을 허용한다.
 - 일반적인 악센트 순서: cote < coté < côte < côté
 - 역방향 악센트 순서: cote < côte < coté < côté

- cbm: 축약 영역 매칭(contraction boundary match). **확장** 과 **축약** 이 있는 콜레이션의 특별한 콜레이션이며, 매칭되는 문자열에서 **축약** 이 발견될 때 동작하는 방법을 설정한다. 콜레이션의 **축약** 을 “ch”로 정의했다고 가정하자. 그러면 일반적으로 “bac”라는 패턴은 문자열 “bachxxx”와는 매칭되지 않는다. 그러나 콜레이션이 “축약을 시작하는 문자 매칭”을 허용하도록 설정되면, 앞서 말한 문자열들은 매칭된다. 이러한 식으로 동작하는 콜레이션은 ‘utf8_ja_exp_cbm’ 밖에 없는데, 일본어 정렬은 무수히 많은 **축약** 을 요구한다.

콜레이션 이름은 동적으로 생성되지 않는다. LDML에 정의되어 있으며, 콜레이션 설정을 반영해야 한다.

콜레이션 이름은 콜레이션의 내부적인 ID에 영향을 준다. CUBRID는 256 개의 콜레이션을 허용하며, 각 식별자들은 다음과 같이 등록된다.

- 0 - 31: 내장된(built-in) 콜레이션(이름과 식별자가 제품에 포함됨)
- 32 - 46: 언어 부분에 “gen”을 가지는 LDML 콜레이션
- 47 - 255: 나머지 LDML 콜레이션

CUBRID가 제공하는 모든 로캘을 데이터베이스에 포함하고 싶다면, 먼저 \$CUBRID/conf 디렉터리의 cubrid_locales.all.txt 파일을 cubrid_locales.txt 파일로 복사한다. 그리고, make_locale 스크립트(확장자가 Linux는 .sh, Windows는 .bat)를 실행하면 된다. make_locale 스크립트에 대한 자세한 설명은 **2단계: 로캘 컴파일하기**를 참고하면 된다.

기존의 데이터베이스에 새로 추가한 로캘 정보를 포함하고 싶다면 cubrid synccolldb <dbname>을 실행한다. 이에 대한 자세한 설명은 **데이터베이스 콜레이션을 시스템 콜레이션에 동기화**를 참고하면 된다.

LDML 파일로 정의된 로캘을 모두 포함하는 경우 CUBRID는 다음의 콜레이션을 가진다.

CUBRID 콜레이션

콜레이션	대소문자 구분을 위한 로캘	문자 범위
iso88591_bin	en_US - 영어	ASCII + ISO88591 (C0-FE, except D7, F7)
iso88591_en_cs	en_US - 영어	ASCII + ISO88591 (C0-FE, except D7, F7)

콜레이션	대소문자 구분을 위한 로캘	문자 범위
iso88591_en_ci	en_US - 영어	ASCII + ISO88591 (C0-FE, except D7, F7)
utf8_bin	en_US - 영어	ASCII
euckr_bin	ko_KR - 한국어, en_US - 영어 와 같음	ASCII
utf8_en_cs	en_US - 영어	ASCII
utf8_en_ci	en_US - 영어	ASCII
utf8_tr_cs	tr_TR - 터키어	터키어 알파벳
utf8_ko_cs	ko_KR - 한국어, en_US - 영어 와 같음	ASCII
utf8_gen	de_DE - 독일어, 독일어 규칙 에 맞게 대소문자를 커스터마 이징한 유니코드	0000-FFFF 범위의 모든 유니 코드 코드포인트
utf8_gen_ai_ci	de_DE - 독일어, 독일어 규칙 에 맞게 대소문자를 커스터마 이징한 유니코드	0000-FFFF 범위의 모든 유니 코드 코드포인트
utf8_gen_ci	de_DE - 독일어, 독일어 규칙 에 맞게 대소문자를 커스터마 이징한 유니코드	0000-FFFF 범위의 모든 유니 코드 코드포인트
utf8_de_exp_ai_ci	de_DE - 독일어, 독일어 규칙 에 맞게 대소문자를 커스터마 이징한 유니코드	0000-FFFF 범위의 모든 유니 코드 코드포인트

콜레이션	대소문자 구분을 위한 로캘	문자 범위
utf8_de_exp	de_DE - 독일어, 독일어 규칙에 맞게 대소문자를 커스터마이징한 유니코드	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_ro_cs	ro_RO - 루마니아어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_es_cs	es_ES - 스페인어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_fr_exp_ab	fr_FR - 프랑스어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_ja_exp	ja_JP - 일본어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_ja_exp_cbm	ja_JP - 일본어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_km_exp	km_KH - 캄보디아어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_ko_cs_uca	ko_KR - 한국어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_tr_cs_uca	tr_TR - 터키어, 터키어 규칙에 맞게 대소문자를 커스터마이징한 유니코드	0000-FFFF 범위의 모든 유니코드 코드포인트
utf8_vi_cs		

콜레이션	대소문자 구분을 위한 로캘	문자 범위
	vi_VN - 베트남어, 일반적인 유니코드 대소문자 규칙과 동일	0000-FFFF 범위의 모든 유니코드 코드포인트
binary	none (invariant to casing operations)	any byte value (null-terminator는 0)

터키어의 대소문자 규칙은 i,I,j,J 에 대해 대소문자를 바꾼다. 독일어 대소문자 규칙은 ß에 대해 대소문자를 바꾼다.

위에서 iso88591_bin, iso88591_en_cs, iso88591_en_ci, utf8_bin, euckr_bin, utf8_en_cs, utf8_en_ci, utf8_tr_cs, utf8_ko_cs와 같은 9개의 콜레이션은 CUBRID에 기본적으로 내장되어 있다.

한 개 이상의 로케일 파일(.ldml)에 콜레이션이 포함되어 있는 경우 대소문자 구분을 위한 로케일(콜레이션의 기본 로케일)이 첫 번째로 포함되는 로케일이다. 로딩 순서는 \$CUBRID/conf/cubrid_locales.txt의 로케일 순서와 같다. 콜레이션 utf8_gen, utf8_gen_ci, utf8_gen_ai_ci에 대한 위의 로케일 대소문자는 cubrid_locales.txt의 기본 순서(알파벳 순서)를 따르므로 모든 일반 LDML 콜레이션의 기본 로케일은 de_DE(독일어)가 된다.

로캘 저장 위치

CUBRID는 로캘 설정을 위해 여러 디렉터리와 파일들을 사용한다.

- **\$CUBRID/conf/cubrid_locales.txt** 파일: 사용할 로캘 리스트를 포함하는 초기 설정 파일
- **\$CUBRID/conf/cubrid_locales.all.txt** 파일: **cubrid_locales.txt** 와 같은 구조를 갖는 초기 설정 파일의 템플릿. 사용자가 직접 정의하지 않아도 되는 CUBRID가 현재 지원하는 CUBRID 로캘 버전의 전체 리스트를 포함한다.
- **\$CUBRID/locales/data** 디렉터리: 로캘 데이터를 생성하는데 필요한 파일들을 포함한다.
- **\$CUBRID/locales/loclib** 디렉터리: 로캘 데이터를 포함하는 공유 라이브러리 생성을 위한 C 언어로 작성된 **locale_lib_common.h** 헤더 파일과 빌드를 위한 makefile을 포함한다.
- **\$CUBRID/locales/data/ducet.txt** 파일: 코드포인트, 축약과 확장 등과 같은 기본적인 범용 콜레이션 정보와 이들의 가중치 값을 표현하는 파일로, 이 정보들은 유니코드 컨소시엄에 의해 제정된 표준을 따른다. 자세한 사항은 http://unicode.org/reports/tr10/#Default_Unicode_Collation_Element_Table 을 참고한다.

- **\$CUBRID/locales/data/unicodedata.txt** 파일: 대소문자 구별, 분해, 정규화 등 각각의 유니코드 코드 포인트를 포함하는 파일로, CUBRID는 대소문자 구분 규칙을 결정하기 위해 이 파일을 사용한다. 더 많은 정보는 <https://docs.python.org/3/library/unicodedata.html> 을 참고한다.
- **\$CUBRID/locales/data/ldml** 디렉터리: `common_collations.xml` 파일과 `cubrid_<locale_name>.xml` 형식의 이름을 지니는 XML 파일들을 포함한다. `common_collations.xml` 파일은 모든 로캘에서 공유되는 콜레이션 정보를 표현하며, 각각의 `cubrid_<locale_name>.xml` 파일은 해당 언어에 대한 로캘 정보를 표현한다.
- **\$CUBRID/locales/data/codepages** 디렉터리: 한 바이트 코드 페이지들을 위한 코드 페이지 콘솔 변환용 파일들(8859-1.txt, 8859-15.txt, 8859-9.txt)과 멀티바이트 코드 페이지를 위한 코드 페이지 콘솔 변환용 파일들(CP1258.txt, CP923.txt, CP936.txt, CP949.txt)을 포함한다.
- **\$CUBRID/bin/make_locale.sh** 파일 또는 **%CUBRID%\bin\make_locale.bat** 파일(Windows): 로캘 데이터를 표현하는 공유 라이브러리를 생성하기 위해 사용되는 스크립트 파일이다.
- **\$CUBRID/lib** 디렉터리: 로캘 데이터를 표현하는 공유 라이브러리 파일이 저장된다.

로캘 설정

특정 언어의 문자셋과 콜레이션을 사용하려면 해당 문자셋은 새로 생성할 데이터베이스와 반드시 일치해야 한다. CUBRID가 지원하는 문자셋은 ISO-8859-1, EUC-KR, UTF-8이며, 데이터베이스를 생성할 때 사용하게 될 문자셋은 데이터베이스 생성 시 지정된다.

예를 들어, 데이터베이스를 생성 시 로캘을 `ko_KR.utf8`로 지정했으면 테이블 및 칼럼 생성, 콜레이션 변경 등에서 `utf8_ja_exp`과 같이 “utf8_”로 시작하는 콜레이션을 모두 사용할 수 있으나, `ko_KR.euckr`로 설정하면 다른 문자셋과 관련된 콜레이션을 모두 사용할 수 없게 된다. (CUBRID 콜레이션 참고)

다음은 로캘을 `en_US.utf8`로 설정하여 데이터베이스를 생성한 후, 콜레이션 `utf8_ja_exp`를 사용한 예이다.

1. `cd $CUBRID/conf`
2. `cp cubrid_locales.all.txt cubrid_locales.txt`
3. `make_locale.sh -t64 # 64 bit locale library creation`
4. `cubrid createdb testdb en_US.utf8`
5. `cubrid server start testdb`
6. `csql -u dba testdb`
7. csql에서 아래 질의 수행

```
SET NAMES utf8;
CREATE TABLE t1 (i1 INT , s1 VARCHAR(20) COLLATE utf8_ja_exp, a
```

```
INT, b VARCHAR(20) COLLATE utf8_ja_exp);
INSERT INTO t1 VALUES (1, 'いイ基盤', 1, 'いイ 繭');
```

보다 자세한 설명은 아래를 참고한다.

1단계: 로캘 선택

사용하려는 로캘을 **\$CUBRID/conf/cubrid_locales.txt** 에 지정한다. 모두 선택하거나 부분만 선택할 수 있다.

CUBRID가 현재 지원하는 로캘은 en_US, de_DE, es_ES, fr_FR, it_IT, ja_JP, km_KH, ko_KR, tr_TR, vi_VN, zh_CN, ro_RO이며, 이 목록은 **\$CUBRID/conf/cubrid_locales.all.txt**에 작성되어 있다.

각 로캘 이름 및 언어, 사용 국가는 다음 표와 같다.

로캘 이름	언어 - 사용 국가
en_US	영어 - 미국
de_DE	독일어 - 독일
es_ES	스페인어 - 스페인
fr_FR	프랑스어 - 프랑스
it_IT	이태리어 - 이탈리아
ja_JP	일본어 - 일본
km_KH	크메르어 - 캄보디아
ko_KR	한국어 - 대한민국
tr_TR	터키어 - 터키

로캘 이름	언어 - 사용 국가
vi_VN	베트남어 - 베트남
zh_CN	중국어 - 중국
ro_RO	루마니아어 - 루마니아

참고

지원하는 로캘들을 위한 LDML 파일들은 `cubrid_<locale_name>.xml` 파일로 명명되며, **\$CUBRID/locales/data/ldml** 폴더에 저장된다. 지원하려는 로캘에 해당하는 LDML 파일이 **\$CUBRID/locales/data/ldml** 디렉터리에 존재해야 한다. **cubrid_locales.txt**에 로캘이 지정되지 않거나 `cubrid_<locale_name>.xml` 파일이 존재하지 않으면 해당 로캘을 사용할 수 없다.

로캘 라이브러리들은 **\$CUBRID/conf/cubrid_locales.txt** 설정 파일에 의해 생성되는데, 이 파일은 원하는 로캘들의 언어 코드들을 포함하고 있다. 사용자가 정의하는 모든 로캘들은 UTF-8 문자셋으로만 생성된다. 또한 이 파일을 통해서 각 로캘 LDML 파일에 대한 파일 경로와 라이브러리들을 선택적으로 설정할 수 있다.

```
<lang_name>  <LDML file>
<lib file>
ko_KR        /home/CUBRID/locales/data/ldml/cubrid_ko_KR.xml    /
home/CUBRID/lib/libcubrid_ko_KR.so
```

기본적으로 LDML 파일은 **\$CUBRID/locales/data/ldml** 디렉터리에, 로캘 라이브러리들은 **\$CUBRID/lib** 디렉터리에 존재한다. 이와 같이 LDML 파일과 로캘 라이브러리가 기본 위치에 존재한다면 `<lang_name>`만 작성해도 된다. LDML을 위한 파일 이름 형식은 `cubrid_<lang_name>.ldml`이다.

라이브러리에 대한 파일 이름 형식은 Linux에서는 `libcubrid_<lang_name>.so`, Windows에서는 `libcubrid_<lang_name>.dll`이다.

2단계: 로캘 컴파일하기

1단계에서 설명한 요구사항들이 충족되었다면 로캘 데이터를 컴파일할 수 있다.

CUBRID에 내장된 로캘을 사용한다면 사용자 로캘 라이브러리를 컴파일하지 않고 사용할 수 있으므로 2단계를 생략할 수 있으나, 내장된 로캘과 라이브러리 로캘에는 차이가 있다. 이와 관련하여 **내장된 로캘과 라이브러리 로캘**을 참고한다.

로캘 데이터를 컴파일하려면 **make_locale** 스크립트(파일의 확장자는 Linux에선 **.sh**, Windows에선 **.bat**)를 사용한다. 이 스크립트는 **\$CUBRID/bin** 디렉터리에 위치하며, 이 경로가 **\$PATH** 환경 변수에 포함되어야 한다. 여기서 **\$CUBRID**, **\$PATH** 는 Linux의 환경 변수이며, Windows에서는 **%CUBRID%**, **%PATH%**이다.

참고

make_locale 스크립트를 Windows에서 실행하려면, Visual C++ 2005, 2008 또는 2010 중 하나가 설치되어 있어야 한다.

사용법은 **make_locale.sh -h** 명령(Windows는 **make_locale /h**)을 실행하면 출력되며, 사용 구문은 다음과 같다.

```
make_locale.sh [options] [locale]

options ::= [-t 32 | 64 ] [-m debug | release]
locale ::= [de_DE | es_ES | fr_FR | it_IT | ja_JP | km_KH | ko_KR |
tr_TR | vi_VN | zh_CN | ro_RO]
```

• options

- **-t**: 32비트 또는 64비트 중 하나를 선택한다(기본값: **64**).
- **-m**: **release** 또는 **debug** 중 하나를 선택한다. 일반적인 사용을 위해서는 **release**를 선택한다(기본값: **release**). **debug** 모드는 로캘 라이브러리를 직접 작성하려는 개발자를 위해 제공한다.
- **locale**: 빌드할 라이브러리의 로캘 이름. **locale** 이 주어지지 않으면, 설정한 모든 로캘의 데이터를 포함하도록 빌드된다. 이 경우 **\$CUBRID/lib** 디렉터리에 **libcubrid_all_locales.so** (Windows의 경우 **.dll**)라는 이름으로 라이브러리 파일이 저장된다.

여러 로캘에 대해서 사용자 정의 로캘 공유 라이브러리를 만들려면 다음 두 가지 방법 중 하나를 사용할 수 있다.

- 모든 로캘을 포함하는 하나의 라이브러리 생성: 다음과 같이 로캘을 명시하지 않고 실행한다.

```
make_locale.sh -t64 # Build and pack all
locales (64/release)
```

- 하나의 로캘만을 포함하는 라이브러리를 여러 개 반복하여 생성: 다음과 같이 하나의 언어를 지정한다.

```
make_locale.sh -t 64 -m release ko_KR
```

이와 같은 두 가지 방법 중에서 첫 번째 방법을 사용하는 것을 권장한다. 공유 라이브러리를 생성하면 로캘들 간에 공유될 수 있는 데이터들이 중복되지 않기 때문에 메모리 사용량을 줄일 수 있다. 첫 번째 방식으로 지원하는 모든 로캘을 포함하도록 생성하면 약 15MB 정도 크기의 라이브러리가 생성되며, 두 번째 방식으로 생성할 경우에는 언어에 따라서 1MB에서 5MB 이상의 크기의 라이브러리가 생성된다. 또한 첫 번째 방식에서는 두 번째 방식을 사용했을 때에 서버 재시작 시점 등에 발생하는 런타임 오버헤드가 없기 때문에 런타임에도 유리하다.

⚠ 경고

제약 사항 및 규칙

- 일단 로캘 라이브러리가 생성된 후에는 **\$CUBRID/conf/cubrid_locales.txt** 파일을 변경하면 안 된다. 다시 말해서, 이 파일에서 명시된 언어들의 순서를 포함하여 어떤 내용도 변경해서는 안 된다. 로캘이 컴파일되고 나면, 일반 콜레이션(generic collation)은 **cubrid_locales.txt**에 존재하는 첫번째 로캘을 기본 로캘로 사용한다. 따라서 순서를 바꾸면 해당 콜레이션(utf8_gen_*)에 대한 대소문자 변환 결과가 달라질 수 있다.
- **\$CUBRID/locales/data/*.txt** 파일들은 변경되어서는 안 된다.

i 참고

make_locale.sh(.bat) 스크립트 수행 절차

make_locale.sh(.bat) 스크립트는 다음과 같은 작업을 수행한다.

1. **\$CUBRID/locales/data/ducet.txt**, **\$CUBRID/locales/data/unicodedata.txt**, **\$CUBRID/locales/data/codepages/*.txt** 와 같이 이미 설치된 공통 파일과 해당 언어의 **.ldml** 파일을 읽는다.
2. 원본(raw) 데이터를 처리한 후 **\$CUBRID/locales/loclib/locale.c** 임시 파일에 로캘 데이터를 포함하는 C 상수 값과 배열을 작성한다.
3. **.so (.dll)** 파일을 빌드하기 위해 임시 파일인 **locale.c** 파일이 플랫폼 컴파일러에 전달된다. 이 단계는 장비가 C/C++ 컴파일러 및 링커를 가지고 있다고 가정한다. 현재 Windows 버전에서는 MS Visual Studio가, Linux 버전에서는 gcc 컴파일러가 사용된다.

4. 임시 파일이 삭제된다.

3단계: 특정 로캘을 사용하기 위해 CUBRID 설정하기

DB 생성 시 로캘 지정을 통해 오직 하나의 로캘을 기본 로캘로 지정할 수 있다.

기본 로캘을 지정하여 기본 캘린더가 정의되지만, **intl_date_lang** 시스템 파라미터를 설정하면 기본 로캘 설정보다 우선 적용된다.

- 로캘의 값은 `<locale_name>[.utf8 | .iso88591]`과 같이 설정한다. (예: tr_TR.utf8, en_US.iso88591, ko_KR.utf8)
- **intl_date_lang** 시스템 파라미터의 값은 `<locale_name>`과 같이 설정한다. `<locale_name>`으로 사용할 수 있는 값은 **1단계: 로캘 선택**을 참고한다.

참고

월, 요일, 오전/오후 표기 및 숫자 형식 설정

날짜/시간을 입출력하는 함수에서 각 로캘 이름에 따라 입출력하는 월, 요일, 오전/오후 표기 방법을 **intl_date_lang** 시스템 파라미터로 설정할 수 있다.

또한 문자열을 숫자로 또는 숫자를 문자열로 변환하는 함수에서 각 로캘에 따라 입출력하는 숫자의 문자열 형식은 **intl_number_lang** 시스템 파라미터로 설정할 수 있다.

내장된 로캘과 라이브러리 로캘

CUBRID에 내장된 로캘에 대해서는 사용자 로캘 라이브러리를 컴파일하지 않고 사용할 수 있으므로 2단계를 생략할 수 있으나, 내장된 로캘과 라이브러리 로캘에는 다음과 같은 차이가 있다.

- 내장된(built-in) 로캘(과 콜레이션)은 유니코드 데이터를 인식하지 못한다. 예를 들어, 내장된 로캘은 (Á, á) 간 대소문자 변환이 불가능하다. 반면 LDML 로캘(컴파일된 로캘)은 유니코드 코드포인트에 대한 데이터를 65535개까지 지원한다.
- 내장된 콜레이션은 ASCII 범위만 다루거나, utf8_tr_cs의 경우 ASCII와 터키어(turkish) 알파벳 글자만 다룬다. 따라서 내장된 UTF-8 로캘은 유니코드와 호환되지 않는 반면, LDML 로캘(컴파일된 로캘)은 유니코드와 호환된다.

DB 생성 시 설정할 수 있는 내장 로캘은 다음과 같다.

- en_US.iso88591

- en_US.utf8
- ko_KR.utf8
- ko_KR.euckr
- ko_KR.iso88591: 월, 요일 표시 방법은 로마자 표기를 따른다(romanized).
- tr_TR.utf8
- tr_TR.iso88591: 월, 요일 표시 방법은 로마자 표기를 따른다(romanized).

DB를 생성하면서 로캘 값 지정 시 문자셋(charset)이 명시되지 않으면 위 순서에서 앞에 있는 로캘의 문자셋으로 결정된다. 예를 들어, 로캘 값이 ko_KR로 설정되면(예: cubrid createdb testdb ko_KR) 위의 목록에서 ko_KR 중 가장 먼저 나타나는 로캘인 ko_KR.utf8을 지정한 것과 같다. 위의 내장된 로캘을 제외한 나머지 언어의 로캘은 뒤에 반드시 **.utf8** 을 붙여야 한다. 예를 들어, 독일어의 경우 로캘 값을 de_DE.utf8로 지정한다.

ko_KR.iso88591과 tr_TR.iso88591에서 월과 요일을 나타낼 때에는 로마자 표기를 따른다. 예를 들어, 한국어 “일요일”(영어로 Sunday)의 로마자 표기는 “Iryoil”이다. 이것은 ISO-8859-1 문자만 제공하기 위해서 요구되는 사항이다. 이에 관한 자세한 설명은 [ISO-8859-1 문자셋에서 한국어와 터키어의 월, 요일](#)를 참고하면 된다.

ISO-8859-1 문자셋에서 한국어와 터키어의 월, 요일

문자셋이 UTF-8인 한국어나 터키어 또는 문자셋이 EUC-KR인 한국어에서 월, 요일, 오전/오후 표시는 각 국가에 맞게 인코딩된다. 그러나, ISO-8859-1 문자셋에서 한국어와 터키어의 월, 요일, 오전/오후 표시를 원래의 인코딩으로 사용하면 복잡한 표현식이 사용되는 경우 서버 프로세스에서 예기치 않은 행동이 발생할 수 있기 때문에, 로마자 표기(romanized)로 출력한다. CUBRID의 기본 문자셋은 ISO-8859-1이며, 한국어와 터키어의 경우 이 문자셋을 사용할 수 있다. 한국어와 터키어에서 각 요일, 월, 오전/오후는 로마자로 다음과 같이 출력한다.

요일의 표기

긴/짧은 표기	한국어 긴/짧은 로마자 표기	터키어 긴/짧은 표기
Sunday / Sun	Iryoil / Il	Pazar / Pz
Monday / Mon	Woryoil / Wol	Pazartesi / Pt
Tuesday / Tue	Hwayoil / Hwa	Sali / Sa

긴/짧은 표기	한국어 긴/짧은 로마자 표기	터키어 긴/짧은 표기
Wednesday / Wed	Suyoil / Su	Carsamba / Ca
Thursday / Thu	Mogyoil / Mok	Persembе / Pe
Friday / Fri	Geumyoil / Geum	Cuma / Cu
Saturday / Sat	Toyoil / To	Cumartesi / Ct

월의 표기

긴 / 짧은 표기	한국어 긴/짧은 로마자 표기 (구분 없음)	터키어 긴 / 짧은 표기
January / Jan	1wol	Ocak / Ock
February / Feb	2wol	Subat / Sbt
March / Mar	3wol	Mart / Mrt
April / Apr	4wol	Nisan / Nsn
May / May	5wol	Mayis / Mys
June / Jun	6wol	Haziran / Hзr
July / Jul	7wol	Temmuz / Tmz
August / Aug	8wol	Agustos / Ags

긴 / 짧은 표기	한국어 긴/짧은 로마자 표기 (구분 없음)	터키어 긴 / 짧은 표기
September / Sep	9wol	Eylul / Eyl
October / Oct	10wol	Ekim / Ekm
November / Nov	11wol	Kasim / Ksm
December / Dec	12wol	Aralik / Arl

오전/오후의 표기

오전/오후 표기	한국어 로마자 표기	터키어 표기
AM	ojeon	AM
PM	ohu	PM

4단계: 선택한 로캘 설정으로 데이터베이스 생성하기

cubrid createdb <db_name> <locale_name.charset>을 실행하면, 해당 언어와 문자셋을 사용하는 데이터베이스가 생성된다. 일단 데이터베이스가 생성되면 해당 데이터베이스에 부여된 로캘 설정은 바꿀 수 없다.

문자셋과 로캘 이름은 **db_root**라는 시스템 카탈로그 테이블에 저장되며, 생성 시점의 설정과 다른 설정을 사용하여 데이터베이스를 구동할 수 없다.

5단계(선택 사항): 로캘 파일의 수동 검증

로캘 라이브러리의 내용들을 **dumplocale** 유틸리티를 이용해서 사람이 읽을 수 있는 형태로 출력할 수 있다. 사용법은 **cubrid dumplocale -h**로 출력할 수 있다.

```
cubrid dumplocale [options] [language-string]
options ::= -i|--input-file <shared_lib>
```

```

-d|--calendar
-n|--numeric
{-a|--alphabet={l|lower|u|upper|both}
-c|--codepoint-order
-w|--weight-order
{-s|--start-value} <starting_codepoint>
{-e|--end-value} <ending_codepoint>
-k
-z

```

```

language-string ::= de_DE | es_ES | fr_FR | it_IT | ja_JP | km_KH |
ko_KR | tr_TR | vi_VN | zh_CN | ro_RO

```

- **dumplocale**: 로캘 라이브러리에 설정된 내용을 텍스트로 출력하는 명령이다.
- *language-string*: de_DE, es_ES, fr_FR, it_IT, ja_JP, km_KH, ko_KR, tr_TR, vi_VN, zh_CN, ro_RO 중 하나의 값. 로캘 공유 라이브러리를 덤프할 로캘 언어를 지정한다. *language-string*이 생략되면 **cubrid_locales.txt** 파일에 명시된 모든 언어가 주어진다.

다음은 **cubrid dumplocale**에 대한 [options]이다.

-i, --input-file=FILE

로캘 공유 라이브러리 파일 이름을 지정하며, 경로를 포함한다.

-d, --calendar

캘린더와 날짜/시간 정보를 추가로 덤프한다.

-n, --numeric

숫자 정보를 덤프한다.

-a, --alphabet=l | lower | u | upper | both

알파벳과 대소문자 구분 정보를 덤프한다.

--identifier-alphabet=l | lower | u | upper

식별자에 대한 알파벳과 대소문자 구분 정보를 추가로 덤프한다.

-c, --codepoint-order

코드포인트 값을 기반으로 정렬한 콜레이션 정보를 추가로 덤프한다. 출력되는 정보는 cp, char, weight, next-cp, char, weight이다.

-w, --weight-order

가중치 값을 기반으로 정렬한 콜레이션 정보를 추가로 덤프한다. 출력되는 정보는 weight, cp, char이다.

-s, --start-value=CODEPOINT

덤프 범위의 시작을 지정한다. **-a**, **--identifier-alphabet**, **-c**, **-w** 옵션들에 대한 시작 코드포인트이며, 기본값은 0이다.

-e, --end-value=CODEPOINT

덤프 범위의 끝을 지정한다. **-a, --identifier-alphabet, -c, -w** 옵션들에 대한 끝 코드포인트이며, 기본값은 로캘 공유 라이브러리에서 읽은 최대값이다.

-k, --console-conversion

콘솔 변환 데이터를 추가로 덤프한다.

-z, --normalization

정규화 데이터를 추가로 덤프한다.

다음은 캘린더 정보, 숫자 표기 정보, 알파벳 및 대소문자 정보, 식별자에 대한 알파벳 및 대소문자 정보, 코드포인트 순서에 기반한 콜레이션의 정렬, 가중치에 기반한 콜레이션의 정렬, 데이터를 정규화하여 ko_KR 로캘의 내용을 ko_KR_dump.txt라는 파일에 덤프하는 예이다.

```
% cubrid dumplocale -d -n -a both -c -w -z ko_KR > ko_KR_dump.txt
```

여러 개의 옵션을 설정하면 출력되는 내용이 매우 많을 수 있으므로, 파일로 리다이렉션하여 저장할 것을 권장한다.

6단계: CUBRID 관련 프로세스 시작

모든 CUBRID 관련 프로세스는 같은 환경 설정을 통해 구동되어야 한다. CUBRID 서버, 브로커, CAS, CSQL 등은 같은 버전의 로캘 바이너리 파일을 사용해야 한다. CUBRID HA 구성 시에도 마찬가지이다. 예를 들어, CUBRID HA 구성에서 마스터 서버, 슬레이브 서버와 레플리카 서버 등은 환경 설정이 모두 같아야 한다.

서버 프로세스와 CAS 프로세스에 의해 사용되는 로캘의 호환성 여부를 시스템이 자동으로 검사하지 않기 때문에, 두 프로세스 간에 LDML 파일들이 똑같다는 것을 보장해야 한다.

로캘 라이브러리 로딩은 CUBRID 구동의 첫 단계로서, 구동 시에 데이터베이스 구조를 초기화하기 위해 로캘 정보를 요구하는 서버, CAS, CSQL, createdb, copydb, unloadb, loadb 프로세스 등은 구동 시점에 로캘 라이브러리를 로딩한다.

로캘 라이브러리 로딩 절차는 다음과 같다.

- 라이브러리 경로가 제공되지 않으면 `$CUBRID/lib/libcubrid_<lang_name>.so` 파일의 로딩을 시도한다. 이 파일이 발견되지 않으면 하나의 파일 `$CUBRID/lib/libcubrid_all_locales.so`에서 모든 로캘이 발견된다고 간주한다.
- 로캘 라이브러리가 발견되지 않거나 라이브러리를 로딩하는 동안 오류가 발생하면 CUBRID 프로세스 구동이 종료된다.
- 데이터베이스와 로캘 라이브러리 간 콜레이션 정보가 다르면 CUBRID 프로세스가 구동되지 않는다. 기존 데이터베이스에 로캘 라이브러리의 변경된 콜레이션을 포함하려면, 먼저 **cubrid synccolldb** 명령을 수행하여 데이터베이스 콜레이션을 로캘 라이브러리에 맞게 동기화한다. 다음

으로, 스키마와 데이터를 원하는 콜레이션에 맞게 기존 데이터베이스에 업데이트해야 한다. 자세한 내용은 **데이터베이스 콜레이션을 시스템 콜레이션에 동기화**를 참고한다.

데이터베이스 콜레이션을 시스템 콜레이션에 동기화

CUBRID가 정상적으로 동작하기 위해서는 시스템 콜레이션과 데이터베이스 콜레이션이 같아야 한다. 시스템 로캘은 내장된 로캘과 cubrid_locales.txt 파일을 통해(**로캘 설정** 참고) 생성한 라이브러리 로캘을 포함한 로캘을 의미하며, 시스템 로캘은 시스템 콜레이션 정보를 포함한다. 데이터베이스 콜레이션 정보는 **_db_collation** 시스템 카탈로그 테이블에 저장된다.

cubrid synccolldb 유틸리티는 데이터베이스 콜레이션이 시스템 콜레이션과 일치하는지 확인하고, 다를 경우 데이터베이스 콜레이션을 시스템 콜레이션에 동기화하는 유틸리티이다. 하지만, 이 유틸리티는 데이터베이스 서버에 저장된 데이터 자체를 변환하지 않음을 인지해야 한다.

이 유틸리티는 시스템 로캘이 변경된 이후 기존의 데이터베이스 콜레이션 정보를 변경해야 할 때 사용할 수 있다. 단, 사용자가 직접 수동으로 진행해야 하는 작업들이 있다.

동기화하기 전에 다음과 같은 작업을 수행한다. **cubrid synccolldb -c** 명령을 수행하여 생성되는 cubrid_synccolldb_<database_name>.sql 파일을 CSQL을 통해 실행하면 된다.

- ALTER TABLE MODIFY 문을 사용하여 콜레이션을 수정한다.
- 콜레이션을 포함하는 뷰, 인덱스, 트리거, 분할(partition) 등을 모두 제거한다.

cubrid synccolldb를 가지고 동기화를 수행한다. 그리고 아래 작업을 수행한다.

- 뷰, 인덱스, 트리거, 분할 등을 재생성한다.
- 새로운 콜레이션에 맞게 응용 프로그램의 질의문들을 업데이트한다.

이 유틸리티는 데이터베이스를 정지한 상태에서 수행해야 한다.

synccolldb 구문은 다음과 같다.

```
cubrid synccolldb [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **synccolldb**: 데이터베이스 콜레이션을 시스템 콜레이션(로캘 라이브러리의 내용과 \$CUBRID/conf/cubrid_locales.txt 파일을 따름)으로 동기화하는 명령이다.
- **database_name**: 콜레이션 정보가 로캘 라이브러리의 콜레이션에 맞게 동기화될 데이터베이스의 이름이다.

[options]를 생략하면 시스템과 데이터베이스 간 콜레이션 차이를 출력하고, 동기화하기 전에 삭제되어야 할 객체 질의문을 포함하는 cubrid_synccolldb_<database_name>.sql 파일을 생성한다.

다음은 **cubrid synccolldb**에서 사용하는 [options]이다.

-c, --check-only

데이터베이스의 콜레이션과 시스템의 콜레이션을 확인하여 불일치하는 콜레이션 정보를 출력한다.

-f, --force-only

데이터베이스에 있는 콜레이션 정보를 시스템에서 설정한 콜레이션과 동일하게 업데이트할 때 업데이트 여부를 질문하지 않는다.

다음의 예는 시스템 콜레이션과 데이터베이스의 콜레이션이 서로 다를 때 어떻게 동작하는지를 보여준다.

먼저 ko_KR 로캘에 대한 로캘 라이브러리를 생성한다.

```
$ echo ko_KR > $CUBRID/conf/cubrid_locales.txt
$ make_locale.sh -t 64
```

다음으로 데이터베이스를 생성한다.

```
$ cubrid createdb --db-volume-size=20M --log-volume-size=20M xdb
en_US
```

스키마를 생성한다. 이때, 각 테이블에 원하는 콜레이션을 지정한다.

```
$ csq1 -S -u dba xdb -i in.sql
```

```
CREATE TABLE dept (depname STRING PRIMARY KEY) COLLATE
utf8_ko_cs_uca;
CREATE TABLE emp (eid INT PRIMARY KEY, depname STRING, address
STRING) COLLATE utf8_ko_cs_uca;
ALTER TABLE emp ADD CONSTRAINT FOREIGN KEY (depname) REFERENCES
dept(depname);
```

시스템의 로캘 설정을 변경한다. **cubrid_locales.txt**에 아무런 값도 설정하지 않으면 데이터베이스에는 내장된 로캘만 존재하는 것으로 간주한다.

```
$ echo "" > $CUBRID/conf/cubrid_locales.txt
```

cubrid synccolldb -c 명령을 수행하여 시스템과 데이터베이스 간 콜레이션 차이를 확인한다.

```
$ cubrid synccolldb -c xdb
```

```

-----
Collation 'utf8_ko_cs_uca' (Id: 133) not found in database or
changed in new system configuration.
-----
Collation 'utf8_gen_ci' (Id: 44) not found in database or changed in
new system configuration.
-----
Collation 'utf8_gen_ai_ci' (Id: 37) not found in database or changed
in new system configuration.
-----
Collation 'utf8_gen' (Id: 32) not found in database or changed in
new system configuration.
-----
There are 4 collations in database which are not configured or are
changed compared to system collations.
Synchronization of system collation into database is required.
Run 'cubrid synccolldb -f xdb'

```

인덱스가 존재한다면 먼저 인덱스를 제거한 후 각 테이블의 콜레이션을 변환하고, 이후 인덱스 생성을 직접 수행해야 한다. 인덱스를 제거하고 테이블의 콜레이션을 변환하는 과정은 **synccolldb**에서 생성된 **cubrid_synccolldb_xdb.sql** 파일로 수행할 수 있다. 다음 예에서는 외래 키가 재생성해야 될 인덱스에 해당한다.

```

$ cat cubrid_synccolldb_xdb.sql

ALTER TABLE [dept] COLLATE utf8_bin;
ALTER TABLE [emp] COLLATE utf8_bin;
ALTER TABLE [emp] DROP FOREIGN KEY [fk_emp_depname];
ALTER TABLE [dept] MODIFY [depname] VARCHAR(1073741823) COLLATE
utf8_bin;
ALTER TABLE [emp] MODIFY [address] VARCHAR(1073741823) COLLATE
utf8_bin;
ALTER TABLE [emp] MODIFY [depname] VARCHAR(1073741823) COLLATE
utf8_bin;

$ csql -S -u dba -i cubrid_synccolldb_xdb.sql xdb

```

시스템 콜레이션을 데이터베이스에 동기화하기 전에 위의 **cubrid_synccolldb_xdb.sql** 스크립트 파일을 실행하여 예전의 콜레이션들을 삭제해야 한다.

cubrid synccolldb 명령을 수행한다. 옵션을 생략하면 해당 명령을 수행할 것인지를 확인하는 메시지가 나타나며, **-f** 옵션을 주면 확인 과정 없이 데이터베이스와 시스템 간 콜레이션 동기화를 수행한다.

```
$ cubrid synccolldb xdb
Updating system collations may cause corruption of database.
Continue (y/n) ?
Contents of '_db_collation' system table was updated with new system
collations.
```

DROP된 외래 키를 다시 생성한다.

```
$ csqL -S -u dba xdb

ALTER TABLE emp ADD CONSTRAINT FOREIGN KEY fk_emp_depname(depname)
REFERENCES dept(depname);
```

참고

CUBRID에서 콜레이션은 CUBRID 서버에 의해 숫자 ID로 인식되며, ID의 범위는 0부터 255까지이다. LDML 파일은 공유 라이브러리로 컴파일되는데, 콜레이션 ID와 콜레이션(이름, 속성)의 매핑 정보를 제공한다.

- 시스템 콜레이션은 CUBRID 서버와 CAS 모듈에 의해 로컬 라이브러리로부터 로딩되는 콜레이션이다.
- 데이터베이스 콜레이션은 **_db_collation** 시스템 테이블에 저장되는 콜레이션이다.

콜레이션 설정

콜레이션(collation)이란 문자열 비교 및 정렬 규칙의 집합이다. 콜레이션의 전형적인 예는 알파벳 순서의 정렬(alphabetization)이다.

테이블 생성 시에 칼럼의 문자셋과 콜레이션이 명시되지 않으면, 칼럼은 테이블의 문자셋과 콜레이션을 따른다. 문자셋과 콜레이션 설정은 기본적으로 클라이언트의 설정을 따른다. 표현식 결과가 문자열 데이터이면 표현식의 피연산자를 감안한 콜레이션 추론 과정을 통하여 문자셋과 콜레이션을 결정한다.

참고

CUBRID는 유럽과 아시아 언어를 포함한 여러 가지 언어들의 콜레이션을 지원한다. 이러한 언어들은 다른 알파벳들을 사용할 뿐만 아니라, 특정 언어들은 일부 문자셋에 대해 확장(expansion) 또는 축약(contraction) 정의를 필요로 한다. 이러한 사항들의 대부분은 The Unicode Consortium에 의해 유니코드 표준(2012년 현재 버전 6.1.0)으로 제정되어 있으며, 대부분의 언어가 요구하는

모든 문자 정보는 DUCET 파일(<http://www.unicode.org/Public/UCA/latest/allkeys.txt>)에 저장되어 있다.

이러한 DUCET에 표현된 대부분의 코드포인트는 0~FFFF 내의 범위에 포함되지만, 이 범위를 넘는 코드포인트도 존재한다. 하지만 CUBRID는 0~FFFF 내의 코드포인트만 사용하고, 나머지들은 무시한다(하위 부분만 사용하도록 설정할 수도 있다).

DUCET에 있는 각각의 코드포인트는 하나 또는 그 이상의 콜레이션 원소(element)를 가지고 있다. 하나의 콜레이션 원소는 네 개 숫자 값의 집합으로, 문자 비교의 네 가지 수준(level)을 가중치(weight)로 표현한다. 각각의 가중치 값은 0~FFFF의 범위를 가진다.

DUCET에서 한 문자는 하나의 라인으로 다음과 같이 표현된다.

```
< codepoint_or_multiple_codepoints > ; [.W1.W2.W3.W4][.....].... #
< readable text explanation of the symbol/character >
```

한국어 문자 기억은 다음과 같이 표현된다.

```
1100 ; [.313B.0020.0002.1100] # HANGUL CHOSEONG KIYEOK
```

위의 예에서 1100은 코드포인트, [.313B.0020.0002.1100]은 하나의 콜레이션 원소이며, 313B는 Level 1, 0020은 Level 2, 0002는 Level 3, 1100은 Level 4의 가중치이다.

언어의 기능적 속성으로 정의되는 확장 지원은 하나의 결합 문자를 그것을 만드는 한 쌍의 문자들로 해석하도록 지원한다는 것을 의미한다. 예를 들어, 한 문자 'æ'를 두 개의 문자 'ae'와 같은 문자로 해석한다. DUCET에서 확장은 하나의 코드포인트나 축약에 대해 하나 이상의 콜레이션 원소들로 표현된다. 확장이 있는 콜레이션을 다루는 것은 두 개의 문자열을 비교할 때 콜레이션의 세기/수준까지 여러 번 비교하는 비용을 감수해야 하기 때문에, CUBRID는 기본적으로는 확장을 지원하지 않도록 설정되어 있다.

칼럼의 문자셋과 콜레이션

칼럼의 문자셋과 콜레이션은 문자열 데이터 타입(**VARCHAR**, **CHAR**)과 **ENUM** 타입에 적용된다. 기본적으로 모든 문자열 데이터 타입은 데이터베이스의 기본 문자셋과 콜레이션을 따르는데, 이를 변경하여 지정할 수 있는 방법을 제공한다.

문자셋

문자셋은 문자열 리터럴이나 따옴표 없는 식별자(identifier)로 명시될 수 있으며, 지원하는 문자셋은 다음과 같다.

- ISO-8859-1
- UTF-8 (문자당 최대 4 바이트 길이, 즉 0~0x10FFFF 범위 내의 코드포인트를 지원)
- EUC-KR (이 문자셋은 하위 호환을 위해서 존재할 뿐 사용을 권장하지 않는다.)

참고

CUBRID 9.0 미만 버전까지는 ISO-8859-1 문자셋이 설정되면 EUC-KR 문자들을 사용할 수 있도록 지원했지만, 이후 버전부터는 이를 지원하지 않는다. EUC-KR 문자들은 오직 EUC-KR 문자셋에서만 사용될 수 있다.

문자열 검사

기본적으로 모든 입력 데이터는 DB 생성 시 설정한 문자로 간주한다. 하지만 **SET NAMES** 문이나 문자셋 소개자(또는 **COLLATE** 문자열 수정자)가 DB 생성 시 설정한 로캘보다 우선한다([문자열 리터럴의 문자셋과 콜레이션](#) 참고).

서버 문자셋이 UTF-8인데 UTF-8 바이트 순서(byte sequence)에 맞지 않는 데이터와 같이 무효한 데이터에 대해 문자열을 검사하지 않으면 정의되지 않은 동작을 보이거나 심지어 서버가 비정상 종료(crash)될 수 있다. 기본적으로는 문자열을 검사하지 않도록 설정되어 있다. 문자열을 검사하려면 **intl_check_input_string** 시스템 파라미터의 값을 yes로 설정한다(기본값: no). 하지만 유효한 데이터만 입력된다고 보장할 수 있다면, 문자열 검사는 하지 않는 것이 성능상 더 유리하다. **intl_check_input_string** 시스템 파라미터의 값이 yes인 경우, UTF-8과 EUC-KR 문자셋에 대해서만 유효한 데이터 인코딩인지 검사한다. ISO-8859-1 문자셋은 한 바이트 인코딩이므로 모든 바이트 값이 유효하기 때문에 검사하지 않는다.

문자셋 변환

콜레이션/문자셋 수정자(**COLLATE** / **CHARSET**) 또는 콜레이션 추론 과정에 의해서 문자셋 변환이 일어날 수 있는데, 이러한 문자셋 변환은 비가역적(irreversible)이다. 일반적으로, 문자셋 변환은 문자 트랜스코딩(transcoding)이다(어떤 문자셋에 있는 문자를 표현하는 바이트 값이 대상 문자셋에서 같은 문자를 나타내지만 다른 바이트 값으로 대체된다).

어떤 변환이든, 손실이 발생할 수 있다. 원본 문자셋에 있는 문자를 대상 문자셋으로 인코딩할 수 없으면 '?' 문자로 대체된다. 바이너리 문자셋을 다른 문자셋으로 변환할 때도 마찬가지이다. 가장 많은 문자를 지원하는 문자셋은 UTF-8이며 유니코드를 인코딩하므로 모든 문자가 인코딩될 수 있다. 그러나 ISO-8859-1에서 UTF-8로 변환하면 일부 "손실"이 발생할 수도 있다. 예를 들어 80-A0 범위의

바이트 값은 ISO-8859-1 문자에서 유효한 값은 아니지만 문자열에 나타날 수도 있다. 이 문자들을 UTF-8로 변환하면 '?'로 대체된다.

한 문자에서 다른 문자로 변환되는 규칙은 다음과 같다.

Source Destination	\	Binary	ISO-8859-1	UTF-8	EUC-KR
Binary		변환 없음	변환 없음	변환 없음. 문자당 유효성 검증. 유효하지 않은 문자 '?'로 대체	변환 없음. 문자당 유효성 검증. 유효하지 않은 문자 '?'로 대체
ISO-8859-1		변환 없음	변환 없음	바이트 변환. 바이트 크기가 증가됨. 사용 가능한 문자 손실은 없음.	바이트 변환. 바이트 크기가 증가됨. 사용 가능한 문자 손실은 없음.
UTF-8		변환 없음. 바이트 크기는 변화 없음. 문자 길이는 증가함.	바이트 재해석. 바이트 크기가 감소될 수 있음. 문자 손실이 예상됨.	변환 없음	바이트 변환. 바이트 크기가 감소될 수 있음. 문자 손실이 예상됨.
EUC-KR		변환 없음. 바이트 크기는 변화 없음. 문자 길이는 증가함.	바이트 변환. 바이트 크기가 감소될 수 있음. 문자 손실이 예상됨.	바이트 변환. 바이트 크기가 증가될 수 있음. 사용 가능한 문자 손실은 없음.	변환 없음

참고

CUBRID 9.x 이전 버전은 바이너리 문자셋을 지원하지 않았으며, 대신 ISO-8859-1 문자셋이 기본 바이너리 문자셋 기능을 했다. UTF-8 및 EUC-KR 문자셋에서 ISO-8859-1로 변환하는 경우 문자 변환이 아니라 원본의 콘텐츠(바이트)를 재해석하여 변환을 수행한다.

콜레이션

콜레이션은 문자열 리터럴이나 따옴표 없는 식별자로 명시될 수 있다.

다음은 내장된(built-in) 콜레이션에 대한 **db_collation** 시스템 카탈로그 뷰의 질의(SELECT * FROM db_collation WHERE is_builtin='Yes') 결과이다.

```

coll_id coll_name      charset_name  is_builtin
has_expansions  contractions uca_strength
=====
=====
0         'iso88591_bin'      'iso88591'   'Yes'
'No'         0          'Not applicable'
1         'utf8_bin'       'utf8'       'Yes'
'No'         0          'Not applicable'
2         'iso88591_en_cs' 'iso88591'   'Yes'
'No'         0          'Not applicable'
3         'iso88591_en_ci' 'iso88591'   'Yes'
'No'         0          'Not applicable'
4         'utf8_en_cs'   'utf8'       'Yes'
'No'         0          'Not applicable'
5         'utf8_en_ci'   'utf8'       'Yes'
'No'         0          'Not applicable'
6         'utf8_tr_cs'   'utf8'       'Yes'
'No'         0          'Not applicable'
7         'utf8_ko_cs'   'utf8'       'Yes'
'No'         0          'Not applicable'
8         'euckr_bin'    'euckr'      'Yes'
'No'         0          'Not applicable'
9         'binary'      'binary'     'Yes'
'No'         0          'Not applicable'

```

내장된 콜레이션은 사용자 로컬 라이브러리의 추가 없이 사용 가능하며, 각 콜레이션은 관련 문자셋을 가지고 있기 때문에 문자셋과 콜레이션이 호환되도록 지정해야 한다.

COLLATE 수정자가 **CHARSET** 수정자 없이 명시되면, 콜레이션의 기본 문자셋이 설정된다. **CHARSET** 수정자가 **COLLATE** 수정자 없이 명시되면, 기본(default) 콜레이션이 설정된다.

문자셋들에 대한 기본 콜레이션은 바이너리 콜레이션으로, 문자셋 및 이에 대응되는 바이너리 콜레이션은 다음과 같다.

- ISO-8859-1: iso88591_bin
- UTF-8: utf8_bin
- EUC-KR: euckr_bin
- Binary: binary

Binary는 콜레이션과 관련 문자셋의 이름이다.

서로 다른 콜레이션(과 문자셋)을 가진 표현식 인자(피연산자)를 가질 때 어떤 콜레이션을 사용할지 결정하는 방법에 대해서는 **콜레이션이 서로 다를 때 결정 방식** 을 참고한다.

CHARSET과 COLLATE 수정자

기본 데이터베이스 콜레이션과 문자셋을 따르지 않고 콜레이션과 문자셋을 변경하여 지정할 수 있는 문자열 타입에 대한 수정자를 제공한다.

- **CHARACTER_SET** (또는 **CHARSET**) 수정자는 칼럼의 문자셋을 바꾼다.
- **COLLATE** 수정자는 칼럼의 콜레이션을 바꾼다.

```
<data_type> ::= <column_type> [<charset_modifier_clause>]
[<collation_modifier_clause>]

<charset_modifier_clause> ::= {CHARACTER_SET | CHARSET}
{<char_string_literal> | <identifier> }

<collation_modifier_clause> ::= COLLATE {<char_string_literal> |
<identifier> }
```

다음은 **VARCHAR** 타입 칼럼의 문자셋을 UTF-8로 설정하는 예이다.

```
CREATE TABLE t1 (s1 VARCHAR (100) CHARSET utf8);
```

다음은 칼럼 s1의 이름을 c1으로 바꾸고, 해당 타입을 콜레이션이 utf8_en_cs인 CHAR(10) 으로 바꾸는 예이다. 문자셋은 해당 콜레이션에 대한 기본 문자셋인 UTF-8으로 지정된다.

```
ALTER TABLE t1 CHANGE s1 c1 CHAR(10) COLLATE utf8_en_cs;
```

다음은 c1 칼럼의 값을 콜레이션 iso88591_en_ci인 VARCHAR(5) 타입으로 바꿔 출력한다. 정렬 연산 또한 첫번째로 선택된 칼럼의 타입에 대한 콜레이션 iso88591_en_ci을 사용하여 수행된다.

```
SELECT CAST (c1 as VARCHAR(5) COLLATE 'iso88591_en_ci') FROM t1
ORDER BY 1;
```

다음은 위와 유사한 질의(같은 정렬)이지만, 출력되는 칼럼 결과가 원래의 값이다.

```
SELECT c1 FROM t1 ORDER BY CAST (c1 as VARCHAR(5) COLLATE
iso88591_en_ci);
```

콜레이션이 서로 다를 때 결정 방식

```
CREATE TABLE t (
  s1 STRING COLLATE utf8_en_cs,
  s2 STRING COLLATE utf8_tr_cs
);

-- insert values into both columns

SELECT s1, s2 FROM t WHERE s1 > s2;
```

```
ERROR: '>' requires arguments with compatible collations.
```

위의 예에서 칼럼 s1과 s2 는 다른 콜레이션을 가지고 있고, s1과 s2 를 비교한다는 것은 테이블 t에 있는 레코드끼리 어떤 칼럼의 값이 “더 큰지” 결정할 수 있는 문자열을 비교한다는 것을 의미한다. 콜레이션 utf8_en_cs와 utf8_tr_cs는 서로 비교할 수 없으므로 이 경우에는 에러를 출력할 것이다.

표현식의 타입 결정 방법의 원칙이 콜레이션 결정 방법에도 마찬가지로 적용된다.

1. 표현식의 모든 인자들을 고려하여 공통 콜레이션과 문자셋을 결정한다.
2. 1.에서 결정된 공통 콜레이션(또는 문자셋)과 다른 인자들을 변환한다.
3. 콜레이션을 변경하기 위해서 `CAST()` 연산자가 사용될 수 있다.

비교 표현식의 결과 콜레이션을 결정하기 위해 “콜레이션 변환도(collation coercibility)”를 사용한다. 이는 자신의 콜레이션이 얼마나 쉽게 상대 인자의 콜레이션으로 변환되기 쉬운가를 표현한 것으로, 표현식의 두 피연산자를 비교할 때 콜레이션 변환도가 크다는 것은 상대 인자의 콜레이션으로 쉽게 변환된다는 것을 의미한다. 즉, 높은 변환도를 지닌 인자는 더 낮은 변환도를 지닌 인자의 콜레이션으로 변환될 수 있다.

표현식의 인자들이 서로 다른 콜레이션을 가지면, 이들에 대한 공통 콜레이션은 각 인자들의 콜레이션과 변환도에 기반하여 결정된다.

1. 높은 변환도를 가진 인자는 더 낮은 변환도를 가진 인자의 콜레이션으로 변환된다.
2. 인자들의 콜레이션이 서로 다르고 변환도가 같은 경우에는 표현식의 콜레이션을 결정할 수 없고 에러가 리턴된다. 단, 콜레이션 변환도 11(호스트 변수, 사용자 정의 변수)이고 동일한 문자셋인 두 피연산자를 비교하는 경우 둘 중 하나가 바이너리 콜레이션(utf8_bin, iso88591_bin, euckr_bin)이면 바이너리가 아닌(non-binary) 콜레이션으로 변환된다. **세션 변수와(또는) 호스트 변수의 비교**를 참고한다.

표현식 인자들의 변환도는 다음의 표와 같다.

콜레이션 변환도	표현식의 인자(피연산자)
-1	호스트 변수만을 가지는 표현식으로, 실행단계 이전에는 콜레이션 변환도를 결정할 수 없는 경우.
0	COLLATE 수정자를 가진 피연산자
1	Columns 이 바이너리가 아닌 (non-binary) 콜레이션을 가진 경우
2	Columns 이 바이너리 콜레이션과 ISO-8859-1 문자셋(iso88591_bin)을 가진 경우
3	Columns 이 ISO-8859-1 문자셋을 가진 경우를 제외하고 바이너리 콜레이션을 가진 경우
4	SELECT 값, 표현식이 바이너리가 아닌 콜레이션을 가진 경우
5	SELECT 값, 표현식이 바이너리 콜레이션과 ISO-8859-1 문자셋(iso88591_bin)을 가진 경우
6	SELECT 값, 표현식이 ISO-8859-1 문자셋을 가진 경우를 제외하고 바이너리 콜레이션을 가진 경우
7	특수 함수들 (<code>SYSTEM_USER()</code> , <code>DATABASE()</code> , <code>SCHEMA()</code> , <code>VERSION()</code>)
8	상수 문자열 이 바이너리가 아닌(non-binary) 콜레이션을 가진 경우
9	

콜레이션 변환도

표현식의 인자(피연산자)

상수 문자열 이 바이너리 콜레이션과 ISO-8859-1 문자셋(iso88591_bin)을 가진 경우

10

상수 문자열이 ISO-8859-1 문자셋을 가진 경우를 제외하고 바이너리 콜레이션을 가진 경우

11

호스트 변수, 사용자 정의 변수

 참고

9.x 버전에서는 바이너리 콜레이션을 지원하지 않았으며, 대신 iso88591_bin 콜레이션이 기본 바이너리 콜레이션 기능을 했다. 버전 10.0부터는 iso88591_bin이 포함된 컬럼의 변환도(coercibility)가 2에서 3으로 강등되었고, iso88591_bin이 포함된 표현식에서는 5에서 6으로, iso88591_bin이 포함된 상수에서는 9에서 10으로 강등되었다.

호스트 변수만을 인자로 가지는 표현식(예: 아래의 UPPER(?))에 대해서는 실행 단계에서 콜레이션 변환도를 결정할 수 있다. 즉, 파싱 단계에서는 이러한 표현식의 콜레이션 변환도를 결정할 수 없으므로 COERCIBILITY 함수는 -1을 반환한다.

```
SET NAMES utf8
PREPARE st FROM 'SELECT COLLATION(UPPER(?)) col1,
COERCIBILITY(UPPER(?)) col2';
EXECUTE st USING 'a', 'a';
```

```
col1                                col2
=====
'utf8_bin'                          -1
```

모든 인자의 변환도가 11이고 콜레이션이 서로 다른 표현식에서는 실행중에 공통 콜레이션이 확인된다(이것은 유추를 위한 변환도 값 기반 규칙의 예외 사항이며 오류를 유발할 것이다.)

```
PREPARE st1 FROM 'SELECT INSERT(?,2,2,?)';
EXECUTE st1 USING _utf8'abcd', _binary'ef';
```

```
insert( ?:0 , 2, 2, ?:1 )
=====
'aefd'
```

콜레이션이 서로 다른 두 개의 인자가 하나의 콜레이션으로 변환되는 경우를 살펴보면 다음과 같다.

· 원하는 콜레이션을 지정하여 변환

앞의 예제에서 실행에 실패한 **SELECT** 문은 다음 질의문처럼 한 칼럼에 **CAST** 연산자로 콜레이션을 지정하여 두 피연산자를 같은 콜레이션을 갖도록 하면 성공적으로 수행된다.

```
SELECT s1, s2 FROM t WHERE s1 > CAST (s2 AS STRING COLLATE
utf8_en_cs);
```

또한, s2를 바이너리 콜레이션으로 **CAST** 하면, s1의 콜레이션으로 변환도 (6)은 s1의 변환도 (1)보다 높다.

```
SELECT s1, s2 FROM t WHERE s1 > CAST (s2 AS STRING COLLATE
utf8_bin);
```

다음과 같은 질의문에서 두 번째 피연산자 “CAST (s2 AS STRING COLLATE utf8_tr_cs)”는 서브 표현식이고, 서브 표현식은 칼럼(s1)보다 더 높은 변환도를 가지기 때문에, “CAST (s2 AS STRING COLLATE utf8_tr_cs)”는 s1의 콜레이션으로 변환된다.

```
SELECT s1, s2 FROM t WHERE s1 > CAST (s2 AS STRING COLLATE
utf8_tr_cs);
```

어떤 표현식이든 표현식은 칼럼보다 높은 변환도를 갖는다.

```
SELECT s1, s2 FROM t WHERE s1 > CONCAT (s2, '');
```

· 상수와 칼럼의 콜레이션 변환

다음의 경우 칼럼 s1의 콜레이션을 사용하여 비교가 수행된다.

```
SELECT s1, s2 FROM t WHERE s1 > 'abc';
```

· 칼럼이 Bin 콜레이션으로 생성되는 경우

```
CREATE TABLE t (
  s1 STRING COLLATE utf8_en_cs,
  s2 STRING COLLATE utf8_bin
```

```
);
SELECT s1, s2 FROM t WHERE s1 > s2;
```

위 경우 s2 칼럼의 변환도는 6(Bin 콜레이션)으로, s1 칼럼의 콜레이션으로 “완전히 변환 가능”하여 utf8_en_cs 콜레이션이 사용된다.

```
CREATE TABLE t (
  s1 STRING COLLATE utf8_en_cs,
  s2 STRING COLLATE iso88591_bin
);
SELECT s1, s2 FROM t WHERE s1 > s2;
```

위 경우에도 마찬가지로 콜레이션으로 utf8_en_cs가 사용되는데, s2 칼럼이 ISO 문자셋이므로 UTF-8로 변환하는 오버헤드가 발생한다는 차이가 있다.

다음 질의문에서 문자셋 변환은 발생하지 않고 s2 칼럼의 UTF-8의 바이트 데이터는 간단하게 ISO-8859-1 문자셋으로 재해석되며, iso88591_en_cs 콜레이션을 사용하여 문자 비교만 수행된다.

```
CREATE TABLE t (
  s1 STRING COLLATE iso88591_en_cs,
  s2 STRING COLLATE utf8_bin
);
SELECT s1, s2 FROM t WHERE s1 > s2;
```

· 서브 표현식과 칼럼의 콜레이션 변환

```
CREATE TABLE t (
  s1 STRING COLLATE utf8_en_cs,
  s2 STRING COLLATE utf8_tr_cs
);
SELECT s1, s2 FROM t WHERE s1 > s2 + 'abc';
```

위 경우 두 번째 피연산자는 표현식이기 때문에 s1의 콜레이션이 사용된다.

다음 예제는 서로 다른 콜레이션을 지닌 s2와 s3에 대해 ‘+’ 연산을 수행하려고 하기 때문에 에러가 발생한다.

```
CREATE TABLE t (
  s1 STRING COLLATE utf8_en_cs,
  s2 STRING COLLATE utf8_tr_cs,
  s3 STRING COLLATE utf8_en_ci
);

SELECT s1, s2 FROM t WHERE s1 > s2 + s3;
```

```
ERROR: '+' requires arguments with compatible collations.
```

다음 예제에서는 s2와 s3가 같은 콜레이션이므로 '+' 표현식이 utf8_tr_cs이 되고, s1은 칼럼이므로 표현식보다 낮은 변환도를 갖기 때문에 비교 연산은 utf8_en_cs 콜레이션을 사용해서 수행된다.

```
CREATE TABLE t (  
    s1 STRING COLLATE utf8_en_cs,  
    s2 STRING COLLATE utf8_tr_cs,  
    s3 STRING COLLATE utf8_tr_cs  
);  
  
SELECT s1, s2 FROM t WHERE s1 > s2 + s3;
```

· 숫자, 날짜 타입 상수의 콜레이션 변환

연산 과정에서 문자열로 변환 가능한 숫자 또는 날짜 타입 상수는 항상 상대 문자열의 콜레이션으로 변환될 수 있다.

· 세션 변수와(또는) 호스트 변수의 콜레이션 변환

콜레이션 변환도가 11(세션 변수, 호스트 변수)이고 문자셋이 동일한 피연산자끼리 비교하는 경우 바이너리가 아닌(non-bin) 콜레이션으로 변환된다.

```
SET NAMES utf8;  
SET @v1='a';  
PREPARE stmt FROM 'SELECT COERCIBILITY(?), COERCIBILITY(@v1),  
COLLATION(?), COLLATION(@v1), ? = @v1';  
SET NAMES utf8 COLLATE utf8_en_ci;  
EXECUTE stmt USING 'A', 'A', 'A';
```

@v1과 'A'를 비교하면 @v1은 바이너리가 아닌 콜레이션인 utf8_en_ci로 변환되어 @v1과 'A'가 서로 동일하게 되므로, 아래처럼 "? = @v1"의 결과는 1이 된다.

```
coercibility( ?:0 )   coercibility(@v1)   collation( ?:1 )  
collation(@v1)       ?:2 =@v1  
=====
```

	11	11	'utf8_en_ci'
'utf8_bin'		1	

```
SET NAMES utf8 COLLATE utf8_en_cs;  
EXECUTE stmt USING 'A', 'A', 'A';
```


@v1과 'A'를 비교하면 @v1은 바이너리가 아닌 콜레이션인 utf8_en_cs로 변환되어 @v1과 'A'가 서로 다르게 되므로, 아래처럼 "? = @v1"의 결과는 0이 된다.

```
coercibility( ?:0 )   coercibility(@v1)   collation( ?:1 )
collation(@v1)        ?:2 =@v1
=====
=====
                        11                11  'utf8_en_cs'
'utf8_bin'            0
```

그러나, 아래와 같이 @v1과 'A'의 콜레이션이 둘 다 바이너리가 아닌 콜레이션이고 두 콜레이션이 서로 다르면 에러가 발생한다.

```
DEALLOCATE PREPARE stmt;
SET NAMES utf8 COLLATE utf8_en_ci;
SET @v1='a';
PREPARE stmt FROM 'SELECT COERCIBILITY(?), COERCIBILITY(@v1),
COLLATION(?), COLLATION(@v1), ? = @v1';
SET NAMES utf8 COLLATE utf8_en_cs;
EXECUTE stmt USING 'A', 'A', 'A';
```

```
ERROR: Context requires compatible collations.
```

ENUM 타입 칼럼의 문자셋과 콜레이션

ENUM 타입 칼럼의 문자셋과 콜레이션은 DB 생성 시 지정한 로캘을 따른다.

예를 들어, en_US.iso88591로 DB를 생성하여 다음 테이블을 생성한다.

```
CREATE TABLE tbl (e ENUM (_utf8'a', _utf8'b'));
```

위에서 생성된 테이블의 칼럼 'e'는 그 원소의 문자셋이 UTF8로 정의되어 있더라도 ISO88591 문자셋, iso88591_bin 콜레이션을 가진다. 문자열 원소는 칼럼이 생성될 때 UTF8 문자셋에서 ISO88591 문자셋으로 변환된다. 사용자가 다른 문자셋이나 콜레이션을 적용하고 싶으면 테이블의 칼럼에 대해 이를 명시해야 한다.

아래는 테이블의 칼럼에 대해 콜레이션을 명시하는 예이다.

```
CREATE TABLE t (e ENUM (_utf8'a', _utf8'b') COLLATE utf8_bin);
CREATE TABLE t (e ENUM (_utf8'a', _utf8'b')) COLLATE utf8_bin;
```

테이블의 문자셋과 콜레이션

테이블 생성 구문에 문자셋과 콜레이션을 지정할 수 있다.

```
CREATE TABLE table_name (<column_list>) [CHARSET charset_name]
[COLLATE collation_name]
```

칼럼의 문자셋과 콜레이션이 생략되면, 테이블의 문자셋과 콜레이션이 사용된다. 테이블의 문자셋과 콜레이션이 생략되면, 시스템의 문자셋과 콜레이션이 사용된다.

다음은 테이블에 콜레이션을 지정하는 예이다.

```
CREATE TABLE tbl (
  i1 INTEGER,
  s STRING
) CHARSET utf8 COLLATE utf8_en_cs;
```

칼럼의 문자셋이 지정되어 있고 테이블의 콜레이션이 지정되는 경우, 해당 칼럼의 콜레이션은 칼럼의 문자셋에 대한 기본 콜레이션(<collation_name>_bin)이 지정된다.

```
CREATE TABLE tbl (col STRING CHARSET utf8) COLLATE utf8_en_ci;
```

위의 질의문에서 칼럼 col의 콜레이션은 칼럼에 대한 문자셋의 기본 콜레이션인 utf8_bin이 된다.

```
mysql> ;sc tbl

<Class Name>

tbl                                COLLATE utf8_en_ci

<Attributes>

col                                CHARACTER VARYING(1073741823) COLLATE utf8_bin
```

문자열 리터럴의 문자셋과 콜레이션

문자열 리터럴(string literal)의 문자셋과 콜레이션은 다음과 같은 우선 순위에 따라 정해진다.

1. 문자셋 소개자 또는 문자열의 COLLATE 수정자
2. SET NAMES 문으로 명시한 문자셋과 콜레이션
3. 시스템 문자셋과 콜레이션(DB 생성 시 지정한 로캘에 의한 기본 콜레이션)

SET NAMES 문

SET NAMES 문은 기본 클라이언트 문자셋과 콜레이션 값을 변경하여, 이를 실행한 클라이언트에서 이후에 실행하는 모든 문장은 지정한 문자셋과 콜레이션을 가지게 된다. 구문은 다음과 같다.

```
SET NAMES [charset_name] [COLLATE collation_name]
```

- *charset_name*: 유효한 문자셋 이름은 iso88591, utf8, euckr 그리고 binary이다.
- *collation_name*: 콜레이션 지정은 생략할 수 있으며, 모든 가능한 콜레이션이 설정될 수 있다. 콜레이션과 문자셋은 호환되어야 하며, 그렇지 않으면 오류가 발생한다. 사용 가능한 콜레이션 이름은 **db_collation** 카탈로그 뷰를 검색하여 확인할 수 있다. ([칼럼의 문자셋과 콜레이션](#) 참고)

SET NAMES 문으로 콜레이션을 지정하는 것은 시스템 파라미터 `intl_collation`을 지정하는 것과 같다. 따라서, 다음 두 문장은 같은 동작을 수행한다.

```
SET NAMES utf8;
SET SYSTEM PARAMETERS 'intl_collation=utf8_bin';
```

다음은 기본 문자셋과 콜레이션을 가진 문자열 상수를 생성한다.

```
SELECT 'a';
```

다음은 utf8 문자셋과 utf8_bin 콜레이션을 가진 문자열 상수를 생성한다. 참고로 기본 콜레이션은 해당 문자셋의 바이너리 콜레이션이다.

```
SET NAMES utf8;
SELECT 'a';
```

문자셋 소개자

상수 문자열 앞에는 문자셋 소개자(introducer)가 올 수 있고, 뒤에는 **COLLATE** 수정자(modifier)가 올 수 있는데, 문자셋 소개자는 언더바(_)로 시작하는 문자셋 이름으로 상수 문자열 앞에 올 수 있다. 문자열에 대해 문자셋 소개자와 **COLLATE** 수정자를 지정하는 구문은 다음과 같다.

```
[charset_introducer]'constant-string' [COLLATE collation_name]
```

- *charset_introducer*: 언더바(_)를 앞에 붙인 문자셋 이름으로 생략할 수 있다. `_utf8`, `_iso88591`, `_euckr`, `_binary` 중 하나를 입력할 수 있다.
- *constant-string*: 상수 문자열 값이다.

- *collation_name*: 시스템에서 사용 가능한 콜레이션 이름으로 생략할 수 있다.

상수 문자열의 기본 문자셋과 콜레이션은 현재의 데이터베이스 연결을 기준으로 정해지며, 가장 마지막에 수행한 **SET NAMES** 문 또는 기본값으로 설정된다. 문자셋 소개자 또는 **COLLATE** 수정자를 생략했을 때는 다음과 같이 동작한다.

- 문자셋 소개자를 지정하고 **COLLATE** 수정자를 생략할 때,
 - 문자셋 소개자가 클라이언트 문자셋(SET NAMES로 지정)과 동일하면 클라이언트 콜레이션이 적용된다.
 - 문자셋 소개자가 클라이언트 문자셋과 동일하지 않으면 문자셋 소개자와 동일한 바이너리 콜레이션(euckr_bin, iso88591_bin, utf8_bin 중 하나)이 적용된다.
- 문자셋 소개자를 생략하고 **COLLATE** 수정자를 지정하면, 문자셋은 콜레이션에 따라 적용된다.

다음은 문자셋 소개자와 **COLLATE** 수정자를 지정하는 예제이다.

```
SELECT 'cubrid';
SELECT _utf8'cubrid';
SELECT _utf8'cubrid' COLLATE utf8_en_cs;
```

다음 예에서는 utf8 문자셋과 utf8_en_cs 콜레이션을 가지는 문자열 상수를 생성한다. **SELECT** 문의 **COLLATE** 수정자가 **SET NAMES** 문에서 지정한 콜레이션을 오버라이드한다.

```
SET NAMES utf8 COLLATE utf8_en_ci;
SELECT 'a' COLLATE utf8_en_cs;
```

표현식의 문자셋과 콜레이션

표현식 결과의 문자셋과 콜레이션은 표현식의 인자들로부터 추론된다. 콜레이션 추론 과정은 콜레이션 변환도(coercibility)에 기반하며 이에 관한 자세한 내용은 **콜레이션이 서로 다를 때 결정 방식**을 참고한다.

모든 문자열 매칭 함수(LIKE, REPLACE, INSTR, POSITION, LOCATE, SUBSTRING_INDEX, FIND_IN_SET 등)와 비교 연산자들(<, >, = 등)에서 콜레이션이 고려된다.

시스템 데이터의 문자셋과 콜레이션

시스템 문자셋은 DB 생성 시 지정한 로캘에 의해 정해진다. 시스템의 콜레이션은 항상 시스템 문자셋의 바이너리 콜레이션(< charset>_bin)이다. CUBRID는 iso88591, euckr, utf8 이렇게 3개의 문자셋을 지원하며, 이에 따른 iso88591_bin, euckr_bin, utf8_bin 이렇게 3개의 시스템 콜레이션을 지원한다.

DB 생성 시 지정한 로캘의 영향

DB 생성 시 지정한 로캘은 다음에 영향을 끼친다.

- 식별자(identifier)와 대소문자 규칙에서 지원되는 문자(이를 “알파벳”이라고 지칭함)
- 날짜-문자열 변환 함수들에 대한 기본 로캘
- 숫자-문자열 변환 함수들에 대한 기본 로캘
- CSQL에서 콘솔 변환

식별자의 대소문자 구분

CUBRID에서 식별자는 대소문자 구분을 하지 않는다. 테이블 칼럼, 사용자 정의 변수, 트리거, 저장 프로시저들의 이름은 소문자로, 사용자 이름 및 그룹 이름은 대문자로 저장된다.

ISO-8859-1은 255 개의 문자만을 포함하므로 전체 문자들이 대소문자 변환의 대상이 되며, EUC-KR 문자셋에서는 ASCII 호환 문자들만이 대소문자 변환의 대상이 된다. ISO-8859-1과 EUC-KR 문자셋은 내장된(built-in) 데이터이다.

UTF-8 문자셋에 해당하는 로캘은 내장된 로캘(en_US.utf8, tr_TR.utf8, ko_KR.utf8)과 LDML 로캘의 두 종류가 있다.

내장된 로캘은 특정 문자(en_US.utf8과 ko_KR.utf8에서는 ASCII 문자, tr_TR.utf8에서는 ASCII 문자와 터키어의 글리프(glyphs) [1])만 구현한 것이다. 즉, 최대 4 바이트까지 인코딩된 모든 UTF-8 문자는 식별자로 받아들여지기는 하지만, 이 중 대부분의 문자들은 일반적인 유니코드 문자로 처리되어 대소문자 변환이 이뤄지지 않는다는 것을 뜻한다. 예를 들어, 테이블 이름에서 È(유니코드 코드포인트 00C8)는 허용되지만, 그것을 포함하는 식별자는 소문자 “è”로 변환되어 표현되지 않는다.

```
CREATE TABLE ÈABC;
```

따라서, 위의 질의 수행 시 테이블 이름은 **_db_class** 시스템 테이블에 “Èabc”라는 이름으로 저장된다.

LDML 로캘(내장된 로캘은 LDML 로캘에 의해 오버라이드됨)은 지원하는 유니코드 문자셋을 코드포인트 FFFF까지 확장하므로 식별자에 대해 확장된 알파벳의 대소문자 변환이 가능하다. 예를 들어, DB 생성 시 지정한 로캘이 es_ES.utf8이고 해당 로캘 라이브러리가 로딩되면, 위의 CREATE TABLE 문은 “èabc”라는 이름을 가진 테이블을 생성한다.

앞서 설명한 바와 같이 대소문자 규칙과 지원되는 문자들로 “알파벳”(LDML 파일에 “alphabet” 태그로 정의됨)이 구성되며, tr_TR과 de_DE와 같은 일부 로캘들은 특별한 대소문자 규칙을 가진다. 터키어에서 lower (‘İ’) = ‘i’(점없는 소문자 i)이고, upper (‘İ’) = ‘I’(점있는 대문자 I)이다. 독일어에서 upper (‘ß’) = ‘SS’(두 개의 대문자 S)이다.

이런 특수한 규칙을 갖는 로캘들은 식별자가 대문자 또는 소문자로 변환 저장되면서 발생할 수 있는 문제를 피하기 위해 시스템 데이터(식별자)를 위한 “알파벳”과 사용자 데이터를 위한 “알파벳” 두 개를 가진다. 즉, 사용자 데이터에 적용되는 “알파벳”은 위에서 설명한 특별한 규칙을 포함하는 반면, 시스템 데이터(식별자)를 위한 “알파벳”은 특별한 규칙을 포함하지 않는다. 예를 들어 터키어에서 그룹 이름 “public”에 대해 사용자 데이터에 적용되는 “알파벳”을 적용하면 “PUBLiC”이 되어 결국 의도했던 “PUBLIC”과는 다른 값이 되는 문제가 발생한다.

또한, 시스템 데이터를 위한 “알파벳”은 서로 다른 로캘을 가진 데이터베이스 간에 호환성 제공을 위해서도 필요하다. 이로 인해 서로 다른 로캘 사이에 데이터베이스의 스키마와 데이터를 언로드-로드 하는 것이 가능해진다.

문자열 리터럴의 입출력

문자열 리터럴 데이터는 CUBRID에 다양한 방법으로 입력된다.

- C API interface (CCI)
- 언어 의존적인 인터페이스. JDBC, Perl 드라이버 등
- CSQL - 콘솔 또는 파일로부터 입력

드라이버를 통해 문자 데이터를 받을 때, CUBRID는 이 문자들의 문자셋을 인지할 수 없다. 문자열 리터럴(따옴표로 감싼 문자)에 해당하는 모든 텍스트 데이터는 있는 그대로(raw data) 다루어진다. 문자셋 메타 정보는 응용 클라이언트에 의해 제공되어야 한다. 클라이언트는 **SET NAMES** 문이나 **문자셋 소개자**를 통해 문자셋 정보를 제공할 수 있다.

CSQL을 위한 텍스트 변환

CSQL 콘솔 인터페이스에서는 텍스트 변환 동작이 일어날 수 있다. 대부분의 로캘들에는 콘솔에서 ASCII 문자가 아닌 문자를 쉽게 쓸 수 있도록 해주는 문자셋이 별도로 존재한다. 예를 들어 로캘 tr_TR.utf8에 대한 LDML 파일에는 다음 라인이 포함되어 있다.

```
<consoleconversion type="ISO88599" windows_codepage="28599"
linux_charset="iso88599,ISO_8859-9,ISO8859-9,ISO-8859-9" />
```

사용자가 이와 같이 콘솔의 문자셋을 설정하면(예: Windows에서 chcp 28599, Linux에서 export LANG=tr_TR.iso88599), 모든 입력이 ISO-8859-9 문자셋으로 인코딩된다고 가정하고 모든 데이터를 UTF-8로 변환한다. 또한 결과를 출력할 때는 반대로 UTF-8을 ISO-8859-9로 변환한다. Linux에서는 이러한 변환을 피하기 위해 UTF-8 콘솔(예: export LANG=tr_TR.utf8)을 직접 사용할 것을 권장한다.

LDML 로캘 파일에서 이 XML 태그 설정은 반드시 요구되지는 않으며 선택 사항이다. 예를 들어, 로캘 km_KH.utf8은 관련 코드 페이지가 없다.

프랑스어 설정 및 프랑스어 문자 입력을 위한 설정 예

프랑스어 설정을 하려면 먼저 cubrid_locales.txt 파일에 fr_FR을 설정하고, 로캘을 컴파일(로캘 설정 참고)한 후 DB 생성 시 로캘을 fr_FR.utf8을 설정해야 한다.

Linux에서는 다음과 같이 설정한다.

- 콘솔이 UTF-8을 입력받을 수 있도록 LANG=fr_FR.utf8 또는 en_US.utf8로 설정한다. 이 설정은 프랑스어 문자 뿐만 아니라 모든 UTF-8 문자를 입력받을 수 있게 한다.
- 또는 콘솔이 ISO-8859-15를 입력받을 수 있도록 LANG=fr_FR.iso885915로 설정하고, LDML 파일의 <consoleconversion> 태그에서 linux_charset="iso885915"로 설정한다. 이와 같이 설정하면 ISO-8859-15 문자만을 입력받아 CSQL에 의해 UTF-8로 변환될 것이다.

Windows에서는 다음과 같이 설정한다.

- Windows 코드페이지를 28605로 변환한다(명령 프롬프트에서 chcp 28605). LDML <consoleconversion> 태그에서 set windows_codepage="28605"로 설정한다. 코드페이지 28605는 ISO-8859-15 문자셋에 해당한다.

루마니아어 설정 및 루마니아어 문자 입력을 위한 설정 예

루마니아어 설정을 하려면 먼저 cubrid_locales.txt 파일에 ro_RO를 설정하고, 로캘을 컴파일(로캘 설정 참고)한 후 DB 생성 시 로캘을 ro_RO.utf8을 설정해야 한다.

Linux에서는 다음과 같이 설정한다.

- 콘솔이 UTF-8을 입력받을 수 있도록 LANG=ro_RO.utf8 또는 en_US.utf8로 설정한다. 이 설정은 루마니아어 문자 뿐만 아니라 모든 UTF-8 문자를 입력받을 수 있게 한다.
- 또는 콘솔이 ISO-8859-2를 입력받을 수 있도록 LANG=ro_RO.iso88592로 설정한다. LDML 파일의 <consoleconversion> 태그에서 linux_charset="iso88592"로 설정한다. 이와 같이 설정하면 ISO-8859-2 문자만을 입력받아 CSQL에 의해 UTF-8로 변환될 것이다.

Windows에서는 다음과 같이 설정한다.

- Windows 코드페이지를 1250으로 변환한다(명령 프롬프트에서 chcp 1250). LDML <consoleconversion> 태그에서 set windows_codepage="1250"로 설정한다. 코드페이지 1250은 ISO-8859-2 문자셋에 해당하며, 루마니아어를 포함하는, 중유럽과 동유럽 언어들 일부에 특화된 문자셋이다. 코드페이지 1250 외의 문자는 제대로 출력되지 않음에 유의한다.

루마니아 알파벳에 존재하는 특수 문자(예: 문자 아래에 세딜라(cedilla)가 있는 “S”와 “T”)를 사용하기 위해서는 Windows의 “제어판”을 통해 루마니아어(레거시)로 키보드 설정을 변경해야 한다.

- ISO8859-2는 코드페이지 1250에 없는 문자를 일부 포함하고 있으므로, ISO8859-2의 모든 문자를 CSQL로 입력하거나 출력할 수는 없다.

입력 시 콘솔 변환 프로세스는 SQL 문장을 포함한 모든 입력에 대해 변환이 필요한 문자 데이터들을 변환한다. 결과 출력과 에러 메시지 출력 시에 CSQL은 모든 텍스트를 변환하지 않고 변환이 필요한

문자들만 변환한다. 예를 들어, 숫자 텍스트들은 모두 ASCII 문자이므로 이를 출력할 때는 콘솔 변환 작업이 발생하지 않는다.

유니코드 정규화

글리프(glyph [1], 문자 기호)는 유니코드 문자를 사용해서 다양한 방법으로 사용된다. 대부분 분해(decomposed)된 형태와 결합(composed form)된 형태로 사용된다. 예를 들어, 글리프 ‘Ä’는 단일 코드포인트 00C4로 결합된 형태로 쓰이는데, UTF-8에서 “C3 84”와 같이 2 바이트로 표현된다. 완전히 분해된 형태에서 2 개의 코드 포인트 0041(‘A’)과 0308(COMBINING DIAERESIS)로 쓰여지며, UTF-8에서 “41 CC 88”과 같이 3 바이트로 표현된다. 대부분의 텍스트 편집기는 두 가지 형식을 모두 다룰 수 있어서, 두 가지 형식의 인코딩은 같은 글리프 ‘Ä’로 나타날 것이다. 내부적으로 CUBRID는 “완전히 결합된” 텍스트만을 처리할 수 있다.

“완전히 분해된” 텍스트로 작업하는 클라이언트에 대해 CUBRID는 “완전히 결합된” 텍스트로 변환하고 돌려줄 때는 “완전히 분해된” 텍스트로 변환하도록 설정될 수 있다. 정규화는 로캘에 대한 기능이 아니며, 따라서 로캘에 의존하지 않는다.

시스템 파라미터 **unicode_input_normalization** 는 결합 여부를 설정하며, 시스템 파라미터 **unicode_output_normalization**는 분해 여부를 설정한다. 보다 자세한 내용은 **unicode_input_normalization**을 참고한다.

응용 클라이언트가 분해된 유니코드만을 다룰 수 있는 경우에는 **unicode_input_normalization**과 **unicode_output_normalization**을 모두 yes로 설정하는 경우를 사용해야 하고, 분해된 형태와 결합된 형태를 모두 다룰 수 있는 경우에는 **unicode_input_normalization** = yes와 **unicode_output_normalization** = no 로 설정하여 사용할 수 있다.

콜레이션의 축약과 확장

콜레이션의 구축을 위해 축약(contraction)과 확장(expansion)을 지원하며, 축약과 확장은 UTF-8 문자셋 콜레이션에서만 가능하다. 이러한 축약과 확장은 LDML 파일의 콜레이션 설정에서 정의할 수 있는데, 이들의 사용은 로캘 데이터(공유 라이브러리)의 크기와 서버의 성능 모두에 영향을 준다.

축약

축약은 둘 또는 그 이상의 코드포인트로 이루어진 일련의 문자들을 하나의 문자로 간주하여 정렬할 수 있도록 해주는 일련의 시퀀스들로 구성된다. 예를 들어, 전통적인 스페인어 정렬 순서에서 “ch”는 하나의 문자로 간주된다. “ch”로 시작하는 모든 단어들은 “c”로 시작하는 모든 단어들 뒤에 정렬되지만, “d”로 시작하는 단어보다 앞에 위치한다. 축약의 다른 예는 체코어의 “ch”인데 “h” 뒤에 정렬되며, 크로아티아어와 세르비아어의 라틴 문자에서 “lj”와 “nj”는 각각 “l”과 “n” 뒤에 정렬된다. 축약에 대한 추가 정보는 <https://unicode-org.github.io/icu/userguide/collation/concepts>를 참고한다. <http://www.unicode.org/Public/UCA/latest/allkeys.txt>의 DUCET에도 축약에 대해 일부가 정의되어 있다.

확장이 있는 콜레이션과 확장이 없는 콜레이션 모두에 대해 축약을 지원한다. 콜레이션을 정의하는 <setting> 태그에서 **DUCETContractions="ignore/use"** 와 **TailoringContractions="ignore/use"** 두 개의 LDML 파라미터를 통해서 설정할 수 있다. DUCETContractions 파라미터는 DUCET 파일에 있는 축약을 콜레이션에 로딩할 것인지를 결정하며, TailoringContractions 파라미터는 LDML 파일의 규칙에 의해 정의된 축약을 사용할 것인지를 결정한다.

확장

확장은 하나의 콜레이션 원소보다 많은 원소들을 가진 코드포인트(문자)들을 참조한다. 확장을 사용하면 아래에 서술된 바와 같이 콜레이션의 동작이 근본적으로 변경된다. LDML 파일의 CUBRIDExpansions="use" 파라미터 설정을 통해 확장을 사용할 수 있다.

확장이 없는 콜레이션

확장이 없는 콜레이션에서 각 코드포인트는 개별적으로 처리된다. 콜레이션의 세기에 기반하여 문자들이 완전히 정렬될 수도 있고 그렇지 않을 수도 있다. 콜레이션 알고리즘은 각 수준들의 집합의 가중치를 비교하여 해당 코드포인트의 가중치를 나타내는 하나의 값을 생성하며, 이를 기반으로 코드포인트들을 정렬한다. 확장이 없는 콜레이션에서 두 문자열 비교는 이와 같이 계산된 가중치로 각각의 코드포인트를 차례로 비교하게 된다.

확장이 있는 콜레이션

확장이 있는 콜레이션에서 일부 결합 문자(코드포인트)들은 다른 문자들로 구성된 순서 있는 리스트(ordered list)로 해석된다. 예를 들어, 'æ'는 'ae', 'ä'는 'ae' 또는 'aa'와 같이 해석된다. DUCET에서 'æ'의 콜레이션 원소 리스트는 'a'와 'e'의 순서로 두 콜레이션 원소 리스트들을 연결(concatenation)한 것이 된다. 코드포인트에 대해 특정한 순서를 부여하는 것은 불가능하며, 각 문자(코드포인트)들의 새로운 가중치를 계산하는 것도 불가능하다.

확장이 있는 콜레이션에서 문자열 비교는 두 개의 코드포인트/축약에 대해 콜레이션 원소들을 연결(concatenation)한 후에, 각 단계별로 두 리스트의 가중치를 비교하는 것이다.

예제 1

다음의 예제는 콜레이션 설정에 따라 문자열 비교가 다른 결과를 가져올 수 있다는 것을 보여준다.

다음의 DUCET 파일 일부는 아래 예에서 사용할 코드포인트들이다.

```
0041 ; [.15A3.0020.0008.0041] # LATIN CAPITAL LETTER A
0052 ; [.1770.0020.0008.0052] # LATIN CAPITAL LETTER R
0061 ; [.15A3.0020.0002.0061] # LATIN SMALL LETTER A
0072 ; [.1770.0020.0002.0072] # LATIN SMALL LETTER R
00C4 ; [.15A3.0020.0008.0041][.0000.0047.0002.0308] #
```

```
LATIN CAPITAL LETTER A WITH DIAERESIS;
00E4 ; [.15A3.0020.0002.0061][.0000.0047.0002.0308] #
LATIN SMALL LETTER A WITH DIAERESIS;
```

콜레이션을 위해 세 가지 설정 타입으로 표현된다.

- 첫 번째 세기(primary strength), 대소문자 구분 없음 (단계 1)
- 두 번째 세기(secondary strength), 대소문자 구분 없음 (단계 1, 2)
- 세 번째 세기(tertiary strength), 대문자 우선 (단계 1, 2, 3)

지금부터 확장이 있는 콜레이션과 확장이 없는 콜레이션을 가지고 문자열 “Ar”과 “Är”의 정렬을 살펴볼 것이다.

확장이 없는 콜레이션

확장이 없을 때 각 코드포인트는 하나의 가중치 값을 부여받는다. A, Ä, R과 해당 소문자 코드포인트에 대한 가중치를 위에서 언급된 알고리즘을 기반으로 한 이 문자들에 대한 코드포인트의 순서는 다음과 같다.

- 첫 번째 세기: $A = \ddot{A} < R = r$
- 두 번째 세기: $A < \ddot{A} < R = r$
- 세 번째 세기: $A < \ddot{A} < R < r$

각 코드포인트에 대해 계산된 가중치가 있기 때문에 문자열들의 정렬 순서는 쉽게 결정된다.

- 첫 번째 세기: “Ar” = “Är”
- 두 번째 세기: “Ar” < “Är”
- 세 번째 세기: “Ar” < “Är”

확장이 있는 콜레이션

확장이 있는 콜레이션이면 정렬 순서가 바뀐다. DUCET에 기반한 콜레이션 원소들의 연결된 리스트들은 다음과 같다.

```
Ar      [.15A3.0020.0008.0041][.
1770.0020.0002.0072]
Är      [.15A3.0020.0008.0041][.
0000.0047.0002.0308][.1770.0020.0002.0072]
```

수준 1의 첫 번째 과정에서 가중치 0x15A3과 0x15A3이 비교된다. 두 번째 과정에서 가중치 0x0000은 비교가 생략되고, 0x1770과 0x1770이 비교된다. 여

기까지 문자열이 동일하므로 수준 2 가중치로 계속 비교하게 되는데, 첫 번째 과정에서 0x0020을 0x0020과 비교하고 두 번째 과정에서 0x0020을 0x0047과 비교하여 “Ar” < “Är”이라는 결과를 생성한다. 이 예제를 통해 확장이 있는 콜레이션을 사용할 때 어떻게 문자열 비교가 수행되는지 살펴보았다.

이제 콜레이션 설정을 독일어 콜레이션으로 변경하여 같은 문자열에 대해 다른 순서로 정렬되는 과정을 살펴보자. 독일어에서 “Ä”는 문자 그룹 “AE”로 해석된다. 이 예에 해당하는 코드포인트와 콜레이션 원소들은 다음과 같다.

```
0041 ; [.15A3.0020.0008.0041] # LATIN CAPITAL LETTER A
0045 ; [.15FF.0020.0008.0045] # LATIN CAPITAL LETTER E
0072 ; [.1770.0020.0002.0072] # LATIN SMALL LETTER R
00C4 ; [.15A3.0020.0008.0041][.15FF.0020.0008.0045] # LATIN CAPITAL LETTER A WITH DIAERESIS; EXPANSION
```

문자열 “Ar”과 “Är”을 비교할 때 확장이 있는 콜레이션을 사용하면, 두 문자열에 있는 문자들의 콜레이션 원소 리스트를 결합한 후 비교하는 과정이 포함된다.

```
Ar [.15A3.0020.0008.0041][.1770.0020.0002.0072]
Är [.15A3.0020.0008.0041][.15FF.0020.0008.0045][.1770.0020.0002.0072]
```

첫 번째 과정에서 수준 1의 가중치 0x15A3과 0x15A3를 비교한다. 그리고 나서 0x1770과 0x15FF를 비교하여 “Ar” > “Är”이라는 결과가 나오는데, 이는 앞의 예제와는 전혀 다른 결과이다.

예제 2

캐나다 프랑스어의 확장이 있는 콜레이션(utf8_fr_exp_ab)에서는 액센트 문자를 뒤쪽에서 앞쪽 방향으로 비교하여 정렬한다.

- 일반적인 액센트 순서: cote < coté < côte < côté
- 역순 액센트 순서: cote < côte < coté < côté

문자셋과 콜레이션을 필요로 하는 연산

문자셋

문자셋 정보는 문자열 데이터를 사용하는 연산자와 함수들에게 필요한 정보이다. 하지만 예외적으로, `OCTET_LENGTH()` 함수와 `BIT_LENGTH()` 함수는 바이트와 비트 값을 반환하기 위해 문자셋 정보가 불필요하다. 그러나, 다른 문자셋으로 저장된 같은 글리프(glyph, 문자 기호)에 대해 이 함수들을 수행하면 다른 값을 반환한다.

```
CREATE TABLE t (s_iso STRING CHARSET iso88591, s_utf8 STRING CHARSET utf8);
SET NAMES iso88591;
INSERT INTO t VALUES ('È', 'È');

-- the first returns 1, while the second does 2
SELECT OCTET_LENGTH(s_iso), OCTET_LENGTH(s_utf8) FROM t;
```

위의 예는 콘솔(또는 응용 클라이언트)에서 iso88591 문자셋으로 질의를 입력해야 한다.

콜레이션

콜레이션은 `STRCMP()`, `POSITION()`, LIKE 조건, `<`, `=`, `>` 등 두 문자열 간의 비교 또는 매칭을 수반하는 연산자와 함수에서 필요하다. 또한 ORDER BY, GROUP BY 및 집계 함수(`MIN()`, `MAX()`, `GROUP_CONCAT()`) 등에서도 콜레이션이 필요하다.

콜레이션은 또한 `UPPER()`, `LOWER()` 함수에서도 고려되는데, 다음과 같이 동작한다.

- 각 콜레이션은 부모에 해당하는 기본 로캘이 있다.
- UPPER와 LOWER 함수는 콜레이션의 기본 로캘을 사용하여 수행된다.

대부분의 콜레이션에서 기본 로캘은 명확하다(콜레이션 이름에 포함됨).

- utf8_tr_cs → tr_TR.utf8
- iso88591_en_ci → en_US (ISO-8859-1 문자셋)

바이너리 콜레이션은 다음의 기본 로캘을 가지고 있다.

- iso88591_bin → en_US (ISO-8859-1 문자셋)
- utf8_bin (en_US.utf8 - 내장된 로캘 - ASCII 문자만 다룬다)
- euckr_bin (ko_KR.euckr - 내장된 로캘 - ASCII 문자만 다룬다)

LDML에서 정의한 일반적인(generic) 콜레이션들이 있는데, 이 콜레이션들은 **\$CUBRID/conf/cubrid_locales.txt** 파일에서 먼저 나타나는 로캘을 기본 로캘로 가진다. 최초 설치 시 가장 처음에 나타나는 로캘은 de_DE(독일어)이므로, de_DE가 일반적인 콜레이션의 기본 로캘이 된다.

문자셋 변환

CUBRID가 지원하는 3 가지 문자셋에 대한 변환 규칙은 다음과 같다.

- 일반적인 규칙은 문자 트랜스코딩((바이트의 표현이 대상 문자셋으로 표현됨)이 발생하는 것이다
 - VARCHAR 문자 데이터에서 문자의 길이(precision)는 유지되는 반면, 바이트 크기는 바뀔 수 있다. 고정 길이 칼럼의 문자셋을 바꿀 때(ALTER..CHANGE), 바이트 크기는 항상 바뀐다. (크기 = 길이 x 문자셋에서 문자 하나가 차지하는 바이트 크기)
- 예외 규칙은 utf8과 euckr에서 iso88591로 변환하는 것인데, 길이는 유지되지만 데이터는 잘릴 수 있다.

다음은 DB 생성 시 지정한 로캘이 en_US(.iso88591)인 데이터베이스에서 utf8로 문자셋을 변환할 때 데이터가 잘리는 예이다.

```
SET NAMES utf8;
CREATE TABLE t1(col1 CHAR(1));
INSERT INTO t1 VALUES ('Ç');
```

위의 질의를 수행하면 'Ç'는 2바이트 문자인데 col1 칼럼의 크기는 1바이트이므로 col1의 데이터가 잘리게 된다. 데이터베이스의 문자셋은 iso88591인데 입력 데이터의 문자셋은 utf8이며, 이는 utf8을 iso88591로 변환한다.

콜레이션 설정으로 인한 영향

LIKE 조건 최적화

LIKE 조건은 문자열 데이터 간의 패턴을 비교하여 문자열이 패턴과 매칭되면 **TRUE** 를 리턴한다. 확장이 없는 콜레이션을 사용할 때에는 각 코드포인트는 가중치를 나타내는 하나의 정수 값을 갖는데, 이 가중치 값은 콜레이션 설정(세기, 대소문자 구분 등)에 기반하여 계산된다.

문자들은 항상 하나의 개체(entity)로 간주될 수 있기 때문에, **LIKE** 조건을 사용한 패턴으로 문자열을 매칭하려는 시도는 문자열이 어떤 범위 내에서 발견될 수 있는지 확인하는 것과 같다고 볼 수 있다. 예를 들어, "s LIKE 'abc%'"는 칼럼 s의 값은 문자열 "abc"로 시작해야 한다는 것을 의미하므로, "s LIKE 'abc%'"와 같은 절을 수행하기 위해 먼저 칼럼 s의 문자열에 대한 제한 범위로 구문을 재작성한다. 확장이 없는 콜레이션에서 이는 s가 "abc"보다 크거나 같지만 후속 문자열보다 작다는 의미이다. 예를 들어, 영어의 알파벳 기준으로 'abc'의 후속 문자열은 'abd'이므로 아래와 같이 변환할 수 있다.

```
s LIKE 'abc%' → s ≥ 'abc' AND s < 'abd' (if using strictly the English alphabet)
```

이와 같이 **LIKE** 조건의 패턴은 간단한 비교 조건으로 대체될 수 있는데, 확장이 있는 콜레이션의 경우는 다르게 동작한다.

확장이 있는 콜레이션을 사용할 때 문자열을 비교하는 것은 각 코드포인트나 확장에 대해 각 수준(level)별로 콜레이션 원소의 결합된 리스트들을 비교하는 것을 뜻한다. 확장이 있는 콜레이션에서 문자열 비교에 대한 자세한 내용은 **확장** 을 참고하면 된다.

확장이 있는 콜레이션에 대해서 위의 예와 같이 **LIKE** 조건을 일반 비교 조건으로 변환하면 비교가 잘못될 수 있다. 정확한 질의 결과를 보장하기 위하여 확장이 있는 콜레이션의 경우에는 **LIKE** 조건 재작성 방법이 아래의 예와 같이 다르게 동작한다. 즉, 범위 조건으로 변경하고 확장 있는 콜레이션에서 잘못 추가될 수 있는 데이터들을 배제시키기 위해 주어진 **LIKE** 조건이 필터로 추가된다.

```
s LIKE 'abc%' → s ≥ 'abc' AND s < 'abd' and s LIKE 'abc%' (if using strictly the English alphabet)
```

커버링 인덱스

커버링 인덱스 스캔은 실제 레코드를 액세스하지 않고 인덱스에 있는 값만을 사용해서 질의를 처리하는 질의 최적화 기법으로, 이에 대한 자세한 내용은 **커버링 인덱스** 를 참고한다.

대소문자 구분이 없는 콜레이션에서 'abc'와 'ABC' 두 개의 문자열을 인덱스에 저장한다고 가정할 때 인덱스의 키 값으로 이 중 하나만이 저장된다. 이와 같이 하나의 콜레이션에서 서로 다른 두 개 이상의 문자열이 하나의 키 값을 가지게 되면, 정확한 질의 결과를 만들어낼 수 없게 된다. 결국 수준 4(quaternary) 보다 작은 세기(strength)의 모든 UTF-8 콜레이션에서 커버링 인덱스에 의한 질의 최적화는 적용되지 않는다.

이러한 세기 수준은 LDML에서 콜레이션 정의를 위한 <strength> 태그의 strength="tertiary/quaternary"에 의해 제어될 수 있다. 확장이 있는 콜레이션의 세기 값을 최대 값으로 설정하는 것은 주의 깊게 고려되어야 하는데, 네 번째 세기 수준은 공유 라이브러리 크기와 메모리 요구량이 커질 뿐만 아니라 문자열 비교 시간을 증가시키기 때문이다.

콜레이션과 관련된 자세한 내용은 **콜레이션 설정** 을 참고한다.

각 콜레이션에 대한 기능 요약

콜레이션	LIKE 문을 범위로 재작성 이 후 조건 유지	커버링 인덱스 허용
iso88591_bin	No	Yes
iso88591_en_cs	No	Yes
iso88591_en_ci	Yes	No
utf8_bin	No	Yes
euckr_bin	No	Yes
utf8_en_cs	No	Yes
utf8_en_ci	Yes	No
utf8_tr_cs	No	Yes
utf8_ko_cs	No	Yes
utf8_gen	No	Yes
utf8_gen_ai_ci	Yes	No
utf8_gen_ci	Yes	No
utf8_de_exp_ai_ci	Yes	No
utf8_de_exp	Yes	No

콜레이션	LIKE 문을 범위로 재작성 이 후 조건 유지	커버링 인덱스 허용
utf8_ro_cs	No	Yes
utf8_es_cs	No	Yes
utf8_fr_exp_ab	Yes	No
utf8_ja_exp	Yes	No
utf8_ja_exp_cbm	Yes	No
utf8_km_exp	Yes	No
utf8_ko_cs_uca	No	Yes
utf8_tr_cs_uca	No	Yes
utf8_vi_cs	No	Yes
binary	No	Yes

콜레이션 정보 보기

콜레이션 정보를 보려면 `CHARSET()`, `COLLATION()` 및 `COERCIBILITY()` 정보 함수를 사용한다.

데이터베이스 콜레이션 정보는 **db_collation** 시스템 뷰 또는 **SHOW COLLATION**을 통해 확인할 수 있다.

JDBC에서 i18n 문자 사용

CUBRID JDBC는 문자열 및 CUBRIDBinaryString 객체를 사용하여 서버에서 수신된 문자열 타입 값을 저장한다. 문자열 객체는 각 문자를 저장하기 위해 내부적으로 UTF-16을 사용하며, 바이너리 문자셋을 제외한 모든 문자열을 저장하는 데 사용된다. CUBRIDBinaryString은 바이트 배열을 사용하며 바

이너리 문자셋의 문자열 값을 저장하는 데 사용된다. 서버에서 수신된 각 문자열 값의 데이터 버퍼에는 서버에서 수신된 값의 문자셋이 포함된다. 이 문자셋은 컬럼의 문자셋, 표현식 결과의 문자셋, 시스템 문자셋(문자셋이 표현되지 않은 경우) 중 하나이다

참고

이전 버전에서는 JDBC 문자열 객체를 인스턴스화하는 데 연결URL의 문자셋이 사용되었다.

UTF-8 문자셋을 갖는 컬럼 한 개와 바이너리 문자셋을 갖는 다른 컬럼이 포함된 테이블 생성:

```
CREATE TABLE t1(col1 VARCHAR(10) CHARSET utf8, col2 VARCHAR(10)
CHARSET binary);
```

한 레코드 입력 (두번째 컬럼은 랜덤 값):

```
Connection conn = getConn(null);
PreparedStatement st = conn.prepareStatement("insert into t1
values( ?, ? )");
byte[] b = new byte[]{(byte)161, (byte)224};
CUBRIDBinaryString cbs = new CUBRIDBinaryString(b);
String utf8_str = new String("abc");
st.setObject(1, utf8_str);
st.setObject(2, cbs);
st.executeUpdate();
```

테이블 질의 및 내용 표시(바이너리 문자열의 경우 hex dump, 다른 문자셋의 경우 문자열 값 표시):

```
ResultSet rs = null;
Statement stmt = null;
rs = stmt.executeQuery("select col1, col2 from t1;");
ResultSetMetaData rsmd = null;
rsmd = rs.getMetaData();
int numberOfColumn = rsmd.getColumnCount();
while (rs.next()) {
    for (int j = 1; j <= numberOfColumn; j++) {
        String columnName = rsmd.getColumnName(j);
        int columnType = rsmd.getColumnType(j);
        if (((CUBRIDResultSetMetaData)
rsmd).getColumnCharset(j).equals("BINARY")) {
            // database string with binary charset
            Object res;
            byte[] byte_array = ((CUBRIDBinaryString)
res).getBytes();
            res = rs.getObject(j);
            System.out.println(res.toString());
        }
    }
}
```

```

        } else {
            // database string with any other charset
            String res;
            res = rs.getString(j);
            System.out.print(res);
        }
    }
}

```

타임존 설정

타임존은 시스템 파라미터를 통해 설정할 수 있는데, 세션에서 설정하는 **timezone** 파라미터와 데이터베이스 서버에서 설정하는 **server_timezone** 파라미터가 있다. 타임존 파라미터에 대한 설명은 **타임존 파라미터**를 참고한다.

timezone 파라미터는 세션에 대한 파라미터이다. 세션 단위로 별개의 설정을 유지할 수 있다.

```
SET SYSTEM PARAMETERS 'timezone=Asia/Seoul';
```

이 값을 설정하지 않은 경우 **server_timezone** 파라미터의 설정을 따른다.

server_timezone 파라미터는 데이터베이스 서버에 대한 파라미터이다.

```
SET SYSTEM PARAMETERS 'server_timezone=Asia/Seoul';
```

이 값을 설정하지 않은 경우 OS의 설정을 따른다.

타임존을 사용하려면 타임존 타입을 사용해야 하는데, 이와 관련하여 **타임존이 있는 날짜/시간 데이터 타입**을 참고한다.

타임존을 지역 이름으로 설정하는 경우 별도의 타임존 라이브러리를 필요로 하는데, 큐브리드 설치 시 제공하는 라이브러리가 아닌 업데이트된 타임존 정보를 가진 라이브러리를 사용하려면, 타임존 정보를 변경한 후 직접 라이브러리를 컴파일해야 한다.

다음은 최신의 타임존 정보를 IANA(<http://www.iana.org/time-zones>)로부터 받아와 타임존 라이브러리를 컴파일하는 예이다.

보다 자세한 설명은 다음 설명을 참고한다.

타임존 라이브러리 컴파일

지역 이름으로 타임존을 명시하여 사용하려면 타임존 라이브러리가 필요한데, 큐브리드 설치 시 기본으로 제공된다. 그런데, 타임존 지역 정보를 최신 정보로 업데이트하려면 IANA(<http://www.iana.org/time-zones>)에서 최신 코드를 받아와 타임존 라이브러리를 컴파일해야 한다.

다음은 타임존 라이브러리를 최신 정보로 업데이트하기 위한 절차이다.

먼저 구동 중인 큐브리드 서비스를 종료한 후, OS에 따라 다음 절차를 수행한다.

윈도우즈

1. <http://www.iana.org/time-zones> 에서 최신의 타임존 데이터를 내려받는다.

“Latest version”의 “Time Zone Data”에 링크된 파일을 받는다.

2. **%CUBRID%/timezones/tzdata** 디렉터리 이하에 압축 파일을 푼다.

3. **%CUBRID%/bin/make_tz.bat** 를 실행한다.

%CUBRID%/lib 이하에 **libcubrid_timezones.dll** 이 생성된다.

```
make_tz.bat
```

참고

make_locale 스크립트를 Windows에서 실행하려면, Visual C++ 2005, 2008 또는 2010 중 하나가 설치되어 있어야 한다.

리눅스

1. <http://www.iana.org/time-zones> 에서 최신의 타임존 데이터를 내려 받는다.

“Latest version”의 “Time Zone Data”에 링크된 파일을 다운로드 한다.

2. 압축된 파일을 **\$CUBRID/timezones/tzdata** directory에 푼다.

3. **make_tz.sh** 를 실행한다.

\$CUBRID/lib 디렉터리에 **libcubrid_timezones.so** 파일이 생성된다.

```
make_tz.sh
```

타임존 라이브러리 및 데이터베이스 호환성

CUBRID에서 빌드된 타임존 라이브러리에는 타임존 데이터의 MD5 체크섬이 포함된다. 이 해시는 해당 라이브러리에서 생성된 모든 데이터베이스의 **db_root** 시스템 테이블의 **timezone_checksum** 컬

럼에 저장된다. 타임존 라이브러리가 변경되고(새로운 타임존 데이터로 다시 컴파일됨) 체크섬이 변경되면 기존 데이터베이스로 CUBRID 서버와 그 밖의 모든 CUBRID 도구를 시작할 수 없다. 이러한 호환성 문제를 방지하는 방법 중 하나는 **make_tz** 도구의 **extend** 인자를 사용하는 것이다. **extend** 옵션을 사용하면 데이터베이스의 **timezone_checksum** 값도 새로운 타임존 라이브러리의 새로운 MD5 체크섬으로 변경된다. IANA 사이트에서 다른 타임존 라이브러리 버전을 사용하려는 경우 **extend** 기능을 사용해야 한다. 이 기능이 수행하는 두 가지 작업은 다음과 같다.

- 이전 타임존 데이터와 새 타임존 데이터를 병합하여 새 라이브러리를 생성한다. 데이터베이스 테이블에 있는 데이터와의 역호환성을 위해, 병합 후에도 새 타임존 데이터베이스에 존재하지 않는 모든 타임존 지역을 유지한다.
- 새 타임존 라이브러리가 생성되었을 때 역호환성을 유지할 수 없는 경우 테이블에서 타임존 데이터를 갱신한다. 오프셋 규칙 또는 서머타임 규칙이 변경될 때 이러한 상황이 발생할 수 있다.

extend 옵션과 함께 **make_tz**를 실행하면 데이터베이스 디렉터리 파일(**databases.txt**)에 있는 모든 데이터베이스가 MD5 체크섬과 함께 갱신된다. 다음과 같은 예외적인 상황이 있다.

- 여러 사용자가 동일한 **CUBRID** 설치를 공유하는데, 그 중 한 명이 **extend**를 수행하는 경우가 있다. **extend**를 수행한 사용자에게 다른 사용자들의 데이터베이스가 포함된 파일에 액세스할 수 있는 권한이 없는 경우 다른 사용자들의 데이터베이스는 갱신되지 않는다. 그 후, 데이터베이스가 갱신되지 않은 다른 사용자가 자신의 데이터베이스에서 **extend**를 수행하려고 시도하면 라이브러리의 체크섬이 자신의 데이터베이스에 있는 체크섬과 다르기 때문에 작업에 실패한다.
- **CUBRID_DATABASES** 환경 변수 값이 다른 사용자의 경우 **databases.txt** 파일이 다르다. 이 경우 현재로서는 한 번에 모든 데이터베이스를 갱신하는 것이 불가능하다.

해결책은 두 가지가 있다.

- 각 사용자에게 자신의 데이터베이스가 포함된 폴더에 액세스할 수 있는 권한을 부여하고, **CUBRID_DATABASES** 변수의 값을 동일하게 만든다.
- 이것이 불가능한 경우 각 사용자에게 대해 다음을 수행할 수 있다.
 - 현재 타임존 라이브러리 및 **databases.txt** 파일 백업
 - **databases.txt** 파일에서 현재 사용자의 데이터베이스를 제외한 모든 데이터베이스 삭제
 - **extend** 실행
 - **databases.txt** 파일 및 타임존 라이브러리 복구

단 마지막 사용자의 경우 최종 단계에서 **databases.txt** 파일만 복구하고 타임존 라이브러리 복구는 하지 않는다.

리눅스의 경우:

```
make_tz.sh -g extend
```

윈도우즈의 경우:

```
make_tz.bat /extend
```

JDBC에서 타임존 데이터 타입 사용

JDBC CUBRID 드라이버는 타임존 정보에 대해 완전하게 CUBRID 서버에 의존한다. 자바와 CUBRID가 타임존에 대해 동일한 정보의 기본 소스가 사용되더라도 지역명과 타임존 정보가 호환되지 않는다고 간주된다.

타임존이 있는 모든 CUBRID 데이터 타입은 **CUBRIDTimestamptz** 자바 객체에 매핑된다.

JDBC를 사용하여 타임존이 있는 값 삽입:

```
String datetime = "2000-01-01 01:02:03.123";
String timezone = "Europe/Kiev";
CUBRIDTimestamptz dt_tz =
CUBRIDTimestamptz.valueOf(datetime, false, timezone);
PreparedStatement pstmt = conn.prepareStatement("insert
into t values(?");
pstmt.setObject(1, ts);
pstmt.executeUpdate();
```

JDBC를 사용하여 타임존이 있는 값 검색:

```
String sql = "select * from tz";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
CUBRIDTimestamptz object1 = (CUBRIDTimestamptz)
rs.getObject(1);
System.out.println("object: " + object1.toString());
```

CUBRID JDBC는 내부적으로 **CUBRIDTimestamptz** 객체의 날짜/시간 부분을 1970년 1월 1일 이후 경과한 시간(밀리초)(Unix epoch)이 보관되는 'long' 값(날짜 객체를 통해 상속)에 저장한다.

CUBRIDTimestamptz 객체는 로컬 타임존을 사용하는 타임스탬프 객체와는 달리 UTC 시간 참조에서 내부 인코딩을 수행하기 때문에, 자바 로컬 타임존과 동일한 타임존으로 생성되어도 보관하는 내부 epoch 값은 다르다. Unix epoch를 제공하려면 **getUnixTime()** 메서드를 사용하면 된다.

```
String datetime = "2000-01-01 01:02:03.123";
String timezone = "Asia/Seoul";
CUBRIDTimestamptz dt_tz =
CUBRIDTimestamptz.valueOf(datetime, false, timezone);
System.out.println("dt_tz.getTime: " + dt_tz.getTime());
```

```
System.out.println("dt_tz.getUnixTime: " +
dt_tz.getUnixTime ());
```

다국어 설정을 위한 고려 사항

데이터베이스 설계자는 데이터베이스 구조를 설계할 때 문자 데이터의 속성을 고려해야 한다. 다음은 문자 데이터 설정 시에 알아야 할 사항들을 요약한 것이다.

로캘

- 기본적으로 DB 생성 시 로캘을 en_US를 사용하면 가장 좋은 성능을 낸다. 영어만 사용한다면 로캘을 en_US로 지정하는 것이 가장 좋다.
- UTF-8 로캘의 사용은 CHAR 타입에 대해 추가적인 저장 공간을 필요로 한다. 고정 길이 문자 타입 (CHAR)의 경우 4배까지 늘어나며, EUC-KR의 경우 3배까지 늘어난다.
- 사용자 문자열 리터럴이 시스템과 다른 문자셋과 콜레이션을 가진다면, 질의문은 문자셋과 콜레이션을 설정하기 위해 더 길어질 것이다.
- ASCII 문자가 아닌 문자를 사용자 식별자로 사용하려면 .utf8 로캘을 사용한다.
- UTF-8 문자셋이 설정되면, 시스템 로캘보다는 LDML 로캘을 사용하는 것이 좋다. LDML 로캘은 식별자에 포함된 유니코드 문자의 대소문자 변환을 보장한다.
- 로캘 설정은 문자열 변환 함수에도 영향을 준다. 가장 빈번하게 변환이 이루어지는 로캘을 선택하는 것이 좋다.
- 로캘 설정 시 문자열 상수, 사용자 테이블 칼럼의 문자셋과 콜레이션에 대해 고민하지 않아도 된다. 질의문에서 `CAST()` 연산자를 사용하여 런타임에 바꿀 수 있으며, "ALTER ... CHANGE" 문을 통해 영구히 바꿀 수 있다. 단, 콜레이션 간에 변환은 같은 문자셋끼리만 가능하다.

CHAR와 VARCHAR

- 일반적으로 데이터의 크기 변화가 심할 때는 VARCHAR를 사용한다.
- CHAR 타입은 고정 길이 타입이므로, 영문만 저장하는 경우에도 UTF-8은 4 bytes의 저장 공간을, EUC-KR은 3 bytes의 저장 공간을 필요로 한다.
- 칼럼의 자릿수(precision)는 문자(글리프, 문자 기호)의 개수이다.
- 자릿수를 정한 이후에, 대부분의 사용 시나리오에 따라 문자셋과 콜레이션이 설정되어야 한다.

문자셋 선택

- 텍스트가 ASCII가 아닌 문자를 포함하고 있더라도 응용 프로그램이 문자 개수 세기, 문자열 치환 등의 문자열 조작이 필요한 경우에 utf8이나 euckr 문자셋을 사용한다.
- CHAR 타입의 저장 공간을 고려해야 한다.
- CHAR와 VARCHAR 타입에서 INSERT/UPDATE 시 오버헤드가 있다. 각 행에서 문자열의 문자 개수 (precision)를 세는 것은 ISO가 아닌 문자셋에서 더 많은 수행 시간을 필요로 한다.
- 질의문 내의 표현식의 문자셋은 `CAST()` 연산자를 사용하여 변환할 수 있다.

콜레이션 선택

- 콜레이션에 의존하는 연산(문자열 검색, 정렬, 비교, 대소문자 구분)을 수행하지 않는 경우 해당 문자셋의 bin 콜레이션이나 바이너리 콜레이션을 선택한다.
- 표현식의 원래 문자셋과 새로운 콜레이션 사이에서 문자셋이 변경되지 않는다면 콜레이션은 `CAST()` 연산자, `COLLATE` 수정자를 사용하여 쉽게 오버라이드할 수 있다.
- 콜레이션은 문자열의 대소문자 구분 규칙을 제어한다.
- 확장이 있는 콜레이션은 더 느리지만 더 유연하며 전체 단어 정렬을 수행한다.

정규화

- 응용 클라이언트가 CUBRID에 분해된 형태의 텍스트 데이터를 보낸다면, **unicode_input_normalization** = yes로 설정하여 CUBRID가 재구성하여 결합된 형태로 다룰 수 있도록 한다.
- 응용 클라이언트가 분해된 형태만 다룰 수 있다면, **unicode_output_normalization** = yes로 설정하여 CUBRID가 항상 분해된 형식으로 텍스트 데이터를 보내도록 한다.
- 응용 클라이언트가 모든 형식을 알고 있다면, **unicode_output_normalization** = no인 상태로 둔다.

CAST vs COLLATE

- `CAST()` 연산자는 `COLLATE` 수정자보다 더 많은 실행 비용이 든다 (문자셋 변환이 발생하면 더 많은 비용이 든다).
- `COLLATE` 수정자는 질의 실행 과정에서 실행 연산이 추가되지 않기 때문에, `COLLATE` 수정자를 사용하는 것이 `CAST()` 연산자를 사용하는 것보다 실행 속도가 더 빠르다.

- **COLLATE 수정자**는 문자셋이 바뀌지 않을 때만 사용될 수 있다.

주의 사항

- 호스트 변수의 자연 바인딩이 있는 경우에 질의 실행 계획 출력 시에 콜레이션이 출력되지 않는다.
- 기본 다국어 영역인 0000~FFFF 범위의 유니코드 코드포인트만 정규화된다.
- 숫자 구분자로 특정 문자(예를 들어, 공백 문자)를 사용할 수 없다. 일부 로캘에서는 공백 문자를 숫자 구분자로 활용하기도 하는데, 이는 허용되지 않는다.

참고

- 9.2 이하 버전에서 사용자 정의 변수는 시스템 콜레이션과 다른 콜레이션으로 설정할 수 없다. 예를 들어, 시스템 콜레이션이 iso88591일 때 “SET @v1='a' COLLATE utf8_en_cs;”과 같은 구문을 사용할 수 없다.
- 9.3 이상 버전에서는 위의 제약이 더 이상 존재하지 않는다.

로캘과 콜레이션 추가 안내서

대부분의 새로운 로캘 또는 콜레이션은 사용자가 LDML 파일을 새로 추가 또는 변경해서 추가될 수 있다. CUBRID에서 사용되는 LDML 파일 포맷은 Unicode Locale Data Markup Language(<http://www.unicode.org/reports/tr35/>)에서 비롯되었다. CUBRID에만 명시되는 태그와 속성은 이름에 “cubrid”를 포함하고 있기 때문에 쉽게 구별할 수 있다.

새로운 로캘을 추가하는 가장 좋은 방법은 기존의 LDML 파일을 복사해서 원하는 결과가 나올 때까지 설정을 바꾸는 것이다. 파일 이름은 “cubrid_<language>.xml”과 같이 주어져야 하며 **\$CUBRID/locales/data/ldml** 디렉터리에 위치한다. <language> 부분은 IETF 포맷(https://en.wikipedia.org/wiki/BCP_47)에 있는 ASCII 문자열(보통 5개 문자)이어야 한다. LDML 파일을 생성한 이후, <language> 부분이 CUBRID 설정 파일인 **\$CUBRID/conf/cubrid_locales.txt** 에 추가되어야 한다. 이 파일에 기록된 언어의 순서가 로캘 라이브러리를 생성(컴파일)하고 구동 시 로캘을 읽는 순서임에 유의한다.

make_locale 스크립트는 새로 추가되는 로캘을 컴파일하고 그 데이터를 CUBRID 로캘 라이브러리 (**\$CUBRID/lib/**)에 추가하는데 사용된다.

LDML 파일은 UTF-8로 인코딩되며, 같은 LDML 파일에는 하나의 로캘 정보만 기록할 수 있다.

LDML 파일에 새로운 로캘을 추가하려면 다음이 요구된다.

- 캘린더 정보를 명시한다(CUBRID 날짜 포맷, 월 및 요일 이름의 다양한 포맷, 오전/오후에 대한 이름). CUBRID는 캘린더 정보를 명시하기 위해 그레고리력만 지원한다(일반적인 LDML은 CUBRID가 지원하지 않는 다른 캘린더 타입을 지원한다).
- 숫자 설정을 명시한다(자릿수 구분 기호)
- 알파벳을 제공한다(대소문자에 대한 규칙의 집합)
 - 옵션으로, 일부 콜레이션이 추가될 수 있다.
 - 또한 옵션으로, Windows CSQL 응용에 대한 콘솔 변환 규칙이 정의될 수 있다.

LDML 캘린더 정보

- 첫 번째 부분은 **DATE**, **DATETIME**, **TIME**, **TIMESTAMP**, **DATETIME WITH TIME ZONE** 및 **TIMESTAMP WITH TIME ZONE** 데이터와 문자열 간의 타입변환을 위한 기본 CUBRID 형식으로 이루어진다. 이 형식은 `TO_DATE()`, `TO_TIME()`, `TO_DATETIME()`, `TO_TIMESTAMP()`, `TO_CHAR()`, `TO_DATETIME_TZ()`, `TO_TIMESTAMP_TZ()` 함수에서 사용된다. 허용되는 형식 요소는 데이터 타입에 따라 다르며 `TO_CHAR()` 함수(Date/Time Format 1)에서 사용된다. 이 형식의 문자열에는 ASCII 문자만 허용된다. **DATE** 및 **TIME** 형식은 30바이트(문자), **DATETIME** 및 **TIMESTAMP** 형식은 48자, **DATETIME WITH TIME ZONE** 및 **TIMESTAMP WITH TIME ZONE** 은 70자까지 허용된다.
- <months>는 긴 형식과 축약된 형식 둘 다에 대한 월 이름이 필요하다. 허용되는 크기는 축약된 형식에 대해 15자(또는 60 바이트)이고 긴 형식에 대해 25자(또는 100바이트)이다.
- <days>는 긴 형식과 축약된 형식 둘 다에 대한 요일 이름이 필요하다. 허용되는 크기는 축약된 형식에 대해 10자(또는 40 바이트)이고 긴 형식에 대해 15자(또는 60 바이트)이다.
 - <dayperiods> 서브 트리는 오전/오후 형식의 변형(타입 속성에 따라 달라짐)에 대한 문자열을 정의한다. 허용되는 크기는 10자(또는 40바이트)이다.

월과 요일 이름은 긴 포맷과 축약된 포맷 둘 다 카멜 표기 형식(Camel case case format, 첫 문자는 대문자, 나머지 문자는 소문자)으로 명시되어야 한다. CUBRID는 최대 허용된 크기를 바이트 단위로 검사한다. 문자의 크기는 UTF-8 문자의 최대 크기에 해당하는 4바이트로만 계산되므로, 100개의 ASCII 문자로만 구성된 월 이름을 설정하는 것이 가능하다(25자 제한은 월 이름의 각 문자가 4바이트로 인코딩된 UTF-8 문자인 경우이다).

LDML 숫자 정보

- <symbols> 태그는 정수와 소수 분할 기호 문자와 자릿수 구분 기호 문자를 정의한다. CUBRID는 이 기호들에 대해 ASCII 기호만을 기대하며, 공백 문자는 허용하지 않는다. CUBRID는 3개의 숫자를 그룹지어 자릿수를 구분한다.

LDML 알파벳

LDML 알파벳은 로캘의 알파벳에 대한 대소문자 규칙을 정의한다. 'CUBRIDAlphabetMode' 속성은 문자에 대한 데이터의 기본 원본(primary source)을 정의한다. 일반적으로 이 속성은 "UNICODEDATAFILE"로 지정되어야 하는데, 이 값은 CUBRID가 유니코드 데이터 파일(**\$CUBRID/locales/data/unicodedata.txt**) 을 사용한다는 의미이다.

이 파일은 수정되어서는 안 되며, 특정 문자에 대한 변경은 LDML 파일에서 이루어져야 한다. 이러한 값이 설정되어 있는 경우, 코드포인트 65535까지의 모든 유니코드 문자가 대소문자 정보를 가지고 로딩된다. 'CUBRIDAlphabetMode'에 대해 허용된 다른 값은 "ASCII"로, ASCII 문자만이 소문자, 대문자 또는 대소문자 구분을 하지 않는 문자열 비교 함수에서 사용될 수 있다.

"ASCII"는 CUBRID가 모든 UTF-8 (4바이트) 인코딩 문자를 지원하는 기능에 영향을 주는 것이 아니라, 특정 문자가 대소문자를 구분하는 기능에 포함되지 않도록 제한할 뿐이다.

대소문자 구분 규칙은 선택적이며 문자 정보(UNICODEDATAFILE 또는 ASCII)의 기본 원본 중 가장 우선적으로 적용된다.

CUBRID는 대문자 규칙(<u> 태그) 정의와 소문자 규칙(<l> 태그) 정의를 허용한다. 대문자 규칙과 소문자 규칙 각각은 원본-대상(<s> = source, <d> destination)의 쌍으로 된 구성을 설정한다. 예를 들어, 다음은 문자 "A"의 소문자가 "aa"(2개의 "a" 문자)가 되도록 정의한다.

```
<l>
  <s>A</s>
  <d>aa</d>
</l>
```

LDML 콘솔 변환

Windows에서 콘솔은 UTF-8 인코딩을 지원하지 않으므로, CUBRID는 UTF-8 인코딩을 원하는 인코딩으로 번역하도록 허용한다. 로캘에 대한 콘솔 변환을 설정한 후에, 사용자는 CSQL 응용 프로그램을 시작하기에 앞서 'chcp' 명령(LDML에서 codepage 속성은 'windows_codepage' 속성과 매칭되어야 한다)을 사용하여 콘솔의 코드페이지를 설정해야 한다. CSQL 입출력 시 변환은 양방향으로 동작하지만, 설정된 코드페이지로서 변환될 수 있는 유니코드 문자로만 제한된다.

<consoleconversion> 원소는 선택적이며 CSQL이 인터랙티브 명령에서 텍스트를 출력(문자 인코딩)하는 방법을 지시하게 한다. 'type' 속성은 변환에 대한 계획(scheme)을 정의한다. 허용되는 값은 다음과 같다.

- ISO: 대상 코드 페이지가 단일 바이트 문자셋인 일반적인 계획(generic scheme)
- ISO88591: ISO-8859-1 문자셋에 대해 미리 준비된 단일 바이트 계획('file' 속성을 필요로 하지 않으며, 있으면 무시됨)
- ISO88599: ISO-8859-9 문자셋에 대해 미리 준비된 단일 바이트 계획(역시 'file' 속성을 필요로 하지 않음)

- DBCS: Double Byte Code-Set. 이것은 대상 코드페이지가 2바이트 문자셋인 일반적인 계획이다.

‘windows_codepage’는 CUBRID가 콘솔 변환을 자동으로 활성화하는 Windows 코드페이지에 대한 값이다. ‘linux_charset’은 UNIX 계열 시스템의 **LANG** 환경 변수에서 문자셋 부분과 동일한 값이다. Linux 콘솔에서는 원래의 CUBRID 문자셋을 사용하기를 권장한다.

‘file’ 속성은 ‘type’ 속성의 값이 “ISO”와 “DBCS”인 경우에만 요구되며 번역 정보(**\$CUBRID/locales/data/codepages/**)를 포함하는 파일이다.

LDML 콜레이션

콜레이션을 설정하는 것은 CUBRID에서 LDML 로컬을 추가할 때 가장 복잡한 작업이다. UTF-8 코드셋을 가진 콜레이션만 설정될 수 있다. CUBRID는 축약과 확장을 포함하는 UCA(Unicode Collation Algorithm, <http://www.unicode.org/reports/tr10/>)에 의해 명시된 대부분의 구성을 설정하는 것을 허용하지만, 콜레이션에 대한 속성은 대부분 ‘settings’ 속성을 통해 제어된다.

LDML 파일은 여러 개의 콜레이션을 포함할 수 있다. 콜레이션은 ‘include’ 태그를 사용하여 외부 파일에 의해 포함될 수 있다. ‘collations’ 태그의 ‘validSubLocales’ 속성은 외부 파일에 의한 외부 콜레이션이 포함될 때 로컬 컴파일을 제어하도록 허용하는 필터이다. 이 속성의 값은 로컬의 리스트 혹은 “*”이 될 수 있으며, “*”을 사용하면 서브트리에 있는 콜레이션이 모든 로컬에 추가된다.

하나의 콜레이션은 ‘collation’ 태그와 서브트리를 사용하여 정의된다. ‘type’ 속성은 콜레이션에 대한 이름을 나타내므로 CUBRID에 추가될 것이다. ‘settings’ 태그는 콜레이션의 속성들을 정의한다.

- ‘id’는 CUBRID에서 사용되는 (내부의) 숫자 식별자이다. 32부터 255까지의 정수이며 선택적이지만, 할당되지 않은 값의 명시적인 설정을 강력히 권장한다. **콜레이션 명명 규칙**을 참고한다.
- ‘strength’는 문자열 비교 방법의 척도이다. **콜레이션 속성**을 참고한다. 허용되는 값은 다음과 같다.
 - “quaternary”: 서로 같은 문자이나 다른 그래픽 기호이면 다르게 비교되지만, 다른 유니코드 코드포인트가 같게 비교될 수 있다.
 - “tertiary”: 서로 같은 문자의 그래픽 심볼은 동일하며, 대소문자를 구분하는 콜레이션이다.
 - “secondary”: 대소문자 구분 없는 콜레이션이며, 액센트가 있는 문자 사이의 비교 결과가 다르다.
 - “primary”: 액센트는 무시되며, 모든 문자는 기본(base) 문자로 비교된다.
- ‘caseLevel’: ‘strength’ 값이 “tertiary” 보다 작은 값을 가지는 콜레이션에 대해 대소문자 구분 있는 비교를 가능하게 하는 특별한 설정. 유효한 값은 “on” 또는 “off”이다.
- ‘caseFirst’: 대소문자 순서. 유효한 값은 “lower”, “upper” 그리고 “off”이다. “upper” 값은 대문자가 같은 문자의 소문자보다 앞선다는 의미이다.
- ‘CUBRIDMaxWeights’: 콜레이션에서 사용자에게 의해 정의된(customized) 코드포인트의 개수(또는 최종 코드포인트 + 1). 최대 값은 65536이다. 이 값을 늘리면 콜레이션 데이터의 크기가 늘어난다.

- ‘DUCETContractions’: 유효한 값은 “use” 또는 “ignore”이다. “use”가 명시되면 (\$CUBRID/locales/data/ducet.txt)에 의해 정의된 축약을 콜레이션에서 사용할 수 있다. “ignore”이면 정의된 축약을 무시한다.
- ‘TailoringContractions’: 위와 같으나 명시된 콜레이션 규칙으로부터 정의되거나 추출된(derived) 축약을 의미한다. 축약을 사용하면 보다 복잡한 콜레이션(더 느린 문자열 비교)이 된다.
- ‘CUBRIDExpansions’: 허용되는 값은 “use” 또는 “ignore”(기본값)이며 DUCET 파일과 사용자 정의 규칙 둘 다에서 콜레이션 확장을 사용함을 의미한다. ‘CUBRIDExpansions’은 콜레이션 속성에서 가장 큰 영향을 준다. 이것을 가능하게 하면 문자열을 비교할 때 여러 단계(콜레이션 세기까지)로 비교하게 되며, 더 “자연스러운” 정렬 순서를 얻을 수 있는 이점이 있지만 콜레이션 데이터를 크게 증가시킨다. **확장**을 참고한다.
- ‘backwards’: “on” 또는 “off” 값이 사용된다. 액센트에 대해 문자열의 끝에서 앞으로 비교되는 “french” 순서를 적용하기 위해 사용된다. 이 값은 ‘CUBRIDExpansions’이 “use”일 때만 적용된다.
- ‘MatchContractionBoundary’: “true” 또는 “false”. 이 값은 축약이 지정될 때 문자열 매칭에서의 동작을 설정하기 위해 확장과 축약을 가지는 콜레이션에서 사용된다.

콜레이션에 대한 주요 데이터는 DUCET 파일에서 읽혀진다. 이 단계 이후, 콜레이션은 “사용자 정의 규칙(tailoring rules)”에 의해 명시될 수 있다. 이 규칙들은 “<rules>” (LDML)와 “<cubridrules>” (CUBRID 전용)이다.

‘cubridrules’ 태그는 선택적이며 코드포인트 또는 코드포인트의 범위에 대한 가중치를 명시적으로 설정하는데 사용될 수 있다. ‘cubridrules’는 DUCET 파일로부터 기본(primary) 콜레이션을 로딩한 후 ‘<rules>’ 태그에 정의된 UCA 규칙이 적용되기 전에 적용된다. 각 규칙은 ‘<set>’ 태그로 닫힌다. 규칙이 하나의 유니코드 코드포인트만을 언급하는 경우, 코드포인트의 16진수 값을 포함하는 ‘<scp>’ 태그가 제공된다.

모든 사용 가능한 CUBRID 콜레이션은 다음과 같은 CUBRID 규칙을 포함한다.

```
<cubridrules>
  <set>
    <scp>20</scp>
    <w>[0.0.0.0]</w>
  </set>
</cubridrules>
```

이 규칙은 20(ASCII 띄어쓰기 문자)으로 시작하는 코드포인트의 가중치(UCA는 콜레이션 원소 당 네 개의 가중치를 정의)가 모두 0으로 설정된다는 것을 나타낸다. ‘<ecp>’ 태그가 없기 때문에 영향을 받는 유일한 코드포인트는 20이다. CUBRID에서 띄어쓰기 문자는 0으로 비교된다. ‘<set>’ 규칙에서 허용되는 태그는 다음과 같다.

- ‘<cp>’: 하나의 코드포인트에 대한 가중치를 설정하는 규칙

- ‘<ch>’: 하나의 문자에 대한 가중치를 설정하는 규칙. 위의 것과 비슷하지만, 코드포인트 대신 이것은 유니코드 문자(UTF-8 encoding)를 기대한다.
- ‘<scp>’: 코드포인트의 범위에 대한 가중치를 설정하는 규칙. 이것이 시작 코드포인트이다.
- ‘<sch>’: 문자의 범위에 대한 가중치를 설정하는 규칙. 이것이 시작 문자이다. 문맥에서, 문자의 순서는 유니코드 코드포인트에 의해 주어진다.
- ‘<ecp>’: 범위 규칙에 대한 끝 코드포인트.
- ‘<ech>’: 범위 규칙에 대한 끝 문자.
- ‘<w>’: 설정할 가중치(단일 값). 가중치는 16진수이다. 각 콜레이션 원소는 점으로 구분되는 네 개의 값을 가지며, 이 값들은 각괄호([])로 감싸진다. 10개의 콜레이션 원소까지 포함할 수 있다.
- ‘<wr>’: 범위에 대해 설정하는 시작 가중치. 선택적으로, 이 태그에는 ‘step’ 속성이 있는데 이는 각 코드포인트 뒤에 증가하는 단계를 설정한다. 기본 단계는 [0001.0000.0000.0000]인데, 이것은 시작 코드포인트에 대한 첫번째 가중치를 설정한 후, 하나의 값이 첫 번째 수준(primary level) 가중치에 추가되고 다음 코드포인트까지 범위로 설정됨을 의미하며, 이 절차는 마지막 코드포인트까지 반복된다.

예:

```
<cupridrules>
  <set>                                <!-- Rule 1 -->
    <scp>0</scp>
    <ecp>20</ecp>
    <w>[0.0.0.0]</w>
  </set>

  <set>                                <!-- Rule 2 -->
    <scp>30</scp>
    <ecp>39</ecp>
    <wr step="[1.0.0.0][1.0.0.0]">[30.0.0.0][30.0.0.0]</
wr>
  </set>

</cupridrules>
```

Rule 1은, 가중치 0을 포함하여 0부터 20까지의 코드포인트를 설정한다. Rule 2는, 30부터 39(10진수)까지의 코드포인트와 증가하는 가중치를 가진 두 개의 콜레이션 원소의 집합을 설정한다. 이 예에서, 코드포인트 39(문자 “9”)는 2개의 콜레이션 원소[39.0.0.0][39.0.0.0]를 보유하는 가중치를 가질 것이다.

‘<rules>’ 태그 또한 선택적인데, LDML과 UCA 명세에 따라 정해진다. 하위 태그의 의미는 다음과 같다.

- ‘<reset>’: 콜레이션 원소를 지정한다. 이 태그의 정의(다음에 나타나는 ‘<reset>’ 태그까지)에 의해 문자 순서에 대한 규칙들이 조정된다. 축약인지 확장인지에 따라 이 값은 단일 문자셋의 문자이거나 다중 문자셋의 문자가 될 수 있다. 기본적으로, 앵커 뒤에 오는 사용자 정의 규칙(tailoring rules)은 “뒤에(after)” 정렬된다(첫 번째 규칙의 원소는 앵커 뒤에 오며, 두 번째 규칙의 원소는 첫 번째 규칙에 있는 원소 뒤에 정렬됨). 선택적 속성 “before”가 존재하면, <reset> 뒤에 오는 첫 번째 규칙만 앵커 앞에 오는 반면, 두 번째와 이하 규칙들은 일반적인 “뒤(after)”의 정렬(두 번째 규칙에 있는 원소는 첫 번째 규칙에 있는 원소 뒤에 정렬됨)을 재개한다.
- ‘<p>’: 첫 번째 수준(primary level)에서 미리 조정된(tailored) 것 뒤(또는 앞, 앵커가 ‘before’ 속성을 가지는 경우)에 오는 문자.
- ‘<s>’: 두 번째 수준(secondary level)에서 미리 조정된(tailored) 것 뒤(또는 앞, 앵커가 ‘before’ 속성을 가지는 경우)에 오는 문자.
- ‘<t>’: 세 번째 수준(tertiary level)에서 미리 조정된(tailored) 것 뒤(또는 앞, 앵커가 ‘before’ 속성을 가지는 경우)에 오는 문자.
- ‘<i>’: 문자가 전의 것과 동등하게 정렬된다.
- ‘<pc>’, ‘<sc>’, ‘<tc>’, ‘<ic>’: ‘<p>’, ‘<s>’, ‘<t>’, ‘<i>’와 같지만 문자의 범위를 적용한다.
- ‘<x>’: 확장 문자를 지정한다.
- ‘<extend>’: 확장의 두 번째 문자를 지정한다.
- ‘<context>’: 규칙이 적용되는 문맥을 지정한다. 축약과 확장을 지정하는 변수.

LDML 규칙을 가지고 UCA를 조정하는 것에 대한 보다 자세한 정보는 <http://www.unicode.org/reports/tr35/tr35-collation.html>를 참고한다.

Footnotes

[1] (1,2)

글리프(glyph): 글자 하나의 모양에 대한 기본 단위로 문자의 모양이나 형태를 나타내는 그래픽 부호. 글리프는 눈에 보이는 모양을 지정하는 것으로서 하나의 글자에 대해 여러 개의 글리프가 존재할 수 있다.

트랜잭션과 잠금

이 장에서는 트랜잭션을 커밋하거나 롤백하는 방법뿐만 아니라 동시성과 복구 이슈에 대하여 논의한다.

다중 사용자 환경에서 데이터베이스의 무결성을 보호하고 사용자의 트랜잭션이 항상 정확하고 일관된 데이터를 확보할 수 있도록 보장하기 위해서는 접근과 갱신의 조정이 필수적이다. 충분한 조정이 없다면 데이터는 타당하지 않거나 순서에 어긋나게 갱신될 수 있다.

동일한 데이터에 대한 병렬 연산을 통제하는 방법은 트랜잭션이 진행되는 동안 데이터를 잠그고 트랜잭션이 끝나기 전까지 다른 트랜잭션에서 용납되지 않은 데이터 접근을 허용하지 않는 것이다. 게다가 특정 테이블에 대한 갱신이 있을 때 이것을 커밋하기 전까지 다른 이들이 볼 수 없도록 한다. 만약 갱신을 커밋하지 않기로 결정했다면 마지막으로 갱신을 커밋하거나 롤백한 이후부터 입력된 모든 질의문을 무효화할 수 있다.

CUBRID는 다중 버전 동시성 제어(Multiversion Concurrency Control)을 사용하여 데이터 액세스를 최적화한다. 수정 중인 데이터를 덮어쓰지 않는 대신 이전 데이터를 삭제됨으로 표시하고 새 버전을 다른 곳에 추가한다. 즉, 다른 트랜잭션에서 이전 버전을 잠그지 않은 상태로 계속 읽을 수 있다. 각 트랜잭션은 트랜잭션 또는 질의문 실행 시작 시 정의된 자신의 데이터베이스 스냅샷 정보를 확인할 뿐 아니라, 스냅샷에 자체 변경 사항만 적용하고 수행 중 일관성을 유지한다.

여기에서 소개하는 모든 예제는 `csql`로 실행한 것이다.

데이터베이스 트랜잭션

데이터베이스 트랜잭션은 CUBRID 질의문을 일관성(다중 사용자 환경에서 유효한 결과를 만들어내는 것)과 복구(시스템 실패와 같은 어떤 장애에도 커밋된 트랜잭션의 결과를 유지하는 것과, 어떤 장애에도 불구하고 중단된 트랜잭션은 데이터베이스로부터 무효화되는 것을 보장하는 것)의 단위로 그룹화한다. 하나의 트랜잭션은 데이터베이스에 접근하고 갱신하는 하나의 질의문 또는 여러 질의문으로 구성된다.

CUBRID는 많은 사용자가 동시에 데이터베이스에 접근하도록 하고 데이터베이스의 불일치를 방지하기 위하여 사용자 간 접근과 갱신을 관리한다. 예를 들어 데이터가 한 사용자에게 의해 갱신되었을 때 그 트랜잭션에 관련된 데이터의 변화는 갱신이 커밋될 때까지 다른 사용자나 데이터베이스에서 일어나는 다른 트랜잭션에 보이지 않는다. 트랜잭션이 커밋되지 않고 롤백될 수 있기 때문에 이 원칙은 중요하다.

트랜잭션 처리 결과를 확신할 때까지, 데이터베이스에 영구적으로 갱신하는 것을 연기할 수 있다. 또한 트랜잭션 처리 과정에서 응용 프로그램이나 컴퓨터 시스템에서 만족할 수 없는 결과나 실패가 발생하면 데이터베이스의 모든 갱신을 제거(**ROLLBACK**)할 수 있다. 트랜잭션의 끝은 **COMMIT WORK** 또는 **ROLLBACK WORK** 문으로 결정된다. **COMMIT WORK** 문은 데이터베이스의 모든 갱신을 영구적으로 만드는 반면에 **ROLLBACK WORK** 문은 트랜잭션에서 입력된 모든 갱신을 무효화시킨다.

트랜잭션 커밋

데이터베이스에서 일어난 갱신들은 **COMMIT WORK** 문이 주어지기 전까지 영구히 저장되지 않는다. “영구히(permanently)” 저장된다는 것은 디스크에 저장이 완료되는 것을 의미한다. 키워드 **WORK**는 생략이 가능하다. 추가로 데이터베이스의 다른 사용자는 변경이 영구히 반영되기 전까지는 변경 사항을 볼 수 없다. 예를 들어 테이블에 새로운 행을 삽입했을 때 데이터베이스 트랜잭션이 커밋되

기 전까지 그 행에 접근할 수 있는 것은 그 행을 삽입한 사용자뿐이다(**UNCOMMITTED INSTANCES** 격리 수준을 사용하면 다른 사용자가 일관성이 없는 커밋되지 않은 갱신을 볼 수도 있다).

트랜잭션이 커밋된 후에는 트랜잭션에서 획득한 모든 잠금이 해제된다.

```
COMMIT [WORK];
```

```
-- ;autocommit off
-- AUTOCOMMIT IS OFF
```

```
SELECT name, seats
FROM stadium WHERE code IN (30138, 30139, 30140);
```

name	seats
'Athens Olympic Tennis Centre'	3200
'Goudi Olympic Hall'	5000
'Vouliagmeni Olympic Centre'	3400

다음 **UPDATE** 문은 3개의 stadium의 seats 칼럼 값을 변경한다. 결과를 검토하기 위해 갱신이 일어나기 전에 현재의 값과 이름을 검색한다. 기본적으로 csql은 자동으로 autocommit으로 작동되므로, 예제에서는 autocommit 모드를 off로 설정한 후 동작을 시험한다.

```
UPDATE stadium
SET seats = seats + 1000
WHERE code IN (30138, 30139, 30140);
```

```
SELECT name, seats FROM stadium WHERE code in (30138, 30139, 30140);
```

name	seats
'Athens Olympic Tennis Centre'	4200
'Goudi Olympic Hall'	6000
'Vouliagmeni Olympic Centre'	4400

만약 갱신이 제대로 이루어 졌다면 변경을 영구적으로 만들 수 있다. 이때 아래처럼 **COMMIT WORK** 문을 사용한다.

```
COMMIT [WORK];
```


참고

CUBRID에서는 트랜잭션 처리 시 기본적으로 자동 커밋 모드로 지정된다.

자동 커밋 모드는 모든 SQL 문을 자동으로 커밋 또는 롤백하는 모드로서, 해당 SQL 문이 정상 수행되면 해당 트랜잭션을 자동 커밋하고, 오류가 발생하면 트랜잭션을 롤백한다.

이러한 자동 커밋 모드는 모든 인터페이스에서 지원되며, CCI, PHP, ODBC, OLE DB 인터페이스는 브로커 파라미터인 **CCI_DEFAULT_AUTOCOMMIT** 을 통해 응용 프로그램 시작 시의 자동 커밋 모드를 설정할 수 있다. 브로커 파라미터 설정이 생략될 경우 기본값은 **ON** 이다. CCI 인터페이스는 **cci_set_autocommit ()**, PHP 인터페이스는 **cubrid_set_autocommit ()** 함수를 이용하여 응용 프로그램 내에서 자동 커밋 모드 설정 여부를 변경할 수 있다.

CSQL 인터프리터에서 자동 커밋 모드를 설정하는 세션 명령어(**;**Autocommit****)에 대해서는 **세션 명령어** 를 참조한다.

트랜잭션 롤백

ROLLBACK WORK 문은 마지막 트랜잭션 이후의 모든 데이터베이스의 갱신을 제거한다. **WORK** 키워드는 생략 가능하다. 이것은 데이터베이스에 영구적으로 입력하기 전에 부정확하고 불필요한 갱신을 무효화할 수 있다. 트랜잭션 동안 획득한 모든 잠금은 해제된다.

```
ROLLBACK [WORK];
```

다음 예제는 동일한 테이블의 정의와 행을 수정하는 두 개의 명령을 보여주고 있다.

```
-- csql> ;autocommit off
CREATE TABLE code2 (
    s_name CHAR(1),
    f_name VARCHAR(10)
);
COMMIT;

ALTER TABLE code2 DROP s_name;
INSERT INTO code2 (s_name, f_name) VALUES ('D','Diamond');
```

```
ERROR: s_name is not defined.
```

`code` 테이블의 정의에서 `s_name` 칼럼이 이미 제거되었기 때문에 **INSERT** 문의 실행은 실패한다. `code` 테이블에 입력하려고 했던 데이터는 틀리지 않으나 테이블에서 칼럼이 잘못 제거되었다. 이 시점에서 `code` 테이블의 원래 정의를 복원하기 위해서 **ROLLBACK WORK** 문을 사용할 수 있다.

ROLLBACK WORK;

이후에 **ALTER CLASS** 명령을 다시 입력하여 *s_name* 칼럼을 제거하며, **INSERT** 문을 수정한다. 트랜잭션이 중단되었기 때문에 **INSERT** 명령은 다시 입력되어야 한다. 데이터베이스 갱신이 의도한 대로 이루어졌으면 변경을 영구화하기 위해 트랜잭션을 커밋한다.

```
ALTER TABLE code2 DROP s_name;
INSERT INTO code2 (f_name) VALUES ('Diamond');

COMMIT WORK;
```

세이브포인트와 부분 롤백

세이브포인트(savepoint)는 트랜잭션이 진행되는 중에 수립되는데, 트랜잭션에 의해 수행되는 데이터베이스 갱신을 세이브포인트 지점까지만 롤백할 수 있도록 하기 위해서이다. 이 연산을 부분 롤백(partial rollback)이라고 부른다. 부분 롤백에서는 세이브포인트 이후의 데이터베이스 연산(삽입, 삭제, 갱신 등)은 하지 않은 것으로 되고 세이브포인트 지점을 포함하여 앞서 진행된 트랜잭션의 연산은 그대로 유지된다. 부분 롤백이 실행된 후에 트랜잭션은 다른 연산을 계속 진행할 수 있다. 또는 **COMMIT WORK** 문이나 **ROLLBACK WORK** 문으로 트랜잭션을 끝낼 수도 있다. 세이브포인트는 트랜잭션에서 수행된 갱신을 커밋하는 것이 아님을 명심해야 한다.

세이브포인트는 트랜잭션의 어느 시점에서도 만들 수 있고 몇 개의 세이브포인트라도 어떤 주어진 시점에 사용될 수 있다. 특정 세이브포인트보다 앞선 세이브포인트로 부분 롤백이 수행되거나 **COMMIT WORK** 또는 **ROLLBACK WORK** 문으로 트랜잭션이 끝나면 특정 세이브포인트는 제거된다. 특정 세이브포인트 이후에 대한 부분 롤백은 여러 번 수행될 수 있다.

세이브포인트는 길고 복잡한 프로그램을 통제할 수 있도록 중간 단계를 만들고 이름을 붙일 수 있기 때문에 유용하다. 예를 들어, 많은 갱신 연산 수행 시 세이브포인트를 사용하면 실수를 했을 때 모든 문장을 다시 수행할 필요가 없다.

```
SAVEPOINT <mark>;

<mark>:
- a SQL identifier
- a host variable (starting with :)
```

같은 트랜잭션 내에 여러 개의 세이브포인트를 지정할 때 *mark* 를 같은 값으로 하면 마지막 세이브포인트만 부분 롤백에 나타난다. 그리고 앞의 세이브포인트는 제일 마지막 세이브포인트로 부분 롤백할 때까지 감춰졌다가 제일 마지막 세이브포인트가 사용된 후 없어지면 나타난다.

```
ROLLBACK [WORK] [TO [SAVEPOINT] <mark> ;

<mark>:
```

- a SQL identifier
- a host variable (starting **with** :)

앞에서는 **ROLLBACK WORK** 문이 마지막 트랜잭션 이후로 입력된 모든 데이터베이스의 갱신을 제거하였다. **ROLLBACK WORK** 문은 특정 세이프포인트 이후로 트랜잭션의 갱신을 되돌리는 부분 롤백에도 사용된다.

*mark*의 값이 주어지지 않으면 트랜잭션은 모든 갱신을 취소하면서 종료한다. 여기에는 트랜잭션에 만들어진 모든 세이프포인트도 포함한다. *mark*가 주어지면 지정한 세이프포인트 이후의 것은 취소되고, 세이프포인트를 포함한 이전의 것은 갱신 사항이 남는다.

다음 예제는 트랜잭션의 일부를 롤백하는 방법을 보여준다. 먼저 savepoint *SP1*, *SP2*를 설정한다.

```
-- csql> ;autocommit off

CREATE TABLE athlete2 (name VARCHAR(40), gender CHAR(1), nation_code
CHAR(3), event VARCHAR(30));
INSERT INTO athlete2(name, gender, nation_code, event)
VALUES ('Lim Kye-Sook', 'W', 'KOR', 'Hockey');
SAVEPOINT SP1;

SELECT * FROM athlete2;
INSERT INTO athlete2(name, gender, nation_code, event)
VALUES ('Lim Jin-Suk', 'M', 'KOR', 'Handball');

SELECT * FROM athlete2;
SAVEPOINT SP2;

RENAME TABLE athlete2 AS sportsman;
SELECT * FROM sportsman;
ROLLBACK WORK TO SP2;
```

위에서 *athlete2* 테이블의 이름 변경은 위의 부분 롤백에 의해서 롤백된다. 다음의 문장은 원래의 이름으로 질의를 수행하여 이것을 검증하고 있다.

```
SELECT * FROM athlete2;
DELETE FROM athlete2 WHERE name = 'Lim Jin-Suk';
SELECT * FROM athlete2;
ROLLBACK WORK TO SP2;
```

위에서 'Lim Jin-Suk'을 삭제한 것은 이후에 진행되는 `rollback work to SP2` 명령문에 의해서 취소되었다. 다음은 *SP1*으로 롤백하는 경우이다.

```
SELECT * FROM athlete2;
ROLLBACK WORK TO SP1;
```

```
SELECT * FROM athlete2;
COMMIT WORK;
```

커서 유지

응용 프로그램이 명시적인 커밋 혹은 자동 커밋 이후에도 **SELECT** 질의 결과의 레코드셋을 유지하여 다음 레코드를 읽을(fetch) 수 있도록 하는 것을 커서 유지(cursor holdability)라고 한다. 각 응용 프로그램에서 연결 수준(connection level) 또는 문장 수준(statement level)으로 커서 유지 기능을 설정할 수 있으며, 설정을 명시하지 않으면 기본으로 커서가 유지된다.

다음 코드는 JDBC에서 커서 유지를 설정하는 예이다.

```
// set cursor holdability at the connection level
conn.setHoldability(ResultSet.HOLD_CURSORS_OVER_COMMIT);

// set cursor holdability at the statement level which can override
the connection
PreparedStatement pstmt = conn.prepareStatement(sql,
                                         ResultSet.TYPE_SCROLL_SENSITIVE,
                                         ResultSet.CONCUR_UPDATABLE,

ResultSet.HOLD_CURSORS_OVER_COMMIT);
```

커밋 시점에 커서를 유지하지 않고 커서를 닫도록 설정하고 싶으면, 위의 예제에서 **ResultSet.HOLD_CURSORS_OVER_COMMIT** 대신 **ResultSet.CLOSE_CURSORS_AT_COMMIT** 를 설정한다.

CCI 로 개발된 응용 프로그램 역시 커서 유지가 기본 동작이며, 연결 수준에서 커서를 유지하지 않도록 설정한 경우 질의를 prepare할 때 **CCI_PREPARE_HOLDABLE** 플래그를 명시하면 해당 질의 수준에서 커서를 유지한다. CCI로 개발된 드라이버(PHP, PDO, ODBC, OLE DB, ADO.NET, Perl, Python, Ruby) 역시 커서 유지가 기본 동작이며, 커서 유지 여부의 설정을 지원하는지에 대해서는 해당 드라이버의 **PREPARE** 함수를 참고한다.

참고

- CUBRID 9.0 미만 버전까지는 커서 유지를 지원하지 않으며, 커밋이 발생하면 커서가 자동으로 닫히는 것이 기본 동작이다.
- CUBRID는 현재 `java.sql.XAConnection` 인터페이스에서 `ResultSet.HOLD_CURSORS_OVER_COMMIT`을 지원하지 않는다.

트랜잭션 종료 시의 커서 관련 동작

트랜잭션이 커밋되면 커서 유지로 설정되어 있더라도 모든 잠금은 해제된다.

트랜잭션이 롤백되면 결과 셋이 닫힌다. 이것은 커서 유지가 설정되어 현재 트랜잭션에서 유지되던 결과 셋이 닫힌다는 것을 의미한다.

```
rs1 = stmt.executeQuery(sql1);
conn.commit();
rs2 = stmt.executeQuery(sql2);
conn.rollback(); // 결과 셋 rs2와 rs1이 닫히게 되어 둘 다 사용하지 못하게 됨.
```

결과 셋이 종료되는 경우

커서가 유지되는 결과 셋은 다음의 경우에 닫힌다.

- 드라이버에서 결과 셋을 닫는 경우(예: rs.close() 등)
- 드라이버에서 statement를 닫는 경우(예: stmt.close() 등)
- 드라이버 연결 종료
- 트랜잭션을 롤백하는 경우(예: 자동 커밋 OFF 모드에서 사용자의 명시적인 롤백 호출, 자동 커밋 ON 모드에서 질의 실행 오류 발생 등)

CAS와의 관계

응용 프로그램에서 커서 유지로 설정되어 있다고 해도 응용 프로그램과 CAS와의 연결이 끊기면 결과 셋은 자동으로 닫힌다. 브로커 파라미터인 **KEEP_CONNECTION**의 설정 값은 결과 셋의 커서 유지에 영향을 미친다.

- KEEP_CONNECTION = ON: 커서 유지에 영향을 주지 않음.
- KEEP_CONNECTION = AUTO: 커서 유지되는 결과 셋이 열려 있는 동안 CAS가 재시작될 수 없음.

⚠ 경고

결과 셋을 닫지 않은 상태로 유지하는 만큼 메모리 사용량이 늘어날 수 있으므로 사용을 마친 결과 셋은 반드시 닫아야 한다.

i 참고

CUBRID 9.0 미만 버전까지는 커서 유지를 지원하지 않으며, 커밋이 발생하면 커서가 자동으로 닫힌다. 즉, **SELECT** 질의 결과의 레코드셋을 유지하지 않는다.

데이터베이스 동시성

다수의 사용자들이 데이터베이스에서 읽고 쓰는 권한을 가질 때, 한 명 이상의 사용자가 동시에 같은 데이터에 접근할 가능성이 있다. 데이터베이스의 무결성을 보호하고, 사용자와 트랜잭션이 항상 정확하고 일관된 데이터를 지니기 위해서는 다중 사용자 환경에서의 접근과 갱신에 대한 통제가 필수적이다. 적절한 통제가 없으면 데이터는 어긋난 순서로 부정확하게 갱신될 수 있다.

트랜잭션은 데이터베이스 동시성을 보장해야하며 각 트랜잭션은 적절한 결과를 보장해야한다. 한 번에 여러 트랜잭션이 실행될 때 트랜잭션 T_1 의 이벤트가 트랜잭션 T_2 의 이벤트에 영향을 미치지 않아야 한다. 이것은 격리를 의미한다. 트랜잭션 격리 수준은 트랜잭션이 다른 모든 동시 트랜잭션과 분리되는 정도이다. 격리 수준이 높으면 다른 트랜잭션의 간섭이 적음을 의미한다. 격리 수준이 낮으면 동시성이 높다는 것을 의미한다. 데이터베이스는 격리 레벨에 따라 어떤 잠금(lock)이 테이블과 레코드에 적용할 것인지 판별한다. 따라서 적절한 격리 수준을 설정하여 서비스 고유의 일관성 및 동시성 수준을 제어 할 수 있다.

트랜잭션 격리 수준 설정을 통해 트랜잭션 간 간섭을 허용할 수 있는 읽기 연산의 종류는 다음과 같다.

- **더티 읽기 (Dirty read)** : 트랜잭션 T_1 이 데이터 D 를 D' 으로 갱신한 후 커밋을 수행하기 전에 트랜잭션 T_2 가 D' 을 읽을 수 있다.
- **반복할 수 없는 읽기 (Non-repeatable read)** : 트랜잭션 T_1 이 데이터를 반복 조회하는 중에 다른 트랜잭션 T_2 가 데이터를 갱신 혹은 삭제하고 커밋하는 경우, 트랜잭션 T_1 은 수정된 값을 읽을 수 있다.
- **유령 읽기 (Phantom read)** : 트랜잭션 T_1 에서 데이터를 여러 번 조회하는 중에 다른 트랜잭션 T_2 가 새로운 레코드 E 를 삽입하고 커밋한 경우, 트랜잭션 T_1 은 E 를 읽을 수 있다.

이러한 간섭을 기반으로 SQL 표준은 트랜잭션 격리 수준을 네 가지로 정의한다.

- **READ UNCOMMITTED** 는 더티 읽기(dirty read), 반복할 수 없는 읽기(unrepeatable read), 유령 읽기(phantom read)를 허용한다.
- **READ COMMITTED** 는 더티 읽기를 허용하지 않으며 반복할 수 없는 읽기와 유령 읽기를 허용한다.
- **REPEATABLE READ** 는 더티 읽기와 반복할 수 없는 읽기를 허용하지 않으며 유령 읽기를 허용한다.
- **SERIALIZABLE** 은 읽기 연산 시 트랜잭션 간 간섭을 허용하지 않는다.

CUBRID가 제공하는 격리 수준

아래 표에서 격리 수준 옆에 있는 괄호 안의 숫자는 격리 수준을 설정할 때 격리 수준 명칭 대신 사용할 수 있는 번호이다.

사용자는 **트랜잭션 격리 수준 설정** 문을 사용하거나 CUBRID가 지원하는 동시성/잠금 파라미터를 이용하여 격리 수준을 설정할 수 있는데, 이에 관한 설명은 **동시성/잠금 파라미터**를 참조한다.

(O: YES, X: NO)

CUBRID 격리 수준 (isolation_level)	더티 읽기	반복할 수 없는 읽기	유령 읽기	조회 중인 테이블에 대한 스키마 갱신
SERIALIZABLE 격리 수준 (6)	X	X	X	X
REPEATABLE READ 격리 수준 (5)	X	X	X	X
READ COMMITTED 격리 수준 (4)	X	O	O	X

CUBRID 격리 수준의 기본값은 **READ COMMITTED 격리 수준** 이다.

다중 버전 동시성 제어(Multiversion Concurrency Control)

이전 CUBRID는 잘 알려진 2단계 잠금 프로토콜을 사용하여 격리 수준을 관리했다. 이 프로토콜에서는 동시에 연산 충돌이 발생하지 않도록 트랜잭션이 객체를 읽기 전에 공유 잠금을 획득하고, 갱신하기 전에 배타 잠금을 획득한다. 트랜잭션 *T1* 에 잠금이 필요한 경우 시스템에서 요청된 잠금이 기존 잠금과 충돌하는지 확인한다. 충돌이 발생하면 트랜잭션 *T1* 은 대기 상태가 되고 잠금이 지연된다. 다른 트랜잭션 *T2* 가 잠금을 해제하면 트랜잭션 *T1* 이 다시 시작되어 잠금을 획득한다. 잠금이 해제되면 해당 트랜잭션에서 새로운 잠금을 획득할 필요가 없다.

CUBRID 10.0에서는 2단계 잠금 프로토콜이 Multiversion Concurrency Control(MVCC) 프로토콜로 대체되었다. 2단계 잠금 프로토콜과 달리, MVCC는 동시 트랜잭션에서 수정 중인 객체에 액세스하여 읽는 것을 허용한다. MVCC는 행을 중복하여 갱신될 때마다 여러 버전을 생성한다. 주로 데이터베이스에서 읽기 연산이 많은 시나리오에 대해서는 읽기 연산을 허용하는 것이 중요하다. 객체를 갱신하기 전에는 여전히 배타 잠금이 필요하다.

MVCC는 데이터베이스의 일관된 시점을 제공하며, 특히 다른 동시성 방법보다 적은 성능 비용으로 진정한 **snapshot isolation** 을 구현할 수 있다.

버전 관리, 가시성 및 스냅샷

MVCC는 각 데이터베이스 행에 대해 여러 버전을 유지한다. 각 버전마다 MVCCID(쓰기 트랜잭션에 대한 고유 식별자)로 삽입자(inserter) 및 삭제자(deleter)가 표시된다. 이러한 마커(Marker)는 변경한 사용자를 파악하고 타임라인에 변경 사항을 표시하는 데 유용하다.

트랜잭션 *T1* 이 새로운 행을 삽입하면 첫 번째 버전이 생성되고 고유 식별자 *MVCCID1* 이 삽입 ID로 설정된다. MVCCID는 레코드 헤더에 메타데이터로 저장된다.

OTHER META-DATA

MVCCID1

RECORD DATA

T1 이 커밋할 때까지 다른 트랜잭션에서는 *T1* 이 삽입한 행을 볼 수 없어야 한다. MVCCID는 데이터베이스 변경 사항의 작성자를 식별하고 타임 라인에 배치하여 다른 트랜잭션이 변경 사항의 유효성을 알 수 있도록 한다. 이 경우 이 행을 검사하는 모든 트랜잭션은 *MVCCID1* 을 찾고 소유자가 여전히 활성 상태이므로 이 행을 볼 수 없다.

T1 이 커밋 된 후 새로운 트랜잭션 *T2* 가 행을 찾아 제거한다. *T2* 는 배타 잠금을 획득하는 대신, 다른 트랜잭션이 액세스할 수 있도록 해당 버전을 제거하지 않은 상태로 두고, 다른 트랜잭션이 변경할 수 없도록 버전을 삭제됨으로 표시한다. 다른 MVCCID를 추가하여 다른 트랜잭션이 삭제자를 식별할 수 있도록 한다.

OTHER META-DATA

MVCCID1

MVCCID2

RECORD DATA

T2 가 레코드 값 중 하나를 갱신하기로 결정하면 행을 신규 버전으로 갱신하고 이전 버전을 로그에 저장해야 한다. 새로운 행은 새로운 데이터, 트랜잭션 MVCCID(삽입 MVCCID) 및 이전 버전이 저장된 로그 엔트리의 주소로 구성된다. 행에 표시되는 내용은 다음과 같다.

HEAP 파일은 OID에 의해 식별되는 단일 행을 포함한다.

OTHER META-DATA

MVCCID_INS1

PREV_VERSION_LSA1

RECORD DATA

LOG 파일에는 로그 엔트리 체인이 있고, 각 로그 엔트리의 연두 부분은 수정되기 전의 힙 레코드를 포함한다.

LOG ENTRY META-DATA

OTHER META-DATA

META-DATA

MVCCID_INS2

PREV_VERSION_LSA2

RECORD DATA

LOG ENTRY META-DATA

OTHER META-DATA

META-DATA

MVCCID_INS3

NULL

RECORD DATA

다른 트랜잭션은 각 레코드의 삽입 및 삭제 MVCCID 값에 따라 결정되는 가시성 조건을 만족하는 레코드가 나올 때까지 다중 로그 레코드의 이전 버전 LSA의 로그 체인을 조사할 필요가 있다.

참고

- 예전 버전(10.0)에서는 갱신된 행의 이전 및 새 버전을 저장하는 데 힙(다른 OID)이 사용되었다. 실제로 이전 버전은 변경되지 않은 행이었으며, 여기에는 새 버전에 대한 OID 링크가 추가되었다. 새 버전과 이전 버전이 둘 다 힙에 저장되었다.

T2 만 갱신된 행을 볼 수 있으며, 다른 트랜잭션은 힙 행에서 획득한 LSA를 통해 로그 페이지에 있는 행 버전에 액세스할 수 있다. 실행 중인 트랜잭션에 의해 보이거나 보이지 않는 버전 속성을 **가시성 (visibility)** 이라 한다. 가시성 속성은 각 트랜잭션과 관련이 있고, 일부 트랜잭션은 이를 true로 간주하지만, 나머지 트랜잭션은 false로 간주할 수 있다.

T2 가 행 갱신을 수행한 후 커밋하기 전, 트랜잭션 T3 가 시작한 경우 T2 가 커밋한 후에도 T3 는 새 버전을 볼 수 없다. T3 의 버전 가시성은 T3 가 시작할 때 삽입자 및 삭제자의 상태에 따라 결정되며, T3 의 트랜잭션이 수행되는 동안 해당 상태가 유지된다.

사실상, 트랜잭션에 대한 모든 버전의 가시성은 트랜잭션이 시작된 이후에 발생하는 변경 사항의 영향을 받지 않는다. 또한 새로 추가되는 버전도 무시된다. 결과적으로 트랜잭션에서 보여진 버전 셋은 변경되지 않고 트랜잭션의 스냅샷으로 구성된다. 따라서 MVCC에 의해 제공된 **snapshot isolation** 는 트랜잭션의 모든 읽기 질의에 대해 일관된 뷰를 보장한다.

CUBRID에서 **스냅샷 (snapshot)** 은 모든 유효하지 않은 MVCCID의 필터이다. 스냅샷이 만들어지기 전에 커밋되지 않으면 MVCCID는 유효하지 않다. 새 트랜잭션을 시작할 때마다 스냅샷 필터가 갱신되지 않기 위해 두 경계(가장 낮은 활성 MVCCID와 가장 높은 커밋 MVCCID)를 통해 스냅샷이 정의된다. 이 경계 안에 있는 활성 MVCCID 값 목록만 저장된다. 스냅샷 이후 시작된 트랜잭션은 가장 높은 커밋 MVCCID보다 큰 MVCCID를 가지므로 자동으로 무효화된다. 가장 낮은 활성 MVCCID보다 낮은 MVCCID는 이미 커밋되었으므로 자동으로 유효하다.

버전 가시성을 결정하는 스냅샷 필터 알고리즘은 삽입 및 삭제에 사용되는 MVCCID 마커를 사용한다. 스냅샷은 힙에 저장된 마지막 버전을 검사하고 결과에 따라 힙에서 버전을 가져오거나 로그에서 이전 버전을 가져올 수 있으며 행을 무시할 수도 있다.

삽입MVCCID	이전 버전 LSA	삭제 MVCCID	스냅샷 테스트 결과
Not visible	NULL	None 또는 not visible	버전이 최신이며 가시적이지 않다. 이전 버전이 없으므로 행이 무시된다.

삽입MVCCID	이전 버전 LSA	삭제 MVCCID	스냅샷 테스트 결과
	LSA	None 또는 not visible	버전이 최신 이며 가시적이지 않다. 이전 버전이 있어 스냅샷은 행의 LSA를 확인해야한다.
None or visible	LSA or NULL	None 또는 not visible	버전이 가시적이며 행의 데이터가 패치(fetch)된다. 이전 버전이 있는지 여부는 중요하지 않다.
		Visible	버전이 너무 오래되고, 삭제되어 가시적이지 않다. 이전 버전이 있는지 여부는 중요하지 않다.

버전이 최신이지만 로그에 저장된 이전 버전이 있는 경우 이전 버전에서도 동일한 확인 과정이 반복된다. 더 이상 이전 버전이 없거나(해당 트랜잭션에 대한 행 체인 전부가 최신인 경우) 가시적인 버전을 발견하면 확인이 중지된다.

스냅샷의 작동 원리는 다음과 같다(전체 트랜잭션에서 동일한 스냅샷을 유지하기 위해 **REPEATABLE READ** 격리 수준 사용).

예제 1: 새로운 행 삽입

session 1	session 2
<pre> csql> ;autocommit off AUTOCOMMIT IS OFF csql> set transaction isolation level REPEATABLE READ; </pre>	<pre> csql> ;autocommit off AUTOCOMMIT IS OFF csql> set transaction isolation level REPEATABLE READ; </pre>

session 1

Isolation level set to:
REPEATABLE READ

session 2

Isolation level set to:
REPEATABLE READ

```
csql> CREATE TABLE
tbl(host_year integer,
nation_code char(3));
csql> COMMIT WORK;
```

```
-- insert a row without
committing
csql> INSERT INTO tbl VALUES
(2008, 'AUS');
```

```
-- current transaction sees
its own changes
```

```
csql> SELECT * FROM tbl;
```

```
      host_year  nation_code
=====
====
           2008   'AUS'
```

```
-- this snapshot should not
see uncommitted row
```

```
csql> SELECT * FROM tbl;
```

There **are no** results.

```
csql> COMMIT WORK;
```

session 1

session 2

```
-- even though inserter did
commit, this snapshot still
can't see the row
csql> SELECT * FROM tbl;
```

There **are no** results.

```
-- commit to start a new
transaction with a new
snapshot
csql> COMMIT WORK;
```

```
-- the new snapshot should
see committed row
csql> SELECT * FROM tbl;
```

```
      host_year  nation_code
=====
=====
                2008  'AUS'
```

예제 2: 행 삭제

session 1

session 2

```
csql> ;autocommit off
```

AUTOCOMMIT IS OFF

```
csql> set transaction
isolation level REPEATABLE
READ;
```

Isolation level set to:
REPEATABLE READ

```
csql> ;autocommit off
```

AUTOCOMMIT IS OFF

```
csql> set transaction
isolation level REPEATABLE
READ;
```

Isolation level set to:
REPEATABLE READ

session 1

session 2

```

csql> CREATE TABLE
tbl(host_year integer,
nation_code char(3));
csql> INSERT INTO tbl VALUES
(2008, 'AUS');
csql> COMMIT WORK;

```

```

-- delete the row without
committing
csql> DELETE FROM tbl WHERE
nation_code = 'AUS';

-- this transaction sees its
own changes
csql> SELECT * FROM tbl;

There are no results.

```

```

-- delete was not committed,
so the row is visible to this
snapshot
csql> SELECT * FROM tbl;

      host_year  nation_code
=====
=====
          2008    'AUS'

```

```

csql> COMMIT WORK;

```

```

-- delete was committed, but
the row is still visible to
this snapshot

```

session 1

session 2

```

csql> SELECT * FROM tbl;

      host_year  nation_code
=====
=====
              2008  'AUS'

-- commit to start a new
transaction with a new
snapshot
csql> COMMIT WORK;

-- the new snapshot can no
longer see deleted row
csql> SELECT * FROM tbl;

There are no results.

```

예제 3: 행 갱신

session 1

session 2

```

csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> set transaction
isolation level REPEATABLE
READ;

Isolation level set to:
REPEATABLE READ

```

```

csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> set transaction
isolation level REPEATABLE
READ;

Isolation level set to:
REPEATABLE READ

```

session 1

session 2

```

csql> CREATE TABLE
tbl(host_year integer,
nation_code char(3));
csql> INSERT INTO tbl VALUES
(2008, 'AUS');
csql> COMMIT WORK;

```

```

-- delete the row without
committing
csql> UPDATE tbl SET host_year
= 2012 WHERE nation_code =
'AUS';

```

```

-- this transaction sees new
version, host_year = 2012
csql> SELECT * FROM tbl;

```

host_year	nation_code
2012	'AUS'

```

-- update was not committed,
so this snapshot sees old
version

```

```

csql> SELECT * FROM tbl;

```

host_year	nation_code
2008	'AUS'

```

csql> COMMIT WORK;

```

session 1

session 2

```
-- update was committed, but
this snapshot still sees old
version
csql> SELECT * FROM tbl;

      host_year  nation_code
=====
=====
              2008  'AUS'

-- commit to start a new
transaction with a new
snapshot
csql> COMMIT WORK;

-- the new snapshot can see
new version, host_year = 2012
csql> SELECT * FROM tbl;

      host_year  nation_code
=====
=====
              2012  'AUS'
```

예제 4: 다양한 트랜잭션에서 각각 다른 버전이 보임

session 1

session 2

session 3

```
csql> ;autocommit
off
```

```
AUTOCOMMIT IS OFF
```

```
csql> set
transaction
isolation level
REPEATABLE READ;
```

```
Isolation level
```

```
csql> ;autocommit
off
```

```
AUTOCOMMIT IS OFF
```

```
csql> set
transaction
isolation
level REPEATABLE
READ;
```

```
Isolation level
```

```
csql> ;autocommit
off
```

```
AUTOCOMMIT IS OFF
```

```
csql> set
transaction
isolation
level REPEATABLE
READ;
```

```
Isolation level
```


session 1

```
set to:
REPEATABLE READ
```

session 2

```
set to:
REPEATABLE READ
```

session 3

```
set to:
REPEATABLE READ
```

```
csql> CREATE TABLE
tbl(host_year
integer,
nation_code
char(3));
csql> INSERT INTO
tbl VALUES (2008,
'AUS');
csql> COMMIT WORK;
```

```
-- update row
csql> UPDATE tbl
SET host_year =
2012 WHERE
nation_code =
'AUS';
```

```
csql> SELECT *
FROM tbl;
```

```
      host_year
nation_code
=====
=====
                2012
'AUS'
```

```
csql> SELECT *
FROM tbl;
```

```
      host_year
nation_code
=====
=====
                2008
'AUS'
```

```
csql> COMMIT WORK;
```

```
csql> UPDATE tbl
SET host_year =
2016 WHERE
```

```
csql> SELECT *
FROM tbl;
```

```
csql> SELECT *
FROM tbl;
```

session 1	session 2	session 3
<pre> nation_code = 'AUS'; csql> SELECT * FROM tbl; host_year nation_code ===== ===== 2016 'AUS' </pre>	<pre> host_year nation_code ===== ===== 2008 'AUS' </pre>	<pre> host_year nation_code ===== ===== 2012 'AUS' </pre>

VACUUM

각 갱신에 대해 새 버전을 생성하고 삭제 시 이전 버전을 유지하면 데이터베이스 크기가 무한으로 증가하여 데이터베이스에 큰 이슈가 발생할 수 있다. 따라서 이전 데이터를 제거하고 점유된 공간을 재사용하기 위한 회수(Cleanup) 시스템이 필요하다.

각 행의 버전은 다음과 같은 동일한 단계를 거친다.

1. 새로 삽입되었으나 커밋되지 않으면, 삽입자만 볼 수 있음
2. 커밋되었으면, 이전 트랜잭션에서는 볼 수 없으나 이후 트랜잭션에서는 볼 수 있음
3. 삭제되었으나 커밋되지 않으면, 다른 트랜잭션에서는 볼 수 있으나 삭제자는 볼 수 없음
4. 커밋되면, 이전 트랜잭션에서는 볼 수 있으나 이후 트랜잭션에서는 볼 수 없음
5. 모든 활성 트랜잭션에서 볼 수 없음
6. 데이터베이스에서 제거됨

회수 시스템의 역할은 5~6단계에서 회수할 버전을 획득하는 것이다. CUBRID에서는 이 시스템을 **VACUUM** 이라고 부른다.

VACUUM 시스템은 세 가지 원칙에 따라 개발되었다.

- **VACUUM** 은 정확하고 완전해야 한다. **VACUUM** 은 일부 사용자가 계속 볼 수 있는 데이터는 제거하지 않으며 이전 데이터는 하나도 놓치지 않는다.
- **VACUUM** 은 신중해야 한다. 회수 프로세스는 데이터베이스의 내용을 변경하기 때문에 수행 중인 트랜잭션의 동작에 간섭을 일으킬 수 있지만 이러한 가능성을 최소화해야 한다.

- **VACUUM** 은 빠르고 효율적이어야 한다. **VACUUM** 이 너무 느리거나 지연되기 시작하면 데이터베이스 상태가 악화되어 전체 성능에 영향을 미칠 수 있다.

이러한 원칙에 따라 **VACUUM** 구현은 다음과 같은 이유로 기존 복구 로깅을 사용했다.

- 복구 로깅에는 힙과 인덱스 변경 사항에 대한 복구 데이터의 주소가 유지된다. 그래서 데이터베이스를 스캔하지 않고 **VACUUM** 이 대상으로 바로 이동할 수 있다.
- 로그 데이터의 처리는 활성 Worker의 작업에 거의 간섭을 일으키지 않는다.

MVCCID 정보를 로깅된 데이터에 추가함으로써 **VACUUM** 요구 사항에 맞게 복구 로깅을 조정했다. 로그 엔트리를 처리할 준비가 되면 MVCCID에 따라 **VACUUM** 이 결정된다. 활성 트랜잭션에서 볼 수 있는 MVCCID는 처리되지 않는다. 시간이 지나면 각 MVCCID를 사용한 변경 사항을 모두 볼 수 없게 된다.

각 트랜잭션은 활성 상태로 간주되는 가장 오래된 MVCCID를 유지한다. 활성으로 간주되는 가장 오래된 MVCCID는 모든 트랜잭션 중 가장 오래된 MVCCID에 의해 결정된다. 이 값보다 오래된 것은 보이지 않으므로 **VACUUM** 이 회수할 수 있다.

VACUUM 병렬 수행

VACUUM 은 세 번째 원칙에 따라 빨라야 하며 활성 Worker보다 뒤처지면 안 된다. 시스템 작업 부하가 많은 경우 하나의 스레드에서 모든 **VACUUM** 작업을 처리할 수 없기 때문에 병렬 처리해야 한다.

병렬 처리를 수행하기 위해 로그 데이터를 고정 크기 블록으로 분할했다. 적절한 시기(최신의 MVCCID를 vacuum 처리 할 수 있음. 이는 블록에 기록된 모든 작업을 vacuum 할 수 있음을 의미함)가 되면 각 블록마다 하나의 vacuum 작업을 생성한다. vacuum 작업은 로그 블록에 있는 관련 로그 항목을 기반으로 데이터 공간을 회수하는 **VACUUM Worker** 들에 의해 선택된다. 로그 블록의 추적 및 vacuum 작업의 생성은 **VACUUM Master** 가 수행한다.

VACUUM 데이터

로그 블록에서 수집된 데이터는 vacuum 데이터 파일에 저장된다. 생성된 vacuum 작업은 나중에 수행되므로 vacuum 작업을 수행할 수 있을 때까지 데이터가 저장되어야 하며 서버가 비정상 종료하는 경우에도 유지되어야 한다. vacuum 작업은 빠짐없이 수행되어야 한다. 서버의 비정상 종료로 인해 전혀 수행되지 않는 경우를 피하기 위해 작업이 두 번 수행되기도 한다.

작업이 성공적으로 수행된 후 처리된 로그 블록의 수집된 데이터가 제거된다.

수집된 로그 블록 데이터는 vacuum 데이터에 바로 추가되지 않는다. vacuum 시스템에서 작업 중인 스레드(로그 블록 및 수집 데이터 생성)들의 간섭을 피하기 위해 래치-프리(latch-free) 버퍼가 사용된다. **VACUUM Master** 가 주기적으로 활성화되어 버퍼에 있는 모든 내용을 vacuum 데이터로 저장하고, 이미 처리된 데이터를 제거한 후 새로운 작업(사용 가능한 경우)을 생성한다.

VACUUM 작업

VACUUM 작업 실행 단계는 다음과 같다.

1. **로그 프리패치** : vacuum Master 또는 Worker가 작업으로 처리할 로그 페이지를 프리 패치한다.
2. **각 로그 레코드에 대해 다음 작업을 반복한다.**
 1. 로그 레코드를 읽는다.
 2. **삭제된 파일을 확인한다.** : 로그 레코드가 삭제된 파일을 가리키면 다음 로그 레코드로 이동한다.
 3. **인덱스 vacuum을 수행하고 힙 OID를 수집한다.**
 - 로그 레코드가 인덱스에 속해 있는 경우 바로 vacuum을 수행한다.
 - 로그 레코드가 힙에 속해 있는 경우 나중에 vacuum을 수행할 OID를 수집한다.
1. 수집된 OID에 따라 **힙 vacuum을 수행** 한다.
2. **작업을 완료한다.** vacuum 데이터에서 작업을 완료됨으로 표시한다.

로그 페이지 읽기를 쉽게 하고 vacuum 수행을 최적화하기 위해 여러 가지 방법이 수행되었다.

삭제된 파일 추적

트랜잭션에서 테이블 또는 인덱스가 삭제되면 일반적으로 해당 테이블을 잠궈 다른 트랜잭션이 액세스하지 못하도록 차단한다. 그러나 활성 트랜잭션과 달리, **VACUUM** worker는 활성 트랜잭션에 대한 간섭을 최소화해야 하고, 회수할 데이터가 있는 경우 **VACUUM** 시스템은 절대로 중지되면 안되므로 잠금 시스템을 사용하지 않는다. 또한, **VACUUM** 은 회수가 필요한 데이터를 건너뛸 수 없다. 이에 따른 두 가지 결과는 다음과 같다.

1. **VACUUM** 은 삭제자(dropper)가 커밋할 때까지 삭제된 테이블 또는 삭제된 인덱스에 속한 파일을 삭제하지 않는다. 트랜잭션에서 테이블을 삭제한 경우에도 해당 파일이 즉시 삭제되지 않고 계속 액세스 할 수 있다. 실질적인 삭제는 커밋한 이후로 연기된다.
2. 실제 파일이 삭제되기 전에 **VACUUM** 시스템에 알려야 한다. 삭제자(dropper)가 **VACUUM** 시스템에 알림을 보내고 확인을 기다린다. **VACUUM** 작업의 반복 주기는 매우 짧고 새롭게 삭제된 파일이 있는지 자주 확인하므로 삭제자(dropper)가 오랫동안 기다리지 않아도 된다.

파일이 삭제된 후에는 **VACUUM** 은 해당 파일에 속한 모든 로그 엔트리를 무시한다. **VACUUM** 에서 제거해도 된다고 결정할 때까지(아직 vacuum되지 않은 가장 오래된 MVCCID에 따라 결정됨) 삭제 시점이 표시된 MVCCID와 함께 파일 식별자가 영구 파일에 저장된다.

잠금 프로토콜

2단계 잠금 프로토콜에서는 동시에 연산 충돌이 발생하지 않도록 트랜잭션이 객체를 읽기 전에 공유 잠금을 획득하고, 갱신하기 전에 배타 잠금을 획득한다. 현재 CUBRID에서 사용하고 있는 MVCC 잠금 프로토콜은 행을 읽기 전에 공유 잠금이 필요하지 않다. 그러나 테이블 객체에 의도 공유 잠금은 해당 행을 읽을 때 계속 사용된다. 트랜잭션 *T1* 에 잠금이 필요한 경우 CUBRID에서 요청된 잠금이 기존 잠금과 충돌하는지 확인한다. 충돌이 발생하면 트랜잭션 *T1* 은 대기 상태가 되고 잠금이 지연된다. 다른 트랜잭션 *T2* 가 잠금을 해제하면 트랜잭션 *T1* 이 다시 시작되어 잠금을 획득한다. 잠금이 해제되면 해당 트랜잭션에서 새로운 잠금을 획득할 필요가 없다.

잠금의 단위

CUBRID는 잠금의 개수를 줄이기 위해서 단위 잠금(granularity locking) 프로토콜을 사용한다. 단위 잠금 프로토콜에서는 잠금 단위의 크기에 따라 계층으로 모델화되며, 행 잠금(row lock), 테이블 잠금(table lock), 데이터베이스 잠금(database lock)이 있다. 이때, 단위가 큰 잠금은 작은 단위의 잠금을 내포한다.

잠금을 설정하고 해제하는 과정에서 발생하는 성능 손실을 잠금 비용(overhead)이라고 하는데, 큰 단위보다 작은 단위의 잠금을 수행할 때 이러한 잠금 비용이 높아지고 대신 트랜잭션 동시성은 향상된다. 따라서, CUBRID는 잠금 비용과 트랜잭션 동시성을 고려하여 잠금 단위를 결정한다. 예를 들어, 한 트랜잭션이 테이블의 모든 행들을 조회하는 경우 행 단위로 잠금을 설정/해제하는 비용이 너무 높으므로 차라리 해당 테이블에 잠금을 설정한다. 이처럼 테이블에 잠금이 설정되면 트랜잭션 동시성이 저하되므로, 동시성을 보장하려면 풀 스캔(full scan)이 발생하지 않도록 적절한 인덱스를 사용해야 할 것이다.

이와 같은 잠금 관리를 위해 CUBRID는 잠금 에스컬레이션(lock escalation) 기법을 사용하여 설정 가능한 단위 잠금의 수를 제한한다. 예를 들어, 한 트랜잭션이 행 단위에서 특정 개수 이상의 잠금을 가지고 있으면 시스템은 계층적으로 상위 단위인 테이블에 대해 잠금을 요청하기 시작한다. 단, 상위 단위로 잠금 에스컬레이션을 수행하기 위해서는 어떤 트랜잭션도 상위 단위 객체에 대한 잠금을 가지고 있지 않아야 한다. 그래야만 잠금 변환에 따른 교착 상태(deadlock)를 예방할 수 있다. 이때, 작은 단위에서 허용하는 잠금 개수는 시스템 파라미터 **lock_escalation** 을 통해 설정할 수 있다.

잠금 모드의 종류와 호환성

CUBRID는 트랜잭션이 수행하고자 하는 연산의 종류에 따라 획득하고자 하는 잠금 모드를 결정하며, 다른 트랜잭션에 의해 이미 선점된 잠금 모드의 종류에 따라 잠금 공유 여부를 결정한다. 이와 같은 잠금에 대한 결정은 시스템이 자동으로 수행하며, 사용자에게 의한 수동 지정은 허용되지 않는다.

CUBRID의 잠금 정보를 확인하기 위해서는 **cubrid lockdb db_name** 명령어를 사용하며, 자세한 내용은 **lockdb** 을 참고한다.

· 공유 잠금(shared lock, S_LOCK, MVCC 프로토콜에서는 더 이상 사용 안 함)

객체에 대해 읽기 연산을 수행하기 전에 획득하며, 여러 트랜잭션이 동일 객체에 대해 획득할 수 있는 잠금이다.

트랜잭션 T1이 특정 객체에 대해 읽기 연산을 수행하기 전에 공유 잠금을 먼저 획득한다. 이때, 트랜잭션 T2, T3 은 동시에 그 객체에 대해 읽기 연산을 수행할 수 있으나 갱신 연산을 수행할 수 없다.

참고

- CUBRID 10.0에서는 MVCC를 사용하므로 공유 잠금은 거의 사용되지 않는다. 현재는 내부 데이터베이스 연산에서 행 또는 인덱스 키가 수정되는 것을 방지하는 데 주로 사용된다.

· 배타 잠금(Exclusive lock, X_LOCK)

객체에 대해 갱신 연산을 수행하기 전에 획득하며, 하나의 트랜잭션만 획득할 수 있는 잠금이다.

트랜잭션 T1 이 특정 객체 X에 대해 갱신 연산을 수행하기 전에 배타 잠금을 먼저 획득하고, 갱신 연산을 완료하더라도 트랜잭션 T1 이 커밋될 때까지 배타 잠금을 해제하지 않는다. 따라서, 트랜잭션 T2, T3 은 트랜잭션 T1 이 배타 잠금을 해제하기 전까지는 X에 대한 읽기 연산도 수행할 수 없다.

· 의도 잠금(내재된 잠금, Intent lock)

특정 단위의 객체에 걸리는 잠금을 보호하기 위하여 이 객체보다 상위 단위의 객체에 내재적으로 설정하는 잠금을 의미한다.

예를 들어, 특정 행에 공유 잠금이 요청되면 이보다 계층적으로 상위에 있는 테이블에도 의도 공유 잠금을 함께 설정하여 다른 트랜잭션에 의해 테이블이 잠금되는 것을 예방한다. 따라서, 의도 잠금은 계층적으로 가장 낮은 단위인 행에 대해서는 설정되지 않으며, 이보다 높은 단위의 객체에 대해서만 설정된다. 의도 잠금의 종류는 다음과 같다.

◦ 의도 공유 잠금(Intent shared lock, IS_LOCK)

특정 행에 공유 잠금이 설정됨에 따라 상위 객체인 테이블에 의도 공유 잠금이 설정되면, 다른 트랜잭션은 칼럼을 추가하거나 테이블 이름을 변경하는 등의 테이블 스키마를 변경할 수 없고, 모든 행을 갱신하는 작업을 수행할 수 없다. 그러나 일부 행을 갱신하는 작업이나, 모든 행을 조회하는 작업은 허용된다.

◦ 의도 배타 잠금(Intent exclusive lock, IX_LOCK)

특정 행에 배타 잠금이 설정됨에 따라 상위 객체인 테이블에 의도 배타 잠금이 설정되면, 다른 트랜잭션은 테이블 스키마를 변경할 수 없고, 모든 행을 갱신하는 작업은 물론, 모든 행을 조회하는 작업은 수행할 수 없다. 그러나, 일부 행을 갱신하는 작업은 허용된다.

- 공유 의도 배타 잠금(shared with intent exclusive lock, SIX_LOCK)

계층적으로 더 낮은 모든 객체에 설정된 공유 잠금을 보호하고, 계층적으로 더 낮은 일부 객체에 대한 의도 배타 잠금을 보호하기 위하여 상위 객체에 내재적으로 설정되는 잠금이다.

테이블에 공유 의도 배타 잠금이 설정되면, 다른 트랜잭션은 테이블 스키마를 변경할 수 없고, 모든 행/일부 행을 갱신할 수 없으며, 모든 행을 조회할 수 없다. 그러나, 일부 행을 조회하는 작업은 허용된다.

- 스키마 잠금

DDL 작업을 수행할 때 스키마 잠금을 획득한다.

- 스키마 안정 잠금(schema stability lock, SCH_S_LOCK)

질의 컴파일을 수행하는 동안 획득되며 질의에 포함된 스키마가 다른 트랜잭션에 의해 수정되지 않음을 보장한다.

- 스키마 수정 잠금(schema modification lock, SCH_M_LOCK)

DDL(**ALTER/CREATE/DROP**)을 실행하는 동안 획득되며 다른 트랜잭션이 수정된 스키마에 접근하는 것을 방지한다.

ALTER, CREATE INDEX 등 일부 DDL 연산은 **SCH_M_LOCK** 을 직접 획득하지 않는다. 예를 들어 필터링된 인덱스를 생성할 때, CUBRID는 필터링 표현식에 대한 타입 검사를 수행한다. 이 기간 동안, 대상 테이블에 유지되는 잠금은 다른 타입 검사 연산의 경우처럼 **SCH_S_LOCK** 이다. 그런 다음 잠금이 **SIX_LOCK** 으로 업그레이드되고(다른 트랜잭션이 대상 테이블 행을 수정할 수 없지만 계속 읽을 수는 있음), 마지막으로 테이블 스키마를 변경하기 위해 **SCH_M_LOCK** 이 요청된다. 이러한 방식은 DDL 연산이 컴파일되고 수행되는 동안 다른 트랜잭션이 연산을 수행하는 것을 허용하여, 동시성을 높일 수 있다는 이점이 있다.

하지만 이 방식은 같은 테이블에 동시에 DDL 연산을 수행할 때 교착 상태를 회피할 수 없다는 단점 또한 존재한다. 인덱스를 로딩함으로 인한 교착 상태의 예는 다음과 같다. 그러나 이 방식은 같은 테이블에 동시에 DDL 연산을 수행할 때 교착 상태를 회피할 수 없다는 단점 또한 존재한다. 인덱스 로딩으로 인한 교착 상태의 예는 다음과 같다.

T1

T2

```
CREATE INDEX i_t_i on t(i)
WHERE i > 0;
```

```
CREATE INDEX i_t_j on t(j)
WHERE j > 0;
```

“i > 0” 경우의 타입 검사중에 SCH_S_LOCK.

“j > 0” case.”j > 0” 타입 검사중에
SCH_S_LOCK

인덱스 로딩 중에 SIX_LOCK.

SIX_LOCK을 요구하나 T1이 SIX_LOCK의 반환
을 대기

SCH_M_LOCK을 요구하나 T2가 SCH_S_LOCK의
반환을 대기

· 특수 잠금

CUBRID 10.2 시스템 내부적으로 특수한 오퍼레이션에서 사용하는 새로운 유형의 잠금을 소개한다.

◦ 대량갱신잠금(Bulk update lock, BU_LOCK)

이 잠금은 데이터베이스에 대량의 데이터를 삽입하는 경우에 사용하도록 설계되었다. **CUBRID 10.2** 를 기준으로, 이 잠금은 테이블에 행을 load하는 동안 **loaddb** 과정에서 배타적으로 사용된다. **BU_LOCK** 은 자기 자신과 **SCH_S_LOCK** 에 호환된다. 결과적으로 이 잠금은 다른 어떤 **SELECT/DML/DDI** 구문도 동일한 테이블에 접근하지 못한다는 것을 보장한다. 그러나 둘 이상의 **loaddb** 가 하나의 테이블에 행을 load할 수는 있다. **BU_LOCK** 잠금 보유자는 행 잠금을 요구하진 않는다.

i 참고

잠금에 대해 요약하면 다음과 같다.

- 잠금 대상 객체에 대해 행(인스턴스)과 스키마(클래스)가 있다. 사용된 객체 종류를 기준으로 잠금을 나누면 다음과 같다.
 - 행 잠금: **S_LOCK, X_LOCK**
 - 의도/스키마 잠금: **IX_LOCK, IS_LOCK, SIX_LOCK, SCH_S_LOCK, SCH_M_LOCK**
 - 특수 잠금: **BU_LOCK**
- 모든 유형의 잠금은 서로 영향을 미친다.

위에서 설명한 잠금들의 호환 관계(lock compatibility)를 정리하면 아래의 표와 같다. 호환된다는 것은 잠금 보유자(lock holder)가 특정 객체에 대해 획득한 잠금과 중복하여 잠금 요청자(lock requester)가 잠금을 획득할 수 있다는 의미이다.

잠금 호환성

- **NULL**: lock이 존재하는 상태.

(O: TRUE, X: FALSE)

		잠금 보유자(Lock holder)								
		NULL	SCH-S	IS	S	IX	BU	SIX	X	SCH-M
잠금 요청자 (Lock requester)	NULL	O	O	O	O	O	O	O	O	O
	SCH-S	O	O	O	O	O	O	O	O	X
	IS	O	O	O	O	O	X	O	X	X
	S	O	O	O	O	X	X	X	X	X
	IX	O	O	O	X	O	X	X	X	X

BU	O	O	X	X	X	O	X	X	X
SIX	O	O	O	X	X	X	X	X	X
X	O	O	X	X	X	X	X	X	X
SCH-M	O	X	X	X	X	X	X	X	X

잠금 변환 테이블

· **NULL**: 아무 잠금도 없는 상태

		획득 잠금 모드(Granted lock mode)								
		NULL	SCH-S	IS	S	IX	BU	SIX	X	SCH-M
요청 잠금 모드 (Requested lock mode)	NULL	NULL	SCH-S	IS	S	IX	BU	SIX	X	SCH-M
	SCH-S	SCH-S	SCH-S	IS	S	IX	BU	SIX	X	SCH-M
	IS	IS	IS	IS	S	IX	X	SIX	X	SCH-M
	S	S	S	S	S	SIX	X	SIX	X	SCH-M
	IX	IX	IX	IX	SIX	IX	X	SIX	X	SCH-M
	BU	BU	BU	BU	X	BU	BU	BU	X	SCH-M

SIX	SIX	SIX	SIX	SIX	SIX	X	SIX	X	SCH-M
X	X	X	X	X	X	X	X	X	SCH-M
SCH-M	SCH-M	SCH-M	SCH-M	SCH-M	SCH-M	SCH-M	SCH-M	SCH-M	SCH-M

잠금 사용 예제

다음 예제에서는 REPEATABLE READ(5) 격리 수준이 사용된다. READ COMMITTED의 행 갱신 규칙은 다양하며, 다음 섹션에서 설명한다. 예제에서는 기존 잠금을 보여주기 위해 lockdb 유틸리티를 사용한다.

잠금 예제: 다음 예제에서는 REPEATABLE READ(5) 격리 수준이 사용되며 동일한 행에서 읽기 및 쓰기가 차단되지 않는다는 것을 보여준다. 갱신 충돌이 시도되고 두 번째 갱신자(updater)가 차단된다. 트랜잭션 T1이 커밋될 때 T2의 차단이 해제되지만 격리 수준의 제약 사항으로 인해 갱신은 허용되지 않는다. T1이 롤백하는 경우 T2가 갱신을 진행할 수 있다.

T1	T2	설명

```
csql> ;au off
csql> SET
TRANSACTION
ISOLATION LEVEL
REPEATABLE READ;
```

```
csql> ;au off
csql> SET
TRANSACTION
ISOLATION LEVEL
REPEATABLE READ;
```

AUTOCOMMIT OFF,
REPEATABLE READ 로 설정

```
csql> CREATE TABLE
tbl(a INT PRIMARY
KEY, b INT);

csql> INSERT INTO
tbl
VALUES (10,
10),
(30,
30),
```

T1	T2	설명
<pre> (50, 50), (70, 70); csql> COMMIT; </pre>		

```

csql> UPDATE tbl
SET a = 90 WHERE a
= 10;

```

a = 10인 열의 첫번째 버전이 잠기고 갱신됨. a = 90 인 열의 새로운 버전이 생성되고 잠김

```
cubrid lockdb:
```

```

OID = 0|
623| 4
Object type:
Class = tbl.
Total mode of
holders =
IX_LOCK,
Total mode
of waiters =
NULL_LOCK.
Num holders= 1,
Num blocked-
holders= 0,
Num waiters=
0
LOCK HOLDERS:
Tran_index
= 1,
Granted_mode =
IX_LOCK

```

```

OID = 0|
650| 5
Object type:
Instance of class
( 0| 623| 4)
= tbl.
MVCC info: insert
ID = 5, delete ID
= missing.
Total mode of
holders =

```

T1	T2	설명
		<pre> X_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: Tran_index = 1, Granted_mode = X_LOCK OID = 0 650 1 Object type: Instance of class (0 623 4) = tbl. MVCC info: insert ID = 4, delete ID = 5. Total mode of holders = X_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: Tran_index = 1, Granted_mode = X_LOCK </pre>
	<pre> csql> SELECT * FROM tbl WHERE a <= 20; </pre>	T2는 $a \leq 20$ 인 조건을 만족하는 모든 열을 읽음. 갱신을 수행한 T1이 커밋을 하지 않았기 때문에 T2는 $a=10$인 열을

T1	T2	설명
	<pre> a b ===== ===== 10 10 </pre>	계속 보게되고 잠금을 하지는 않음 <pre> cubrid lockdb: OID = 0 623 4 Object type: Class = tbl. Total mode of holders = IX_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 2, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: Tran_index = 1, Granted_mode = IX_LOCK Tran_index = 2, Granted_mode = IS_LOCK </pre>

```

csql> UPDATE tbl
      SET a = a +
100
      WHERE a <=
20;

```

T2는 $a \leq 20$ 인 모든 열에 대해 갱신을 시도한다. T2가 클래스의 잠금을 IX_LOCK로 업그레이드하고, 이미 잠긴 $a = 10$ 인 열의 갱신을 시도하지만, T1에 의해 이미 잠긴 상태이므로 T2는 차단된다.

```

cubrid lockdb:

OID =  0|
623|   4
Object type:
Class = tbl.
Total mode of

```

T1	T2	설명
		<pre> holders = IX_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 2, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: Tran_index = 1, Granted_mode = IX_LOCK Tran_index = 2, Granted_mode = IX_LOCK OID = 0 650 5 Object type: Instance of class (0 623 4) = tbl. MVCC info: insert ID = 5, delete ID = missing. Total mode of holders = X_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: Tran_index = 1, Granted_mode = X_LOCK OID = 0 650 1 </pre>

T1	T2	설명
		<pre>Object type: Instance of class (0 623 4) = tbl. MVCC info: insert ID = 4, delete ID = 5. Total mode of holders = X_LOCK, Total mode of waiters = X_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 1 LOCK HOLDERS: Tran_index = 1, Granted_mode = X_LOCK LOCK WAITERS: Tran_index = 2, Blocked_mode = X_LOCK</pre>
<pre>csql> COMMIT;</pre>		T1의 잠금이 해제되었다.
	<pre>ERROR: Serializable conflict due to concurrent updates</pre>	T2가 차단에서 해제되어 T1이 이미 갱신한 개체의 갱신을 시도한다. REPEATABLE READ 격리 수준에서 이것은 허용되지 않고 오류가 전송됨

unique 제약 조건을 보호하기 위한 잠금

이전 CUBRID 버전의 2단계 잠금 프로토콜은 unique 제약 조건과 상위 격리 제한을 보호하기 위해 인덱스 키 잠금을 사용했다. CUBRID 10.0에서는 키 잠금이 제거되었다. 격리 수준 제한은 MVCC 스냅샷으로 해결되었지만 unique 제약 조건에는 여전히 특정 유형의 보호가 필요했다.

MVCC는 행처럼 고유 인덱스가 동시에 여러 버전을 유지하므로 각각 다른 트랜잭션에서 볼 수 있다. 여러 버전 중 하나는 마지막 버전이고, 나머지 버전들은 비가시화된 후 **VACUUM**에 의해 제거될 때까지 일시적으로 유지된다. unique 제약 조건을 보호하는 규칙은 키를 수정하려는 모든 트랜잭션이 키의 마지막 버전을 잠그는 것이다.

아래 예제는 잠금을 통해 unique 제약 조건 위반 방지를 위해 REPEATABLE READ 격리 수준을 사용한다.

T1	T2	설명
<pre>csql> ;au off csql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;</pre>	<pre>csql> ;au off csql> SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;</pre>	AUTOCOMMIT OFF, REPEATABLE READ로 설정
<pre>csql> CREATE TABLE tbl(a INT PRIMARY KEY, b INT); csql> INSERT INTO tbl VALUES (10, 10), (30, 30), (50, 50), (70, 70); csql> COMMIT;</pre>		T1이 테이블에 새로운 열을 삽입하고 잠금으로써 기본키 20은 보호된다.

T1	T2	설명

```
csql> INSERT INTO
tbl VALUES (20,
20);
```

```
INSERT INTO tbl
VALUES (20, 120);
```

T2는 테이블에 새로운 열을 삽입하고 잠금을 요청한다. 하지만, T2가 기본키에 새로운 열을 삽입하려고 할 때, 이미 기본키 20이 존재하기 때문에 T2는 T1이 커밋할 때까지 차단된다.

```
cubrid lockdb:
```

```
OID = 0|
623| 4
Object type:
Class = tbl.
Total mode of
holders =
IX_LOCK,
Total mode of
waiters =
NULL_LOCK.
Num holders= 2,
Num blocked-
holders= 0,
Num waiters=
0
LOCK HOLDERS:
Tran_index
= 1,
Granted_mode =
IX_LOCK
Tran_index
= 2,
Granted_mode =
IX_LOCK
```

```
OID = 0|
650| 5
Object type:
Instance of class
```

T1	T2	설명
		<pre> (0 623 4) = tbl. MVCC info: insert ID = 5, delete ID = missing. Total mode of holders = X_LOCK, Total mode of waiters = X_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 1 LOCK HOLDERS: Tran_index = 1, Granted_mode = X_LOCK LOCK WAITERS: Tran_index = 1, Blocked_mode = X_LOCK OID = 0 650 6 Object type: Instance of class (0 623 4) = tbl. MVCC info: insert ID = 6, delete ID = missing. Total mode of holders = X_LOCK, Total mode of waiters = NULL_LOCK. Num holders= 1, Num blocked- holders= 0, Num waiters= 0 LOCK HOLDERS: </pre>

T1	T2	설명
		<pre>Tran_index = 2, Granted_mode = X_LOCK</pre>

T1의 잠금이 해제된다.

```
COMMIT;
```

```
ERROR: Operation
would have caused
one or more
unique constraint
violations.
INDEX
pk_tbl_a(B+tree:
0|186|640)
ON CLASS
tbl(CLASS_OID: 0|
623|4).
key: 20(OID: 0|
650|6).
```

차단이 해제된 T2는 T1이 커밋한 키로 인해 unique 제약 위반 오류가 발생한다.

트랜잭션 교착 상태(deadlock)

교착 상태(deadlock)는 둘 이상의 트랜잭션이 서로 맞물려 상대방의 잠금이 해제되기를 기다리는 상태이다. 이러한 교착 상태에서는 서로가 상대방의 작업 수행을 차단하기 때문에 CUBRID는 트랜잭션 중 하나를 롤백시켜 교착 상태를 해결한다. 롤백되는 트랜잭션은 일반적으로 가장 적은 갱신을 수행한 것인데 보통 가장 최근에 시작된 트랜잭션이다. 시스템에 의해 트랜잭션이 롤백되자마자 그 트랜잭션이 가지고 있던 잠금이 해제되고 교착 상태에 있던 다른 트랜잭션이 진행되도록 허가된다.

이러한 교착 상태 발생은 예측할 수 없지만 가급적 교착 상태가 발생하지 않도록 하려면, 인덱스를 설정하여 잠금이 설정되는 범위를 줄이거나 트랜잭션을 짧게 만들거나 트랜잭션 격리 수준(isolation level)을 낮게 설정하는 것이 좋다.

에러 심각성 수준을 설정하는 시스템 파라미터인 **error_log_level**의 값을 NOTIFICATION으로 설정하면 교착 상태 발생 시 서버 에러 로그 파일에 잠금 관련 정보가 기록된다.

이전 버전에 비해 CUBRID 10.0은 더 이상 인덱스 키 잠금을 사용하여 인덱스를 읽고 쓰지 않으므로 교착 상태 발생이 현저히 줄어들었다. 교착 상태가 자주 발생하지 않는 또 다른 이유는 이전 CUBRID 버전은 일정 범위의 인덱스를 읽을 때 높은 격리 수준으로 인해 여러 객체들을 잠겼지만, CUBRID 10.0은 더 이상 잠금을 사용하지 않기 때문이다.

그러나 서로 다른 두 개의 트랜잭션에서 동일한 객체를 다른 순서로 갱신할 경우 여전히 교착 상태가 발생할 가능성이 남아 있다.

예제

session 1

```
csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> set transaction
isolation level REPEATABLE
READ;

Isolation level set to:
REPEATABLE READ
```

session 2

```
csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> set transaction
isolation level REPEATABLE
READ;

Isolation level set to:
REPEATABLE READ
```

```
csql> CREATE TABLE
lock_tbl(host_year INTEGER,
nation_code CHAR(3));
csql> INSERT INTO lock_tbl
VALUES (2004, 'KOR');
csql> INSERT INTO lock_tbl
VALUES (2004, 'USA');
csql> INSERT INTO lock_tbl
VALUES (2004, 'GER');
csql> INSERT INTO lock_tbl
VALUES (2008, 'GER');
csql> COMMIT;
```

```
csql> DELETE FROM lock_tbl
WHERE nation_code = 'KOR';

/* The two transactions lock
```

```
csql> DELETE FROM lock_tbl
WHERE nation_code = 'GER';
```

session 1

```
different objects
* and they do not block each-
other.
*/
```

session 2

```
csql> DELETE FROM lock_tbl
WHERE host_year=2008;

/* T1 want's to modify a row
locked by T2 and is blocked */
```

```
csql> DELETE FROM lock_tbl
WHERE host_year = 2004;

/* T2 now want to delete the
row blocked by T1
* and a deadlock is created.
*/
```

```
ERROR: Your transaction (index
1, dba@ 090205|4760)
      has been unilaterally
aborted by the system.
```

```
/* System rolled back the
transaction 1 to resolve a
deadlock */
```

```
/* T2 is unblocked and
proceeds on modifying its
rows. */
```

트랜잭션 잠금 타임아웃

CUBRID는 트랜잭션 잠금 설정이 허용될 때까지 잠금을 대기하는 시간을 설정하는 잠금 타임아웃 (lock timeout) 기능을 제공한다.

만약 설정된 잠금 타임아웃 시간 이내에 잠금이 허용되지 않으면, 잠금 타임아웃 시간이 경과된 시점에 해당 트랜잭션을 롤백시키고 에러를 출력한다. 또한, 잠금 타임아웃 시간 이내에 트랜잭션 교착

상태가 발생하면, CUBRID는 교착 상태에 있는 여러 트랜잭션 중 대기시간이 타임아웃 시간에 가까운 트랜잭션을 롤백시킨다.

잠금 타임아웃 값 설정

\$CUBRID/conf/cubrid.conf 파일 내의 시스템 파라미터 **lock_timeout** 또는 **SET TRANSACTION** 구문을 통해 응용 프로그램이 잠금을 대기하는 타임아웃 시간(초 단위)을 설정하며, 설정된 시간이 경과된 이후에는 해당 트랜잭션을 롤백시키고 에러를 출력한다. **lock_timeout** 파라미터의 기본값은 -1이며, 이는 트랜잭션 잠금이 허용되는 시점까지 무한정 대기한다는 의미이다. 따라서, 사용자는 응용 프로그램의 트랜잭션 패턴에 맞게 이 값을 변경할 수 있다. 만약, 잠금 타임아웃 값이 0으로 설정되면 잠금이 발생하는 즉시 에러 메시지가 출력될 것이다.

```
SET TRANSACTION LOCK TIMEOUT timeout_spec [ ; ]
timeout_spec:
- INFINITE
- OFF
- unsigned_integer
- variable
```

- **INFINITE** : 트랜잭션 잠금이 허용될 때까지 무한정 대기한다. 시스템 파라미터 **lock_timeout**을 -1로 설정한 것과 같다.
- **OFF** : 잠금을 대기하지 않고, 해당 트랜잭션을 롤백시킨 후 에러 메시지를 출력한다. 시스템 파라미터 **lock_timeout**을 0으로 설정한 것과 같다.
- **unsigned_integer** : 초 단위로 설정되며, 설정된 시간만큼 트랜잭션 잠금을 대기한다.
- **variable** : 변수를 지정할 수 있으며, 변수에 저장된 값만큼 트랜잭션 잠금을 대기한다.

예제 1

```
vi $CUBRID/conf/cubrid.conf
...
lock_timeout = 10s
...
```

예제 2

```
SET TRANSACTION LOCK TIMEOUT 10;
```

잠금 타임아웃 값 확인

GET TRANSACTION 문을 이용하여 현재 응용 프로그램이 설정된 잠금 타임아웃 값을 확인할 수 있고, 이 값을 변수에 저장할 수도 있다.

```
GET TRANSACTION LOCK TIMEOUT [ { INTO | TO } variable ] [ ; ]
```

예제

```
GET TRANSACTION LOCK TIMEOUT;
```

Result

```
=====
1.000000e+001
```

잠금 타임아웃 에러 메시지 확인과 조치 방법

다른 트랜잭션의 잠금이 해제되기를 대기하던 트랜잭션에 대해 잠금 타임아웃이 발생하면, 아래와 같은 에러 메시지를 출력한다.

```
Your transaction (index 2, user1@host1|9808) timed out waiting on
IX_LOCK lock on class tbl. You are waiting for
user(s) user1@host1|csql(9807), user1@host1|csql(9805) to finish.
```

- Your transaction(index 2 ...): 잠금을 대기하다가 타임아웃으로 롤백된 트랜잭션의 인덱스가 2라는 의미이다. 트랜잭션 인덱스는 클라이언트가 데이터베이스 서버에 접속하였을 때 순차적으로 할당되는 번호이다. 이는 **cubrid lockdb** 유틸리티 실행을 통해서도 확인할 수 있다.
- (... user1@host1|9808): *user1* 는 클라이언트의 로그인 아이디이고, @의 뒷 부분은 클라이언트가 수행된 호스트 이름이다. 또한 | 의 뒷 부분은 클라이언트의 프로세스 ID(PID)이다.
- IX_LOCK: 특정 행에 배타 잠금이 설정됨에 따라 상위 객체인 테이블에 의도 배타 잠금이 설정된 다. 이에 관한 상세한 설명은 **잠금 모드의 종류와 호환성** 을 참고한다.
- user1@host1|csql(9807), user1@host1|csql(9805): **IX_LOCK** 잠금을 설정하기 위해 종료되기를 기다리는 다른 트랜잭션들이다.

즉, 위의 잠금 에러 메시지는 “다른 트랜잭션들이 *participant* 테이블의 특정 행에 **X_LOCK** 을 점유하고 있으므로, *host1* 호스트에서 수행된 트랜잭션 3은 잠금이 해제되기를 기다리다가 제한 시간이 초과되었다.”로 해석할 수 있다. 만약, 에러 메시지에 명시된 트랜잭션의 잠금 정보를 확인하고자 한다면, **cubrid lockdb** 유틸리티를 통해 현재 **X_LOCK** 이 설정되어 있는 특정 행의 OID 값(예: 0|636|34)을 검색하여 현재 잠금을 점유 중인 트랜잭션 ID, 클라이언트 프로그램명 및 프로세스 ID(PID)를 확인할 수 있다. 이에 관한 상세한 설명은 **lockdb 상태 확인** 을 참고한다. CUBRID Manager에서 트랜잭션 잠금 정보를 확인할 수도 있다.

이처럼 트랜잭션의 잠금 정보를 확인한 후에는 SQL 로그를 통해 커밋되지 않은 질의문을 확인하여 트랜잭션을 정리할 수 있다. SQL 로그를 확인하는 방법은 **브로커 로그**를 참고한다.

또한, **cubrid killtran** 유틸리티를 통해 문제가 되는 트랜잭션을 강제 종료할 수 있으며, 이에 관한 상세한 설명은 **killtran** 를 참고한다.

트랜잭션 격리 수준

트랜잭션의 격리 수준은 트랜잭션이 동시에 진행 중인 다른 트랜잭션에 의해 간섭받는 정도를 의미하며, 트랜잭션 격리 수준이 높을수록 트랜잭션 간 간섭이 적으며 직렬적이고, 트랜잭션 격리 수준이 낮을수록 트랜잭션 간 간섭은 많으나 높은 동시성을 보장한다. 사용자는 적용하고자 하는 서비스의 특성에 따라 격리 수준을 적절히 설정함으로써 데이터베이스의 일관성(consistency)과 동시성(concurrency)을 조정할 수 있다.

참고

지원되는 모든 격리 수준에서 트랜잭션은 복구 가능하다. 이는 트랜잭션이 끝나기 전에는 갱신을 커밋하지 않기 때문이다.

트랜잭션 격리 수준 설정

\$CUBRID/conf/cubrid.conf의 **isolation_level** 및 **SET TRANSACTION** 문을 사용하여 트랜잭션 격리 수준을 설정할 수 있다. 기본적으로 **READ COMMITTED** 수준이 설정되어 있으며, 4~6 수준 중에서 4 수준에 해당한다(1~3 수준은 CUBRID의 이전 버전에서 사용되었으며 더 이상 사용하지 않음). 이에 관한 상세한 설명은 [데이터베이스 동시성](#)을 참고한다.

```
SET TRANSACTION ISOLATION LEVEL isolation_level_spec ;

isolation_level_spec:
    SERIALIZABLE | 6
    REPEATABLE READ | 5
    READ COMMITTED | CURSOR STABILITY | 4
```

예제 1

```
vi $CUBRID/conf/cubrid.conf
...

isolation_level = 4
...

-- or

isolation_level = "TRAN_READ_COMMITTED"
```

예제 2

```
SET TRANSACTION ISOLATION LEVEL 4;
-- or
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

CUBRID가 지원하는 격리 수준

격리 수준 이름	설명
READ COMMITTED (4)	트랜잭션 T1이 테이블 A를 조회하는 동안 다른 트랜잭션 T2는 테이블 A의 스키마를 갱신 할 수 없다. 트랜잭션 T1은 여러번 R를 조회하는 중에 트랜잭션 T2가 갱신하고 커밋한 R'를 읽기 (반복 불가능한 읽기)를 경험할 수 있다.
REPEATABLE READ (5)	트랜잭션 T1이 테이블 A를 조회하는 동안 다른 트랜잭션 T2는 테이블 A의 스키마를 갱신 할 수 없다.
SERIALIZABLE (6)	지원 예정 - 상세한 사항은 다음을 참고한다. SERIALIZABLE 격리 수준

응용 프로그램에서 트랜잭션 수행 중에 격리 수준이 변경되면, 수행 중인 트랜잭션의 남은 부분부터 변경된 격리 수준이 적용된다. 이처럼 설정된 격리 수준이 하나의 트랜잭션 전체에 적용되는 것이 아니라 트랜잭션 중간에 변경되어 적용될 수 있기 때문에, 트랜잭션 격리 수준은 트랜잭션 시작 시점 (커밋, 롤백, 또는 시스템 재시작 이후)에 변경하는 것이 바람직하다.

트랜잭션 격리 수준 값 확인

GET TRANSACTION ISOLATION LEVEL 문을 이용하여 현재 클라이언트에 설정된 격리 수준 값을 출력하거나 *variable* 에 할당할 수 있다. 아래는 격리 수준을 확인하기 위한 구문이다.

```
GET TRANSACTION ISOLATION LEVEL [ { INTO | TO } variable ] [ ; ]
```

```
GET TRANSACTION ISOLATION LEVEL;
```

Result

=====

READ COMMITTED

READ COMMITTED 격리 수준

비교적 낮은 격리 수준(4)으로서 더티 읽기는 발생하지 않지만, 반복 불가능한 읽기와 유령 읽기는 발생할 수 있다. 즉, 트랜잭션 $T1$ 이 하나의 객체를 반복하여 조회하는 동안 다른 트랜잭션 $T2$ 에서의 삽입 또는 갱신이 허용되어, 트랜잭션 $T1$ 이 다른 값을 읽을 수 있다는 의미이다.

다음과 같은 규칙이 적용된다.

- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 삽입되는 레코드를 읽거나 수정할 수 없다. 대신 레코드가 무시된다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 갱신하는 레코드를 읽을 수 있으며, 레코드의 마지막 커밋된 버전을 확인한다(커밋되지 않은 버전은 볼 수 없음).
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 갱신 중인 레코드를 수정할 수 없다. $T1$ 은 $T2$ 가 커밋되기를 기다린 후 커밋이 되면 레코드 값을 다시 평가한다. 재평가시 적합하면 $T1$ 은 레코드를 수정하고 적합하지 않으면 무시한다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 레코드를 수정할 수 있다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 테이블에 레코드를 갱신/삽입할 수 있다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 테이블의 스키마를 변경할 수 없다.
- 트랜잭션 $T1$ 이 수행된 각각의 질의문에 새로운 스냅샷을 생성하므로 유령 읽기 또는 반복할 수 없는 읽기는 발생할 수 있다.

이 격리 수준은 배타 잠금에 대해 MVCC 잠금 프로토콜을 따른다. 행에 공유 잠금은 필요하지 않지만 테이블에 대한 의도 잠금은 스키마에 대한 반복 가능한 읽기를 보장하기 위하여 트랜잭션이 종료될 때 해제된다.

예:

다음 예제는 트랜잭션 격리 수준이 **READ COMMITTED** 인 경우, 한 트랜잭션이 객체 읽기를 수행하는 동안 다른 트랜잭션이 레코드를 추가하거나 갱신할 수 있기때문에 유령 또는 반복 불가능한 읽기가 발생할 수 있음을 보여주고 테이블 스키마 갱신에 대해서는 반복 가능한 읽기가 보장된다는 것을 보여준다.

session 1

session 2

```
csql> ;autocommit off
```

```
AUTOCOMMIT IS OFF
```

```
csql> SET TRANSACTION
ISOLATION LEVEL READ COMMITTED;
```

```
Isolation level set to:
READ COMMITTED
```

```
csql> ;autocommit off
```

```
AUTOCOMMIT IS OFF
```

```
csql> SET TRANSACTION
ISOLATION LEVEL READ
COMMITTED;
```

```
Isolation level set to:
READ COMMITTED
```

```
csql> CREATE TABLE
isol4_tbl(host_year integer,
nation_code char(3));
```

```
csql> INSERT INTO isol4_tbl
VALUES (2008, 'AUS');
```

```
csql> COMMIT;
```

```
csql> SELECT * FROM isol4_tbl;
```

```
      host_year  nation_code
=====
=====
      2008      'AUS'
```

```
csql> INSERT INTO isol4_tbl
VALUES (2004, 'AUS');
```

```
csql> INSERT INTO isol4_tbl
VALUES (2000, 'NED');
```

```
csql> COMMIT;
```

session 1

session 2

```
/* phantom read occurs
because tran 1 committed */
csql> SELECT * FROM isol4_tbl;
```

host_year	nation_code
=====	
=====	
2008	'AUS'
2004	'AUS'
2000	'NED'

```
csql> UPDATE isol4_tbl
csql> SET nation_code = 'KOR'
csql> WHERE host_year = 2008;
csql> COMMIT;
```

```
/* unrepeatable read occurs
because tran 1 committed */
csql> SELECT * FROM isol4_tbl;
```

host_year	nation_code
=====	
=====	
2008	'KOR'
2004	'AUS'
2000	'NED'

```
csql> ALTER TABLE isol4_tbl
ADD COLUMN gold INT;
```

```
/* unable to alter the table
schema until tran 2 committed
*/
```

session 1

session 2

```
/* repeatable read is ensured
while
  * tran_1 is altering table
schema
*/
```

```
csql> SELECT * FROM isol4_tbl;
```

```
      host_year  nation_code
=====
=====
      2008      'KOR'
      2004      'AUS'
      2000      'NED'
```

```
csql> COMMIT;
```

```
csql> SELECT * FROM isol4_tbl;
```

```
/* unable to access the table
until tran_1 committed */
```

```
csql> COMMIT;
```

```
host_year  nation_code  gold
=====
=====
      2008      'KOR'           NULL
      2004      'AUS'           NULL
      2000      'NED'           NULL
```

READ COMMITTED UPDATE RE-EVALUATION

READ COMMITTED 격리 수준은 동시에 발생하는 행 갱신에 대해 높은 격리 수준과 다르게 처리한다. 높은 격리 수준에서는 동시 트랜잭션 $T1$ 에서 이미 갱신한 행을 $T2$ 가 수정하려고 시도하면 $T1$ 이 커밋 또는 롤백할 때까지 차단되며, $T1$ 이 커밋되면 $T2$ 가 질의 수행을 중단하고 직렬화 오류가 표시된다. **READ COMMITTED** 격리 수준에서는 $T1$ 이 커밋되면 $T2$ 가 질의 수행을 바로 중단하지 않고 최신 버전을 재 평가한다. 이전 버전을 선택하는 데 사용된 조건값이 최신 버전에 대해서도 계속 적용되면 $T2$ 는 최신 버전을 수정한다. 조건값이 더 이상 만족하지 않으면 $T2$ 는 해당 레코드의 수정을 무시한다.

예:

session 1

```
csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> SET TRANSACTION
ISOLATION LEVEL 4;

Isolation level set to:
READ COMMITTED
```

session 2

```
csql> ;autocommit off

AUTOCOMMIT IS OFF

csql> SET TRANSACTION
ISOLATION LEVEL 4;

Isolation level set to:
READ COMMITTED
```

```
csql> CREATE TABLE
isol4_tbl(host_year integer,
nation_code char(3));
csql> INSERT INTO isol4_tbl
VALUES (2000, 'KOR');
csql> INSERT INTO isol4_tbl
VALUES (2004, 'USA');
csql> INSERT INTO isol4_tbl
VALUES (2004, 'GER');
csql> INSERT INTO isol4_tbl
VALUES (2008, 'GER');
csql> COMMIT;
```

```
csql> UPDATE isol4_tbl
csql> SET host_year =
host_year - 4
csql> WHERE nation_code =
```

session 1

```
'GER';

/* T1 locks and modifies
(2004, 'GER') to (2000, 'GER')
*/
/* T1 locks and modifies
(2008, 'GER') to (2004, 'GER')
*/
```

session 2

```
csql> UPDATE isol4_tbl
csql> SET host_year =
host_year + 4
csql> WHERE host_year >= 2004;
```

```
/* T2 snapshot will try to
modify three records:
* (2004, 'USA'), (2004,
'GER'), (2008, 'GER')
*
* T2 locks and modifies
(2004, 'USA') to (2008, 'USA')
* T2 is blocked on lock on
(2004, 'GER').
*/
```

```
csql> COMMIT;
```

```
/* T1 releases locks on
modified rows. */
```

```
/* T2 is unblocked and will
do the next steps:
*
* T2 finds (2004, 'GER')
has a new version (2000,
'GER')
* that doesn't satisfy
predicate anymore.
```


session 1

session 2

```

*   T2 releases the lock on
object and ignores it.
*
*   T2 finds (2008, 'GER')
has a new version (2004,
'GER')
*   that still satisfies the
predicate.
*   T2 keeps the lock and
changes row to (2008, 'GER')
*/

```

```

csql> SELECT * FROM isol4_tbl;

      host_year  nation_code
=====
=====
          2000      'KOR'
          2000      'GER'
          2008      'USA'
          2008      'GER'

```

REPEATABLE READ 격리 수준

REPEATABLE READ 격리 수준(5)은 **snapshot isolation** 때문에 더티 읽기, 반복 불가능한 읽기 및 유령 읽기가 발생하지 않는다. 하지만 완벽하게 **serializable** 하지는 않으므로, 동시에 실행 중인 다른 트랜잭션이 없는 트랜잭션 수행이라고 말할 수 없으며, **serializable snapshot isolation** 수준이 허용하지 않는 스큐 쓰기(write skew)와 같은 복잡한 예외가 발생할 수 있다.

스큐 쓰기는 두 개의 트랜잭션이 동시에 겹치는 데이터 셋을 읽고 겹친 각각 다른 영역의 갱신을 수행하여 다른 사용자가 수행한 갱신을 확인할 수 없는 현상이다. 직렬화 시스템에서는 한 트랜잭션이 먼저 발생하고 두 번째 트랜잭션이 첫 번째 트랜잭션의 갱신을 확인하기 때문에 이러한 현상이 발생하지 않는다.

다음과 같은 규칙이 적용된다.

- 트랜잭션 *T1* 은 다른 트랜잭션 *T2* 에서 삽입되는 레코드를 읽거나 수정할 수 없으며 대신 해당 레코드를 무시한다.

- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 갱신 중인 레코드를 읽을 수 있으며, 레코드의 마지막 커밋된 버전을 확인한다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 갱신 중인 레코드를 수정할 수 없다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 레코드를 수정할 수 있다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 테이블에 레코드를 갱신/삽입할 수 있다.
- 트랜잭션 $T1$ 은 다른 트랜잭션 $T2$ 에서 조회 중인 테이블의 스키마를 변경할 수 없다.
- 트랜잭션 $T1$ 은 트랜잭션의 수행 동안에 유효한 고유 스냅샷을 생성한다.

예제

다음 예제는 **snapshot isolation** 으로 인해 반복 불가능한 읽기 및 유령 읽기가 발생하지 않는 것을 보여준다. 그러나 격리 수준이 **serializable** 되어 있지 않기 때문에 스큐 쓰기는 발생할 수 있다.

session 1

```
csql> ;autocommit off
AUTOCOMMIT IS OFF

csql> SET TRANSACTION
ISOLATION LEVEL 5;

Isolation level set to:
REPEATABLE READ
```

session 2

```
csql> ;autocommit off
AUTOCOMMIT IS OFF

csql> SET TRANSACTION
ISOLATION LEVEL 5;

Isolation level set to:
REPEATABLE READ
```

session 1

session 2

```

csql> CREATE TABLE
isol5_tbl(host_year integer,
nation_code char(3));
csql> CREATE UNIQUE INDEX
isol5_u_idx
      on
isol5_tbl(nation_code,
host_year);
csql> INSERT INTO isol5_tbl
VALUES (2008, 'AUS');
csql> INSERT INTO isol5_tbl
VALUES (2004, 'AUS');

csql> COMMIT;

```

```

csql> SELECT * FROM isol5_tbl
WHERE nation_code='AUS';

```

host_year	nation_code
2004	'AUS'
2008	'AUS'

```

csql> INSERT INTO isol5_tbl
VALUES (2004, 'KOR');
csql> INSERT INTO isol5_tbl
VALUES (2000, 'AUS');

/* able to insert new rows */
csql> COMMIT;

```

```

csql> SELECT * FROM isol5_tbl
WHERE nation_code='AUS';

```

```

/* phantom read cannot occur

```

session 1

session 2

```
due to snapshot isolation */
```

```

      host_year  nation_code
=====
=====
      2004      'AUS'
      2008      'AUS'

```

```

csql> UPDATE isol5_tbl
csql> SET host_year = 2012
csql> WHERE nation_code =
'AUS' and
csql> host_year=2008;

/* able to update rows viewed
by T2 */
csql> COMMIT;

```

```

csql> SELECT * FROM isol5_tbl
WHERE nation_code = 'AUS';

/* non-repeatable read cannot
occur due to snapshot
isolation */

```

```

      host_year  nation_code
=====
=====
      2004      'AUS'
      2008      'AUS'

```

```
csql> COMMIT;
```

```

csql> SELECT * FROM isol5_tbl
WHERE host_year >= 2004;

```

```

csql> SELECT * FROM isol5_tbl
WHERE nation_code = 'AUS';

```

session 1

```

      host_year  nation_code
=====
=====
      2004      'AUS'
      2004      'KOR'
      2012      'AUS'

csql> UPDATE isol5_tbl
csql> SET nation_code = 'USA'
csql> WHERE nation_code =
'AUS' and
csql> host_year = 2004;

csql> COMMIT;

```

session 2

```

      host_year  nation_code
=====
=====
      2000      'AUS'
      2004      'AUS'
      2012      'AUS'

csql> UPDATE isol5_tbl
csql> SET nation_code = 'NED'
csql> WHERE nation_code =
'AUS' and
csql> host_year = 2012;

csql> COMMIT;

```

```

/* T1 and T2 first have selected each 3 throws and rows (2004,
'AUS'), (2012, 'AUS') overlapped.
 * Then T1 modified (2004, 'AUS'), while T2 modified (2012, 'AUS'),
without blocking each other.
 * In a serial execution, the result of select query for T1 or T2,
whichever executes last, would be different.
*/

```

```

csql> SELECT * FROM isol5_tbl
WHERE nation_code = 'AUS';

      host_year  nation_code
=====
=====
      2000      'AUS'

```

```

csql> ALTER TABLE isol5_tbl
ADD COLUMN gold INT;

/* unable to alter the table

```

session 1

```
schema until tran 2 committed
*/
```

session 2

```
/* repeatable read is ensured
while tran_1 is altering
* table schema
*/
```

```
csql> SELECT * FROM isol5_tbl
WHERE nation_code = 'AUS';
```

```
      host_year  nation_code
=====
=====
      2000      'AUS'
```

```
csql> COMMIT;
```

```
csql> SELECT * FROM isol5_tbl
WHERE nation_code = 'AUS';
```

```
/* unable to access the table
until tran_1 committed */
```

```
csql> COMMIT;
```

session 1

session 2

```

host_year  nation_code  gold
=====
=====
2000      'AUS'          NULL

```

SERIALIZABLE 격리 수준

CUBRID 10.0의 **SERIALIZABLE** 격리 수준은 **REPEATABLE READ** 격리 수준과 동일하다. **REPEATABLE READ 격리 수준** 섹션에 설명된 대로, SNAPSHOT 격리 수준으로 인해 반복 불가능한 읽기 및 유령 읽기 현상은 발생하지 않지만 스큐 쓰기 현상은 여전히 발생할 수 있다. 스큐 쓰기를 방지하기 위해서 읽기에 대해 인덱스 키 잠금을 사용하거나, 격리 충돌을 발생할 수 있는 트랜잭션을 중단시키는 등의 다양한 **SERIALIZABLE SNAPSHOT ISOLATION** 방법이 있다. 향후 CUBRID 버전에서는 이런 방법 중 하나를 제공될 예정이다.

즉, 기존 버전과의 호환성을 위해 키워드는 제거되지 않았지만 **SERIALIZABLE**은 **REPEATABLE READ**와 유사하다.

CUBRID에서 더티 레코드를 다루는 방법

CUBRID는 다양한 상황에서 클라이언트의 버퍼에 존재하는 더티 데이터(또는 더티 레코드)를 데이터베이스 서버로 내려쓰기(flush)한다. 아래에 명시된 상황이 아닌 경우에도 내려쓰기가 발생할 수 있다.

- 트랜잭션 커밋이 수행될 때 더티 데이터는 서버로 내려쓰기된다.
- 클라이언트의 버퍼에 적재된 데이터가 많은 경우, 일부 더티 데이터는 서버로 내려쓰기된다.
- 테이블 A의 더티 데이터는 테이블 A의 스키마가 갱신될 때 서버로 내려쓰기된다.
- 테이블 A의 더티 데이터는 테이블 A가 조회(**SELECT**)될 때 서버로 내려쓰기된다.
- 더티 데이터의 일부는 서버 함수가 호출될 때 내려쓰기될 수 있다.

트랜잭션 종료와 복구

CUBRID에서 복구 프로세스를 사용하면 소프트웨어 또는 하드웨어에 오류가 발생하더라도 데이터베이스에는 영향을 미치지 않도록 할 수 있다. CUBRID에서 모든 읽기와 갱신 명령문은 원자성(atomic)을 보장한다. 이것은 명령문들이 커밋되어 데이터베이스가 갱신되거나, 커밋되지 않아 갱신이 무효

화되어야 함을 의미한다. 원자성의 개념은 트랜잭션을 구성하는 연산의 집합으로 확장된다. 트랜잭션은 커밋을 성공하여 모든 영향이 데이터베이스에 영구화 되거나 아니면 롤백되어 트랜잭션의 모든 영향이 제거되어야 한다. 트랜잭션의 원자성을 보장하기 위해서 CUBRID는 모든 트랜잭션의 갱신이 디스크에 쓰여지지 않은 채 오류가 발생할 때마다 커밋된 트랜잭션의 영향을 다시 적용시킨다. 또한 CUBRID는 사이트가 실패(몇몇 트랜잭션이 커밋되지 못했거나 응용 프로그램이 트랜잭션 취소를 요청했을 수 있다)할 때마다 데이터베이스에서 부분적으로 커밋된 트랜잭션의 영향을 제거한다. 이러한 복구 기능은 응용 프로그램이 시스템 오류에 따라 어떻게 데이터베이스를 일관성 있는 상태로 되돌릴 지에 대한 부담을 덜어준다. CUBRID에서 사용되는 복구 기능은 언두/리두 로깅 기법을 기반으로 한다.

CUBRID는 하드웨어와 소프트웨어 오류가 발생하는 동안 트랜잭션의 원자성을 유지하기 위해서 자동 복구 기법을 제공한다. CUBRID의 복구 기능은 응용 프로그램 또는 컴퓨터 시스템의 오류가 발생 하더라도 데이터베이스를 항상 일관된 상태로 되돌려놓기 때문에 사용자는 복구에 대한 책임을 가질 필요가 없다. 이것을 위해 CUBRID는 시스템이나 응용 프로그램의 실패 또는 사용자의 명시적 요청에 따라 커밋된 트랜잭션의 일부를 자동적으로 롤백한다. 예를 들어 **COMMIT WORK** 문이 수행되는 동안 발생한 시스템 오류는 트랜잭션이 아직 커밋되지 않았다면(사용자의 연산이 커밋되었다는 확인을 받지 못한다) 중단해야 할 것이다. 자동 중단은 커밋되지 않은 갱신을 취소함으로써 데이터베이스에 원하지 않은 변경을 야기하는 오류를 방지한다.

데이터베이스 재구동

CUBRID는 시스템과 매체(디스크)에 오류가 발생했을 때 커밋되었거나 커밋되지 않은 트랜잭션을 복구하기 위해 로그 볼륨/파일과 데이터베이스 백업을 이용한다. 로그는 사용자가 지정한 롤백을 지원하는데도 사용된다. 로그는 CUBRID가 생성한 순차적인 파일의 모음으로 구성된다. 가장 최근의 로그를 활성 로그(active log)라고 부르며, 나머지 로그를 보관 로그(archive log)라고 부른다. 로그 파일은 활성 로그와 보관 로그 전체를 가리키는데 사용된다.

데이터베이스에 대한 모든 갱신은 로그에 기록된다. 실제로 갱신에 대한 2개의 복사본이 기록되는데, 첫 번째 복사본은 before 이미지(UNDO log)라고 불리며 사용자가 명시한 **ROLLBACK WORK** 문이 수행되는 동안이나 매체 또는 시스템에 오류가 발생했을 때 데이터를 복원하는데 사용된다. 두 번째 복사본은 after 이미지(REDO log)인데 매체 또는 시스템에 오류가 발생했을 때 갱신을 다시 적용시키는데 사용된다.

CUBRID는 활성 로그가 꽉 차면 보관 로그로 복사하여 디스크에 보존한다. 보관 로그는 시스템 장애가 발생했을 때 데이터베이스 복구를 위해 필요하다.

정상적인 종료 또는 오류

데이터베이스가 정상적인 종료나 장비의 오류로 다시 시작되면 CUBRID는 자동적으로 데이터베이스를 복구한다. 복구 프로세스는 데이터베이스에 빠져있는 커밋된 변화를 다시 적용하고 데이터베이스에 저장되어 있는 커밋되지 않은 변경을 제거한다. 데이터베이스 시스템의 일반적인 연산은 복구가 끝나고 난 후 재개된다. 이러한 복구 프로세스는 어떠한 보관 로그나 데이터베이스 백업도 사용하지 않는다.

데이터베이스는 **cubrid server** 유틸리티를 이용해 재구동할 수 있다.

매체 오류

매체에 오류가 발생 한 후에 데이터베이스를 다시 구동시키는 데는 사용자의 개입이 다소 필요하다. 첫 번째 단계는 좋은 상태로 알려진 데이터베이스의 백업을 설치하여 데이터베이스를 복원하는 것이다. CUBRID에서 가장 최근의 로그 파일(마지막 백업 이후의 것)을 설치하는 것을 필요로 한다. 이 특정 로그(보관, 활성화)는 데이터베이스의 백업 복사본에 적용된다. 복원이 커밋된 후 데이터베이스는 일반적인 종료의 경우와 마찬가지로 재구동할 수 있다.

참고

데이터베이스의 정보를 잃어버릴 가능성을 줄이기 위해서 보관 로그가 디스크에서 삭제되기 전에 보관 로그의 스냅샷을 만들고 이를 백업 장치에 보관할 것을 권장한다. DBA는 **cubrid backupdb**, **cubrid restoredb** 유틸리티를 사용하여 데이터베이스를 백업하고 복원할 수 있다. 이 유틸리티에 대한 상세한 내용을 보려면 **backupdb**를 참고한다.

트리거

CREATE TRIGGER

트리거 정의를 위한 가이드라인

트리거 정의로 다양하고 강력한 기능을 만들 수 있다. 트리거를 생성하기 전에 다음과 같은 정의 사항을 고려해야 한다.

- 트리거의 조건 영역 표현식이 데이터베이스에 예측할 수 없는 결과(side effect)를 가져오지는 않는가?

SQL 문을 예측이 가능한 범위 내에서 사용해야 한다.

- 트리거의 실행 영역이 트리거의 이벤트 대상으로 주어진 테이블을 변경하지는 않는가?

이러한 유형의 설계가 트리거의 정의에서 사용이 금지되어 있지는 않지만, 무한 루프로 빠지는 트리거가 만들어질 수 있어 주의해서 사용해야 한다. 트리거 실행 영역이 이벤트 대상 테이블을 수정할 때, 같은 트리거가 다시 불려질 수 있다. 또한 **WHERE** 절을 포함하는 문장에 의해 트리거가 발생하면, 해당 트리거는 **WHERE** 절에 의해 수행되는 테이블에는 일반적으로 부작용이 없다.

- 트리거가 불필요한 오버헤드를 만들어 내고 있지는 않는가?

원하는 동작이 소스 내에서 조금 더 효과적으로 표현될 수 있다면 직접 소스 내에서 구현하도록 한다.

· 트리거가 재귀적으로 실행되고 있지는 않는가?

트리거의 실행 영역이 트리거를 호출하고 이 트리거가 다시 처음 트리거를 호출하면 재귀 루프(recursive loop)가 데이터베이스에 만들어진다. 재귀 루프가 만들어지면, 트리거가 정확히 수행되지 않거나 진행 중인 루프를 단절하기 위해 현재 세션을 강제로 종료해야 할 수도 있다.

· 트리거의 정의는 유일한가?

동일한 테이블에서 정의된 트리거나, 동일한 실행 영역에서 시작된 트리거는 복구할 수 없는 에러의 원인이 된다. 동일한 테이블에 있는 트리거는 다른 트리거 이벤트를 가져야 한다. 또한, 트리거의 우선순위는 명시적으로 정의되어 있거나 모호하지 않아야 한다.

트리거 정의 구문

CREATE TRIGGER 문을 사용하여 새로운 트리거를 생성하고, 트리거 대상, 실행 조건과 수행할 내용을 정의할 수 있다. 트리거는 데이터베이스 객체로서, 특정 이벤트가 대상 테이블에 대해 발생하면 정의된 동작을 수행한다.

```
CREATE TRIGGER [schema_name.]trigger_name
[ STATUS { ACTIVE | INACTIVE } ]
[ PRIORITY key ]
<event_time> <event_type> [<event_target>]
[ IF condition ]
EXECUTE [ AFTER | DEFERRED ] action
[COMMENT 'trigger_comment'];

<event_time> ::=
    BEFORE |
    AFTER |
    DEFERRED

<event_type> ::=
    INSERT |
    STATEMENT INSERT |
    UPDATE |
    STATEMENT UPDATE |
    DELETE |
    STATEMENT DELETE |
    ROLLBACK |
    COMMIT

<event_target> ::=
    ON [schema_name.]table_name |
    ON [schema_name.]table_name [ (column_name) ]

<condition> ::=
    expression

<action> ::=
```

```
REJECT |
INVALIDATE TRANSACTION |
PRINT message_string |
INSERT statement |
UPDATE statement |
DELETE statement
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 정의하려는 트리거의 이름을 지정한다.
- [**STATUS** { **ACTIVE** | **INACTIVE** }]: 트리거의 상태를 정의한다(정의하지 않을 경우 기본값은 **ACTIVE**).
 - **ACTIVE** 상태인 경우 관련 이벤트가 발생할 때마다 트리거를 실행한다.
 - **INACTIVE** 상태인 경우 관련 이벤트가 발생하여도 트리거를 실행하지 않는다. 트리거의 활성 여부는 변경할 수 있다. 자세한 내용은 **ALTER TRIGGER** 을 참조한다.
- [**PRIORITY** *key*]: 하나의 이벤트에 대해서 다수의 트리거가 불려질 경우 실행되는 우선순위를 부여한다. *key* 값은 반드시 음수가 아닌 부동 소수점 값이어야 한다. 우선순위를 정의하지 않을 경우 가장 낮은 우선순위인 0을 할당한다. 같은 우선순위를 가지는 트리거는 임의의 순서로 실행된다. 트리거의 우선순위는 변경할 수 있다. 자세한 내용은 **ALTER TRIGGER** 을 참조한다.
- *<event_time>*: 트리거의 조건 영역과 실행 영역이 실행되는 시점을 지정하며 **BEFORE**, **AFTER**, **DEFERRED** 가 있다. 자세한 내용은 **이벤트 시점** 을 참조한다.
- *<event_type>*: 트리거 타입은 사용자 트리거와 테이블 트리거로 나뉜다. 자세한 내용은 **트리거 이벤트 타입** 을 참조한다.
- *<event_target>*: 이벤트 대상은 트리거가 호출되기 위한 대상을 지정할 때 쓰인다. 자세한 내용은 **트리거 이벤트 대상** 을 참조한다.
- *<condition>*: 트리거의 조건영역을 지정한다. 자세한 내용은 **트리거 조건 영역** 을 참조한다.
- *<action>*: 트리거의 실행영역을 지정한다. 자세한 내용은 **트리거 실행 영역** 을 참조한다.
- *trigger_comment*: 트리거의 커멘트를 지정한다.

참고

- **DBA** 와 **DBA** 멤버는 다른 스키마에 트리거를 생성할 수 있다. 사용자가 **DBA** 도 아니고 **DBA** 멤버도 아니면 해당 사용자의 스키마에서만 트리거를 생성할 수 있다.

다음은 *participant* 테이블의 레코드를 갱신할 때 획득 메달의 개수가 0보다 작을 경우 갱신을 거절하는 트리거를 생성하는 예제이다. 2004년도 올림픽에 한국이 획득한 금메달의 개수를 음수로 갱신할 경우 갱신이 거절되는 것을 알 수 있다.

```
CREATE TRIGGER medal_trigger
BEFORE UPDATE ON participant
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;

UPDATE participant SET gold = -5 WHERE nation_code = 'KOR'
AND host_year = 2004;
```

```
ERROR: The operation has been rejected by trigger "medal_trigger".
```

이벤트 시점

트리거의 조건 영역과 실행 영역이 실행되는 시점을 지정한다. 이벤트 시점의 종류에는 **BEFORE**, **AFTER**, **DEFERRED** 가 있다.

- **BEFORE**: 이벤트가 처리되기 전에 조건을 검사한다.
- **AFTER**: 이벤트가 처리된 후에 조건을 검사한다.
- **DEFERRED**: 이벤트에 대한 트랜잭션의 끝에서 조건을 검사한다. **DEFERRED** 로 지정할 경우 이벤트 타입에 **COMMIT** 이나 **ROLLBACK** 을 사용할 수 없다.

트리거 타입

사용자 트리거(User Trigger)

- 데이터베이스의 특정 사용자와 관련된 트리거를 사용자 트리거(user trigger)라고 한다.
- 사용자 트리거는 이벤트 대상이 없으며 트리거의 소유자(트리거를 생성한 사용자)에 의해서만 실행된다.
- 사용자 트리거를 정의하는 이벤트 타입은 **COMMIT** 과 **ROLLBACK** 이 있다.

테이블 트리거(Table Trigger)

- 특정 테이블을 이벤트 대상으로 가지는 트리거를 테이블 트리거(클래스 트리거)라 한다.
- 테이블 트리거는 대상 테이블에 **SELECT** 권한을 가지는 모든 사용자가 볼 수 있다.
- 테이블 트리거를 정의하는 이벤트 타입은 인스턴스 이벤트와 문장 이벤트가 있다.

트리거 이벤트 타입

- 인스턴스 이벤트(instance event) : 인스턴스 이벤트는 이벤트 연산의 단위가 인스턴스(레코드)인 이벤트 타입을 말한다. 인스턴스 이벤트의 종류는 다음과 같다.
 - **INSERT**
 - **UPDATE**
 - **DELETE**
- 문장 이벤트(statement event): 이벤트 타입을 문장 이벤트로 정의하면 주어진 문장(이벤트)에 의해 영향을 받는 객체(인스턴스)가 많더라도, 트리거는 문장이 시작할 때 한 번만 불려지게 된다. 문장 이벤트의 종류는 다음과 같다.
 - **STATEMENT INSERT**
 - **STATEMENT UPDATE**
 - **STATEMENT DELETE**
- 기타 이벤트: **COMMIT** 과 **ROLLBACK** 은 개별적인 인스턴스에는 적용할 수 없다.
 - **COMMIT**
 - **ROLLBACK**

다음은 인스턴스 이벤트를 사용하는 예제이다. *example* 트리거는 데이터베이스 갱신에 의해 영향을 받는 각각의 인스턴스에 대해서 한번씩 불려진다. 예를 들어, *history* 테이블의 다섯 개 인스턴스의 *score* 를 변경했다면, 이 트리거는 다섯 번 불려진다.

```
CREATE TABLE update_logs(event_code INTEGER, score VARCHAR(10), dt
DATETIME);

CREATE TRIGGER example
BEFORE UPDATE ON history(score)
EXECUTE INSERT INTO update_logs VALUES (obj.event_code, obj.score,
SYSDATETIME);
```

만약 *score* 칼럼의 첫 번째 인스턴스가 갱신되기 전에 트리거가 한 번만 호출되게 하려면, 아래의 예와 같이 **STATEMENT UPDATE** 형식을 사용한다.

다음은 문장 이벤트를 사용하는 예제이다. 문장 이벤트를 지정하면 갱신의 영향을 받는 인스턴스가 많더라도, 첫 번째 인스턴스가 갱신되기 전에 트리거가 한 번만 불려지게 된다.

```
CREATE TRIGGER example
BEFORE STATEMENT UPDATE ON history(score)
EXECUTE PRINT 'There was an update on history table';
```

참고

- 이벤트 타입으로 인스턴스 이벤트와 문장 이벤트를 지정할 경우에는 반드시 이벤트 대상을 명시해야 한다.
- **COMMIT** 과 **ROLLBACK** 은 이벤트 대상을 가질 수 없다.

트리거 이벤트 대상

이벤트 대상은 트리거가 호출되기 위한 대상을 지정할 때 쓰인다. 트리거 이벤트의 대상은 테이블명 혹은 테이블명과 칼럼명으로 지정할 수 있으며 칼럼명을 지정하면 해당 칼럼이 이벤트의 영향을 받을 때에만 트리거가 불려진다. 만약 칼럼을 지정하지 않으면 지정된 테이블 내에 어떤 칼럼이 영향을 받더라도 트리거가 호출된다. 오직 **UPDATE**, **STATEMENT UPDATE** 이벤트만이 이벤트 대상에 칼럼을 지정할 수 있다.

다음은 *example* 트리거의 이벤트 대상을 *history* 테이블의 *score* 칼럼으로 지정한 예제이다.

```
CREATE TABLE update_logs(event_code INTEGER, score VARCHAR(10), dt
DATETIME);

CREATE TRIGGER example
BEFORE UPDATE ON history(score)
EXECUTE INSERT INTO update_logs VALUES (obj.event_code, obj.score,
SYSDATETIME);
```

이벤트 타입과 대상 조합

트리거를 호출하는 데이터베이스 이벤트는 트리거 이벤트 타입과 트리거 정의 내의 이벤트 대상에 의해 식별된다. 다음은 트리거 이벤트 타입과 대상 조합, 트리거 이벤트가 나타내는 CUBRID 데이터베이스 이벤트의 활동을 표로 정리한 것이다.

이벤트 타입	이벤트 대상	대응되는 데이터베이스 활동
UPDATE	테이블	테이블에 UPDATE 문이 실행되었을 때 트리거가 호출된다.
INSERT	테이블	테이블에 INSERT 문이 실행되었을 때 트리거가 호출된다.
DELETE	테이블	테이블에 DELETE 문이 실행되었을 때 트리거가 호출된다.
COMMIT	없음	데이터베이스 트랜잭션이 커밋되었을 때 트리거가 호출된다.
ROLLBACK	없음	데이터베이스의 트랜잭션이 롤백되었을 때 트리거가 호출된다.

트리거 조건 영역

트리거를 정의할 때 조건 영역을 정의하여 트리거의 수행 영역에 대한 수행 여부를 결정한다.

- 트리거 조건 영역이 기술된다면, 참 또는 거짓을 평가할 수 있는 단독적인 복합 표현식으로 쓰여질 수 있다. 이 경우에 표현식은 **SELECT** 문의 **WHERE** 절에 허용되는 산술 연산자와 논리 연산자를 포함할 수 있다. 조건 영역이 참이면, 트리거 실행 영역이 수행되고, 거짓이면 실행되지 않는다.
- 트리거의 조건 영역을 생략하면 조건 없는 트리거(unconditional trigger)가 되며 트리거가 호출될 때 항상 트리거의 실행 영역이 수행된다.

다음은 조건 영역 내의 표현식에 상관명을 이용한 예제이다. 이벤트 타입이 **INSERT**, **UPDATE**, **DELETE** 인 경우에, 조건 영역 내의 표현식은 특정 칼럼 값에 접근하기 위하여 상관명 **obj**, **new**, **old** 를 사용할 수 있다. 예제에서 *example* 트리거는 칼럼의 새로운 값을 이용해서 조건 영역을 검사하기 위해 트리거 조건 영역에 **new** 를 칼럼 이름 앞에 사용하였다.

```
CREATE TRIGGER example
BEFORE UPDATE ON participant
```

```
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;
```

다음은 조건 영역 내의 표현식에 **SELECT** 문을 사용한 예제이다. 예제의 트리거는 집계함수 **COUNT** (*)를 사용하는 **SELECT** 문을 사용하여 그 값과 상수를 비교한다. **SELECT** 문은 반드시 괄호로 싸여 있어야 하고, 표현식의 마지막에 위치해야 한다.

```
CREATE TRIGGER example
BEFORE INSERT ON participant
IF 1000 > (SELECT COUNT(*) FROM participant)
EXECUTE REJECT;
```

참고

트리거 조건 영역에 주어진 표현식은 조건 영역이 수행되는 동안에 메시드가 호출되면 데이터베이스에 부작용을 초래할 수 있다. 트리거 조건 영역은 데이터베이스에 생각지 못한 부작용이 발생하지 않도록 구성해야 한다.

상관명(correlation name)

트리거를 정의할 때 상관명을 사용하여 대상 테이블의 칼럼 값에 접근할 수 있다. 상관명은 실제로 트리거를 호출하는 데이터베이스 연산에 의해 영향을 받는 인스턴스를 나타낸다. 상관명은 트리거의 조건 영역이나 실행 영역에도 기술할 수 있다.

상관명의 종류에는 **new**, **old**, **obj** 가 있으며 이러한 상관명은 인스턴스 트리거에서 **INSERT**, **UPDATE**, **DELETE** 의 이벤트 타입을 가지고 있는 트리거에서만 사용할 수 있다.

상관명의 사용은 아래 표와 같이 트리거 조건 영역에 정의된 이벤트 시점에 의해 더욱 제한된다.

	BEFORE	AFTER or DEFERRED
INSERT	new	obj
UPDATE	obj	obj
	new	old (AFTER)
DELETE	obj	N/A

상관명	대표 속성 값
obj	인스턴스의 현재 속성 값을 나타낸다. 인스턴스가 갱신되거나 삭제되기 전에 속성값에 접근하기 위해서 사용한다. 그리고 인스턴스가 갱신되거나 삽입된 후에 속성 값에 접근하기 위해 사용한다.
new	삽입이나 갱신 연산에 의해 제시되는 속성값을 나타낸다. 새로운 값은 인스턴스가 실제로 삽입되거나 갱신되기 전에만 접근할 수 있다.
old	갱신 연산의 완료 전에 존재하던 속성값을 나타낸다. 이 값은 트리거가 수행되는 동안만 유지된다. 트리거가 종료되면 old 값은 잃어버리게 된다.

트리거 실행 영역

트리거 실행 영역은 트리거의 조건 영역이 참이거나 조건 영역이 생략된 경우 수행될 내용을 기술하는 영역이다. 실행 영역 절에 특정 시점(**AFTER** 나 **DEFERRED**)이 주어지지 않으면, 실행 영역은 트리거 이벤트와 같은 시점에서 수행된다.

아래 목록은 트리거를 정의할 때 사용할 수 있는 실행 영역의 목록이다.

- **REJECT**: 트리거에서 조건 영역이 참이 아닌 경우 트리거를 발동시킨 연산은 거절되고 데이터베이스의 예전 상태를 그대로 유지한다. 연산이 수행된 후에는 거절할 수 없기 때문에 **REJECT** 는 실행 시점이 **BEFORE** 일 때만 허용된다. 따라서 실행 시점이 **AFTER** 나 **DEFERRED** 인 경우 **REJECT** 를 사용해서는 안 된다.
- **INVALIDATE TRANSACTION**: 트리거를 부른 이벤트 연산은 수행되지만, 커밋을 포함하고 있는 트랜잭션은 수행되지 않도록 한다. 트랜잭션이 유효하지 않으면 반드시 **ROLLBACK** 문으로 취소시켜야 한다. 이러한 실행은 데이터를 변경하는 이벤트가 발생한 후에 유효하지 않은 데이터를 가지는 것으로부터 데이터베이스를 보호하기 위해 사용된다.
- **PRINT**: 터미널 화면에 텍스트 메시지로 트리거 활동을 가시적으로 보여주기 때문에 트리거의 개발이나 시험하는 도중에 사용될 수 있다. 이벤트 연산의 결과를 거절하거나 무효화시키지는 않는다.

- **INSERT**: 테이블에 하나 혹은 그 이상의 새로운 인스턴스를 추가한다.
- **UPDATE**: 테이블에 있는 하나 혹은 그 이상의 칼럼 값을 변경한다.
- **DELETE**: 테이블로부터 하나 혹은 그 이상의 인스턴스를 제거한다.

다음은 트리거 생성 시에 실행영역의 정의 방법을 보여주는 예제이다. *medal_trig* 트리거는 실행 영역에 **REJECT** 를 지정하였다. **REJECT** 는 실행 시점이 **BEFORE** 일 때만 지정 가능하다.

```
CREATE TRIGGER medal_trig
BEFORE UPDATE ON participant
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;
```

참고

- **INSERT** 이벤트가 정의된 트리거의 실행 영역에 **INSERT** 를 사용할 때는 트리거가 무한 루프에 빠질 수 있으므로 주의해야 한다.
- **UPDATE** 이벤트가 정의된 트리거가 분할된 테이블에서 동작하는 경우, 정의된 분할이 깨지거나 의도하지 않은 오동작이 발생할 수 있으므로 주의해야 한다. 이를 방지하기 위해 CUBRID는 트리거가 동작중인 경우 분할 변경을 야기하는 **UPDATE** 가 실행되지 않도록 오류 처리한다. **UPDATE** 이벤트가 정의된 트리거의 실행 영역에 **UPDATE** 를 사용할 때는 무한 루프에 빠질 수 있으므로 주의해야 한다.

트리거의 커멘트

트리거의 커멘트를 다음과 같이 명시할 수 있다.

```
CREATE TRIGGER trg_ab BEFORE UPDATE on abc(c) EXECUTE UPDATE cube_ab
SET sumc = sumc + 1
COMMENT 'test trigger comment';
```

트리거의 커멘트는 다음 구문에서 확인할 수 있다.

```
SELECT name, comment FROM db_trigger;
SELECT trigger_name, comment FROM db_trigger;
```

또는 CSQL 인터프리터에서 스키마를 출력하는 ;sc 명령으로 트리거의 커멘트를 확인할 수 있다.

```
$ csql -u dba demodb
csql> ;sc tbl
```

트리거 커멘트의 변경은 아래의 **ALTER TRIGGER** 문을 참고한다.

ALTER TRIGGER

트리거 정의에서 **STATUS** 와 **PRIORITY** 옵션에 대해 **ALTER** 구문을 이용하여 변경할 수 있다. 만약 트리거의 다른 부분에 대해 변경(이벤트 대상 또는 조건 표현식)이 필요하다면, 트리거를 삭제한 후 재생성해야 한다.

```
ALTER TRIGGER [schema_name.]trigger_name <trigger_option> ;

<trigger_option> ::=
    STATUS { ACTIVE | INACTIVE } |
    PRIORITY key
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 변경할 트리거의 이름을 지정한다.
- **STATUS { ACTIVE | INACTIVE }**: 트리거의 상태를 변경한다.
- **PRIORITY key**: 우선순위를 변경한다.

다음은 medal_trig 트리거를 생성하고 트리거의 상태를 **INACTIVE** 로, 우선순위를 0.7로 변경하는 예제이다.

```
CREATE TRIGGER medal_trig
STATUS ACTIVE
BEFORE UPDATE ON participant
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;

ALTER TRIGGER medal_trig STATUS INACTIVE;
ALTER TRIGGER medal_trig PRIORITY 0.7;
```

참고

- 같은 **ALTER TRIGGER** 문 내에서는 한 개의 *trigger_option* 만 기술할 수 있다.
- 만약 테이블 트리거를 변경하려면, 해당 트리거의 소유자이거나, 해당 트리거가 있는 테이블에 대해 **ALTER** 권한이 부여되어 있어야 한다.

- 사용자 트리거를 변경하기 위해서는 반드시 해당 트리거의 소유자여야 한다. *trigger_option*에 대한 자세한 내용은 **CREATE TRIGGER** 을 참조한다. **PRIORITY** 옵션과 같이 기술하는 key는 반드시 음이 아닌 부동 소수점 값(non-negative floating point value)이어야 한다.

트리거 커멘트

트리거 커멘트는 **ALTER TRIGGER** 문을 실행하여 다음과 같이 변경할 수 있다.

```
ALTER TRIGGER [schema_name.]trigger_name [trigger_option]
[COMMENT 'comment_string'];
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 변경할 트리거의 이름을 지정한다.
- *comment_string*: 트리거의 커멘트를 지정한다.

트리거의 커멘트만 변경하는 경우 트리거 옵션(trigger_option)을 생략할 수 있다.

*trigger_option*은 위의 **ALTER TRIGGER** 구문을 참고한다.

```
ALTER TRIGGER trg_ab COMMENT 'new trigger comment';
```

DROP TRIGGER

DROP TRIGGER 구문을 이용하여 트리거를 삭제한다.

```
DROP TRIGGER [schema_name.]trigger_name ;
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 삭제할 트리거의 이름을 지정한다.

다음은 medal_trig 트리거를 삭제하는 예제이다.

```
DROP TRIGGER medal_trig;
```

참고

- 트리거가 사용자 트리거(즉 트리거 이벤트가 **COMMIT** 이거나 **ROLLBACK**)이면, 트리거의 소유자만 볼 수 있고 소유자만 제거할 수 있다.

- 한 개의 **DROP TRIGGER** 문에서는 한 개의 트리거만 제거할 수 있다. 테이블 트리거는 트리거가 속해 있는 테이블에 대해 **ALTER** 권한이 있는 사용자에게 의해 제거될 수 있다.

RENAME TRIGGER

트리거의 이름은 **RENAME** 구문의 **TRIGGER** 예약어를 이용해서 변경한다.

```
RENAME TRIGGER [schema_name.]old_trigger_name {AS | TO}
[schema_name.]new_trigger_name ;
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다. 현재 트리거의 스키마와 변경할 트리거의 스키마가 동일해야 한다.
- *old_trigger_name*: 트리거의 현재 이름을 입력한다.
- *new_trigger_name*: 변경할 트리거의 이름을 지정한다.

```
RENAME TRIGGER medal_trigger AS medal_trig;
```

참고

- 트리거 이름은 사용자가 소유하고 있는 트리거 중에서 유일해야 한다. 하지만 데이터베이스 내의 테이블 이름과 같거나 다른 사용자가 소유하고 있는 트리거의 이름과는 같을 수 있다.
- 만약 테이블 트리거의 이름을 변경하려면, 트리거의 소유자이거나, 해당 트리거가 있는 테이블에 대해 **ALTER** 권한이 부여되어 있어야 한다. 사용자 트리거는 트리거의 소유자만 이름을 변경할 수 있다.

지연된 트리거

지연된 트리거 실행영역과 조건 영역은 나중에 실행되거나 취소될 수 있다. 이러한 트리거들은 이벤트 시점(event time)이나 실행 영역(action) 절에 **DEFERRED** 시간 옵션을 포함하고 있다. **DEFERRED** 옵션이 이벤트 시점에 기술되고, 실행 영역 앞에 시간이 생략되었다면, 실행 영역은 자동으로 지연된다.

지연된 영역 실행

지연된 트리거의 조건 영역이나 실행 영역을 즉시 실행시킨다.

```
EXECUTE DEFERRED TRIGGER <trigger_identifier> ;

<trigger_identifier> ::=
    [schema_name.]trigger_name |
    ALL TRIGGERS
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 트리거의 이름을 지정하면 지정된 트리거의 지연된 활동이 실행된다.
- **ALL TRIGGERS**: 현재 모든 지연된 활동이 실행된다.

지연된 영역 취소

지연된 트리거의 조건 영역과 실행 영역을 취소한다.

```
DROP DEFERRED TRIGGER <trigger_identifier> ;

<trigger_identifier> ::=
    [schema_name.]trigger_name |
    ALL TRIGGERS
```

- *schema_name*: 트리거의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *trigger_name*: 트리거의 이름을 지정하면 지정된 트리거의 지연된 활동이 취소된다.
- **ALL TRIGGERS**: 현재 모든 지연된 활동이 취소된다.

트리거 권한 부여

트리거에 대한 권한은 명시적으로 부여되지 않는다. 트리거의 정의에 기술된 이벤트 대상 테이블에 권한이 부여되었을 때 사용자는 테이블 트리거에 대한 권한을 자동적으로 획득한다. 다시 말하자면, 테이블 대상(**INSERT**, **UPDATE** 등)을 가지는 트리거는 해당 테이블에 적절한 권한을 가지는 모든 사용자에게 보인다. 사용자 트리거(**COMMIT** 과 **ROLLBACK**)는 트리거를 정의한 사용자만 볼 수 있다. 트리거의 소유자이면 모든 권한은 자동적으로 부여된다.

참고

- 테이블 트리거를 정의하기 위해서는 관련된 테이블에 **ALTER** 권한이 반드시 있어야 한다.
- 사용자 트리거를 정의하기 위해서는 유효한 사용자를 이용하여 데이터베이스에 접근하는 것이 필요하다.

REPLACE와 INSERT ... ON DUPLICATE KEY UPDATE에서의 트리거

CUBRID에서는 **REPLACE** 문과 **INSERT ... ON DUPLICATE KEY UPDATE** 문 실행 시 내부적으로 **DELETE**, **UPDATE**, **INSERT** 작업이 발생하면서 해당 트리거가 실행된다. 다음 표는 **REPLACE** 혹은 **INSERT ... ON DUPLICATE KEY UPDATE** 문이 수행될 때 발생하는 이벤트에 따라 CUBRID에서 트리거가 어떤 순서로 동작하는지를 나타낸다. **REPLACE** 문과 **INSERT ... ON DUPLICATE KEY UPDATE** 문 모두 상속받은 클래스(테이블)에서는 트리거가 동작하지 않는다.

REPLACE와 INSERT ... ON DUPLICATE KEY UPDATE 문에서 트리거의 동작 순서

이벤트	트리거 동작 순서
REPLACE 레코드가 삭제되고 삽입될 때	BEFORE DELETE > AFTER DELETE > BEFORE INSERT > AFTER INSERT
INSERT ... ON DUPLICATE KEY UPDATE 레코드가 업데이트될 때	BEFORE UPDATE > AFTER UPDATE
REPLACE, INSERT ... ON DUPLICATE KEY UPDATE 레코드가 삽입만 될 때	BEFORE INSERT > AFTER INSERT

다음은 *with_trigger* 테이블에 **INSERT ... ON DUPLICATE KEY UPDATE** 와 **REPLACE** 를 수행하면 트리거가 동작하여 *trigger_actions* 테이블에 레코드를 삽입하는 예제이다.

```
CREATE TABLE with_trigger (id INT UNIQUE);
INSERT INTO with_trigger VALUES (11);

CREATE TABLE trigger_actions (val INT);

CREATE TRIGGER trig_1 BEFORE INSERT ON with_trigger EXECUTE INSERT
INTO trigger_actions VALUES (1);
CREATE TRIGGER trig_2 BEFORE UPDATE ON with_trigger EXECUTE INSERT
INTO trigger_actions VALUES (2);
CREATE TRIGGER trig_3 BEFORE DELETE ON with_trigger EXECUTE INSERT
INTO trigger_actions VALUES (3);

INSERT INTO with_trigger VALUES (11) ON DUPLICATE KEY UPDATE id=22;

SELECT * FROM trigger_actions;
```

```

      va
=====
      2

```

```

REPLACE INTO with_trigger VALUES (22);

SELECT * FROM trigger_actions;

```

```

      va
=====
      2
      3
      1

```

트리거 디버깅

트리거를 정의한 후에는 트리거가 의도한 대로 동작하는지 검사하는 것이 좋다. 종종 트리거가 기대했던 것보다 처리하는데 오랜 시간이 걸리는 경우가 있다. 이는 시스템에 너무 많은 오버헤드를 주거나, 재귀적 루프에 빠졌다는 뜻이다. 이 절에서는 트리거를 디버그하는 몇 가지 방법을 설명한다.

다음은 호출되면 재귀적으로 루프에 빠지도록 정의한 트리거이다. *loop_trg* 트리거는 목적이 다소 인위적이지만 트리거를 디버그하기 위한 예제로 사용될 수 있다.

```

CREATE TRIGGER loop_tgr
BEFORE UPDATE ON participant(gold)
IF new.gold > 0
EXECUTE UPDATE participant
    SET gold = new.gold - 1
    WHERE nation_code = obj.nation_code AND host_year =
obj.host_year;

```

트리거 실행 로그 보기

SET TRIGGER TRACE 문을 이용하여 터미널에서 트리거의 실행 로그를 볼 수 있다.

```

SET TRIGGER TRACE <switch> ;

<switch> ::=
    ON |
    OFF

```

- **ON: TRACE** 가 작동되며 **OFF** 하거나 현재 데이터베이스 세션을 종료할 때까지 계속 유지된다.

- **OFF: TRACE**의 작동을 멈춘다.

다음 예제는 트리거의 실행 로그를 보기 위해 **TRACE**를 작동시키고 *loop_trg* 트리거를 작동시키는 예제이다. 트리거가 호출될 때 수행된 각각의 조건 영역과 실행 영역에 대한 추적을 식별하기 위한 메시지가 터미널에 나타난다. *loop_trg* 트리거는 *gold* 값이 0이 될 때까지 실행되므로 예제에서는 아래의 메시지가 15번 나타난다.

```
SET TRIGGER TRACE ON;
UPDATE participant SET gold = 15 WHERE nation_code = 'KOR' AND
host_year = 1988;
```

```
TRACE: Evaluating condition for trigger "loop".
TRACE: Executing action for trigger "loop".
```

중첩된 트리거 제한

SET TRIGGER 문의 **MAXIMUM DEPTH** 키워드를 이용하여 단계적으로 발동되는 트리거 수를 제한할 수 있다. 이를 이용하면 재귀적으로 호출되는 트리거가 무한루프에 빠지는 것을 막을 수 있다.

```
SET TRIGGER [ MAXIMUM ] DEPTH count ;
```

- *count*: 양의 정수값으로 트리거가 다른 트리거나 자신을 재귀적으로 발동할 수 있는 횟수를 지정한다. 트리거의 수가 최대 깊이에 도달하면 데이터베이스 요청은 중단되고 트랜잭션은 유효하지 않은 것처럼 표시된다. 설정된 **DEPTH**는 현재 세션을 제외한 나머지 모든 트리거에 적용된다. 최대값은 32이다.

다음은 재귀적 트리거 호출의 최대 값을 10으로 설정하는 예제이다. 이는 이후에 발동하는 모든 트리거에 적용된다. 이 예제에서 *gold* 칼럼에 대한 값은 15로 갱신되어 트리거는 총 16번 불려지게 된다. 이는 현재 설정된 최대 깊이를 초과하게 되고 아래와 같은 에러 메시지가 발생한다.

```
SET TRIGGER MAXIMUM DEPTH 10;
UPDATE participant SET gold = 15 WHERE nation_code = 'KOR' AND
host_year = 1988;
```

```
ERROR: Maximum trigger depth 10 exceeded at trigger "loop_trg".
```

참고

DEFERRED 트리거는 호출되어도 동작이 즉시 실행되지 않고 트랜잭션이 종료될 때까지 지연된 활동으로 누적되므로, 한 트랜잭션에 누적되는 지연된 활동이 많아질수록 CAS의 메모리 사용량

도 함께 증가한다. 특히 트리거가 서로를 반복해서 호출하거나 한 트랜잭션에서 많은 수의 **DEFERRED** 트리거가 호출되면 지연된 활동이 과도하게 누적될 수 있으며, 이때 CAS의 메모리 사용량이 **APPL_SERVER_MAX_SIZE_HARD_LIMIT**에 도달하면 해당 CAS는 재구동된다.

MAXIMUM DEPTH는 트리거가 중첩 호출되는 깊이만 제한하며 누적되는 지연된 활동의 총 개수는 제한하지 않으므로, 이러한 누적은 **MAXIMUM DEPTH**로 방지할 수 없다. 따라서 트리거가 서로를 순환 호출하지 않도록 하고, 한 트랜잭션에서 대량의 행을 변경하는 작업에는 **DEFERRED** 트리거를 사용하지 않는 등 누적되는 지연된 활동의 수가 과도해지지 않도록 트리거를 설계해야 한다.

트리거를 이용한 응용

여기에서는 데모 데이터베이스에 있는 트리거 정의에 대해 알아본다. *demodb* 데이터베이스에 생성되어 있는 트리거는 그리 복잡하지는 않지만 CUBRID에서 사용할 수 있는 대부분의 기능을 활용한다. 이러한 트리거를 테스트할 때, *demodb* 데이터베이스의 원형을 유지하고 싶다면 데이터에 변경이 발생한 후 롤백을 수행해야 한다.

사용자 데이터베이스에 직접 생성한 트리거는 사용자가 만든 응용 프로그램만큼이나 강력할 수 있다.

participant 테이블에 만들어진 아래 트리거는 제시된 값이 0보다 작을 때 메달 칼럼(*gold*, *silver*, *bronze*)에 대한 업데이트를 거절한다. 트리거의 조건에 상관명 *new*가 사용되었기 때문에 시작 시점(evaluation time)은 반드시 **BEFORE**가 되어야 한다. 비록 기술하지는 않았지만, 이 트리거에서 실행 시점(action time) 또한 **BEFORE**이다.

```
CREATE TRIGGER medal_trigger
BEFORE UPDATE ON participant
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;
```

국가 코드가 'BLA'인 나라의 금메달(*gold*) 수를 업데이트 할 때, *medal_trigger* 트리거가 발동한다. 금메달 수가 음수인 경우를 허용하지 않도록 트리거를 생성하였으므로, 업데이트를 허용하지 않는다.

```
UPDATE participant
SET gold = -10
WHERE nation_code = 'BLA';
```

아래 트리거는 위의 예제와 같은 조건인데, **STATUS ACTIVE**가 추가된 경우이다. **STATUS** 문이 생략될 경우 기본값은 **ACTIVE**이며, **ALTER TRIGGER** 문에 의해 **STATUS**를 **INACTIVE**로 변경할 수 있다.

STATUS의 값에 따라 트리거의 실행 여부를 지정할 수 있다.

```
CREATE TRIGGER medal_trig
STATUS ACTIVE
BEFORE UPDATE ON participant
IF new.gold < 0 OR new.silver < 0 OR new.bronze < 0
EXECUTE REJECT;

ALTER TRIGGER medal_trig
STATUS INACTIVE;
```

다음 트리거는 트랜잭션이 커밋되었을 때 어떻게 무결성 제약 조건을 강제로 수행하는지 보여 준다. 하나의 트리거가 여러 테이블에 대해 지정 조건을 넣을 수 있다는 점이 앞의 예제와 다르다.

```
CREATE TRIGGER check_null_first
BEFORE COMMIT
IF 0 < (SELECT count(*) FROM athlete WHERE gender IS NULL)
OR 0 < (SELECT count(*) FROM game WHERE nation_code IS NULL)
EXECUTE REJECT;
```

다음 트리거는 *record* 테이블에 대해서 트랜잭션이 커밋될 때까지 업데이트 무결성 제약조건 검사를 지연시킨다. **DEFERRED** 키워드가 이벤트 시점으로 주어졌기 때문에 업데이트 실행 시점에 즉시 트리거가 실행되지는 않는다.

```
CREATE TRIGGER deferred_check_on_record
DEFERRED UPDATE ON record
IF obj.score = '100'
EXECUTE INVALIDATE TRANSACTION;
```

record 테이블에서 업데이트가 완료되었을 때, 해당 업데이트는 현재 트랜잭션의 마지막(커밋이나 롤백할 때)에 확인하게 된다. 상관명 **old** 는 **DEFERRED UPDATE** 를 사용하는 트리거의 조건 절에 사용할 수 없다. 따라서 아래와 같은 트리거는 생성할 수 없다.

```
CREATE TABLE foo (n int);
CREATE TRIGGER foo_trigger
DEFERRED UPDATE ON foo
IF old.n = 100
EXECUTE PRINT 'foo_trigger';
```

위와 같이 트리거를 생성하려고 하면 다음과 같은 에러 메시지를 보여주고, 실패한다.

```
ERROR: Error compiling condition for 'foo_trigger' : old.n is not
defined.
```

상관명 **old** 는 트리거의 조건 시간이 **AFTER** 일 때에만 사용될 수 있다.

CUBRID DBLink

CUBRID DBLink 소개

데이터베이스에서 정보를 조회하다 보면 종종 외부 데이터베이스의 정보 조회가 필요한 경우가 있다. 이렇게 외부 데이터베이스의 정보를 조회하기 위해서 CUBRID DBLink를 이용하면 타 데이터베이스의 정보를 조회할 수 있다.

CUBRID DBLink는 동일 기종인 CUBRID와 이기종인 Oracle, MySQL, MariaDB의 데이터베이스의 정보를 조회할 수 있도록 기능을 제공하고 있다. 외부 데이터베이스의 정보를 마치 하나의 데이터베이스에서 조회하는 것과 같은 효과를 발휘한다. 단 외부 데이터베이스를 여러 개 설정은 가능 하나, 정보를 조회할 때는 한 개의 타 데이터베이스의 정보만 조회할 수 있다.

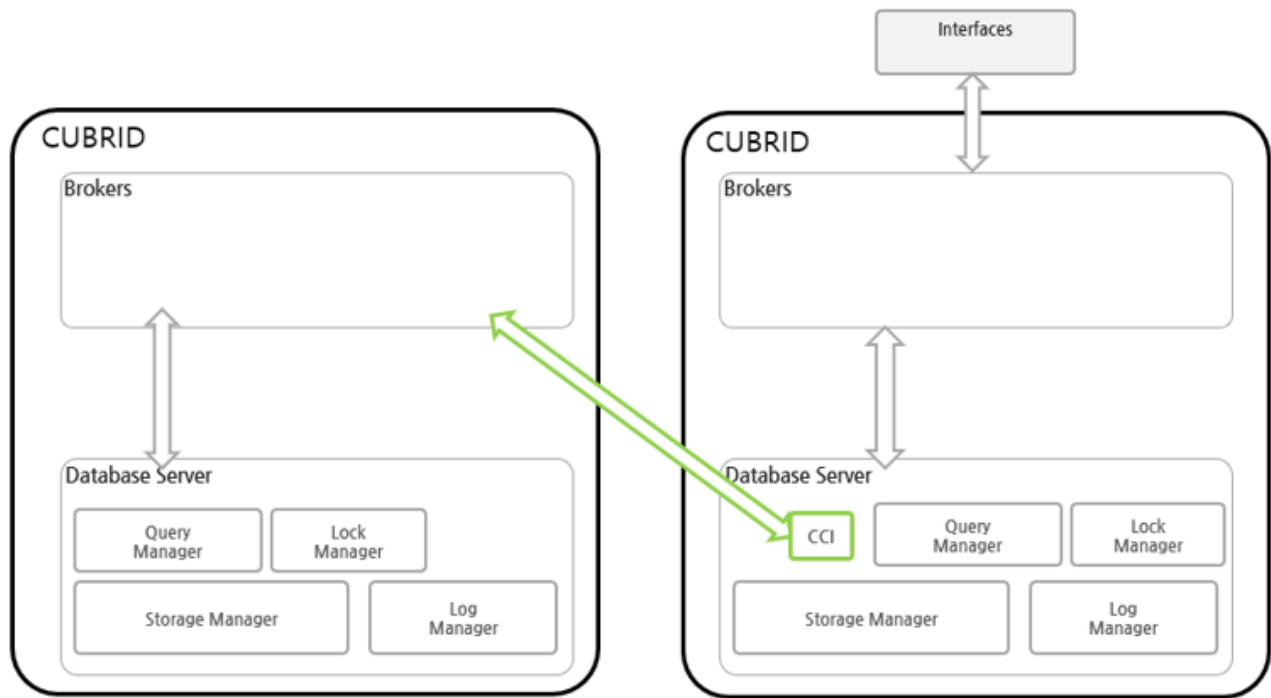
CUBRID DBLink는 SELECT의 FROM절에 연결될 서버와 실행될 질의를 명시한 DBLINK 구문 형식과 원격 테이블 (테이블 확장명) 형식으로 사용 가능하며, INSERT/REPLACE/UPDATE/DELETE/MERGE구문은 원격 테이블 형식만 사용할 수 있다.

CUBRID DBLink 구성도

CUBRID DBLink는 동일기종 간에 DBLink와 이기종 간의 DBLink를 지원한다.

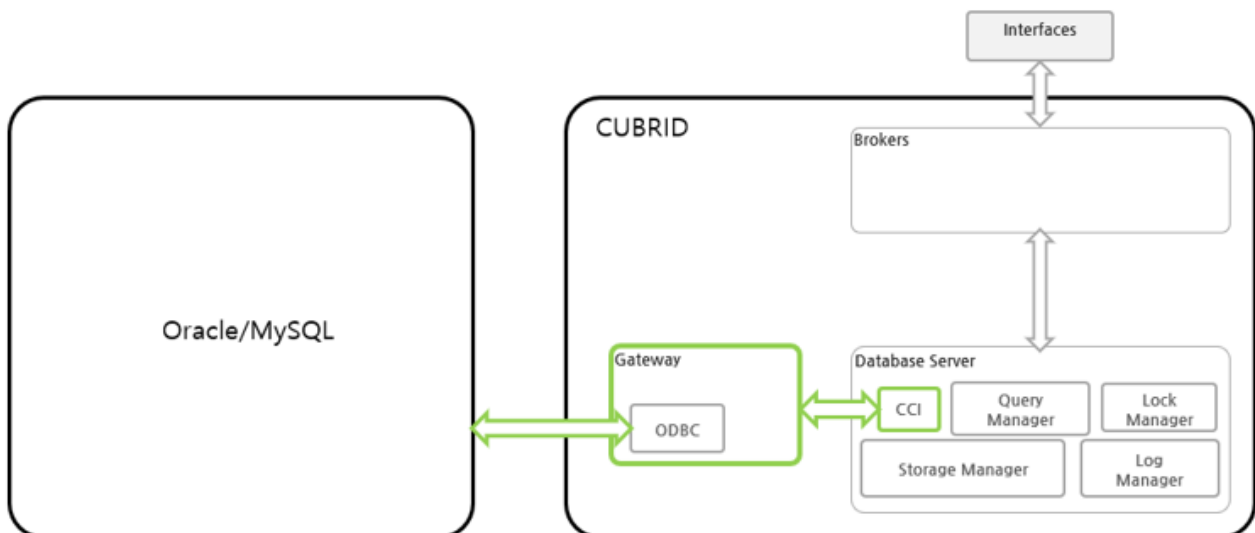
동일기종 간의 DBLink 구성도

동일기종의 외부 데이터베이스의 정보를 조회하기 위한 구성도를 보면 Database Server에서 CCI를 이용하여 동일기종의 Brokers에 접속하여 외부 데이터베이스의 정보를 조회할 수 있다.



이기종 간의 Dblink 구성도

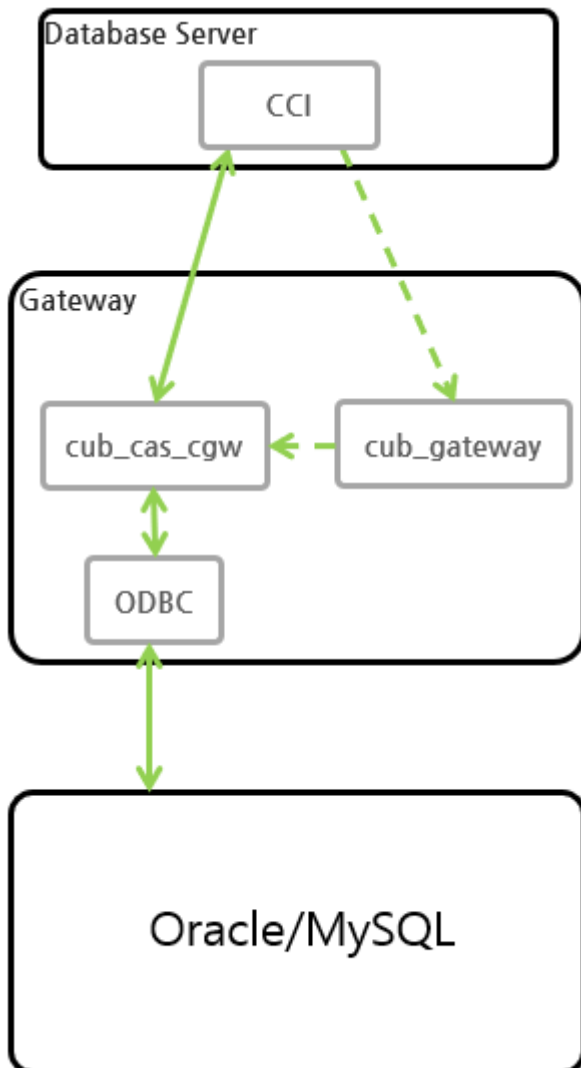
이기종 데이터베이스의 정보를 조회하기 위한 구성도를 보면 게이트웨이를 통해서 이기종 데이터베이스의 정보를 조회할 수 있다. 게이트웨이는 연결하는 데이터베이스의 ODBC(Open DataBase Connectivity) 드라이버를 이용하고 있다.



DBLink를 위한 게이트웨이

게이트웨이는 CUBRID 데이터베이스와 이기종 데이터베이스 간의 중개하는 미들웨어로 브로커 (Broker)와 유사하다. 게이트웨이는 이기종 데이터베이스 서버 (Oracle/MySQL/MariaDB 등)에 연결하고 데이터를 조회하여 CUBRID 데이터베이스 서버에 전달하는 역할을 한다.

게이트웨이를 포함하는 큐브리드 시스템은 아래 그림과 같이 cub_gateway, cub_cas_cgw를 포함한 다중 계층 구조를 가진다.



cub_cas_cgw

cub_cas_cgw(CAS Gateway)는 CUBRID Database Server에서 외부의 Database의 연결을 요청하는 공용 서버 역할을 한다. 또한, cub_cas_cgw는 데이터베이스 서버의 클라이언트로 동작하여 CUBRID Database Server의 요청에 의해 외부 데이터베이스 서버와 연결을 제공한다. 서비스 풀(service pool) 내에서 구동되는 cub_cas_cgw의 개수는 cubrid_gateway.conf 설정 파일에 지정할 수 있으며, cub_gateway에 의해 동적으로 조정된다.

cub_gateway

cub_gateway는 CUBRID Database Server와 cub_cas_cgw 사이의 연결을 중개하는 기능을 수행한다. 즉, CUBRID Database Server가 접근을 요청하면, cub_gateway는 공유 메모리(shared memory)를 통해 cub_cas_cgw의 상태를 파악하여 할당 가능한 cub_cas_cgw에게 요청을 전달하고, 해당 cub_cas_cgw로부터 전달받은 요청에 대한 처리 결과를 CUBRID Database Server에게 반환한다.

또한, cub_gateway는 서비스 풀 내의 cub_cas_cgw 개수를 조정하여 서버 부하를 관리하고, cub_cas_cgw의 구동 상태를 모니터링 및 관리한다. 만약, CUBRID Database Server의 요청을 cub_cas_cgw 1에게 전달하였는데, 비정상적인 종료로 인해 cub_cas_cgw 1과의 연결이 실패하면, cub_gateway는 CUBRID Database Server에게 연결 실패에 관한 에러 메시지를 전송하고 cub_cas_cgw 1을 재구동한다. 새롭게 구동된 cub_cas_cgw 1은 정상적인 대기 상태가 되어, 새로운 응용 클라이언트의 요청에 의해 재연결된다.

공유 메모리

공유 메모리에는 cub_cas_cgw의 상태 정보가 저장되며, cub_gateway는 공유 메모리에 저장된 cub_cas_cgw의 상태 정보를 참조하여 CUBRID Database Server와의 연결을 중개한다. 공유 메모리에 저장된 cub_cas_cgw의 상태 정보를 통해 시스템 관리자는 어떤 cub_cas_cgw가 현재 작업을 수행 중인지 확인할 수 있다.

게이트웨이 구동

게이트웨이를 구동하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway start
```

이미 게이트웨이가 구동 중이라면 다음과 같은 메시지가 출력된다.

```
$ cubrid gateway start
@cubrid gateway is running.
```

게이트웨이 종료

게이트웨이를 종료하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway stop
```

이미 게이트웨이가 종료되었다면 다음과 같은 메시지가 출력된다.

```
$ cubrid gateway stop
@cubrid gateway stop
++ cubrid gateway is not running.
```

게이트웨이 재시작

게이트웨이를 재시작하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway restart
```

게이트웨이 상태 확인

cubrid gateway status 는 다양한 옵션을 제공하며, 각 게이트웨이의 처리 완료된 작업 수, 처리 대기 중인 작업 수를 포함한 게이트웨이 상태 정보를 확인할 수 있도록 한다.

게이트웨이 상태 정보는 브로커와 동일하므로 **브로커 상태 확인**을 참조 한다.

```
cubrid gateway status [options] [expr]
```

CUBRID 서비스 시작시 게이트웨이 함께 시작

CUBRID 서비스 시작(**cubrid service start**) 시 게이트웨이 를 같이 시작되게 하려면, **cubrid.conf** 파일의 **service** 파라미터에 *gateway* 를 설정한다.

```
# cubrid.conf

[service]

service=server,broker,gateway,manager

...
```

CUBRID DBLINK 설정

CUBRID DBLink를 사용하기 위한 설정은 동일기종 DBLink와 이기종 DBLink의 설정이 다르다.

동일기종 DBLink 설정

위의 동일기종 구성도를 보면 원격지 데이터베이스의 Broker에 연결을 해야 하므로 원격지 데이터베이스의 Broker 설정이 필요 하다. 이 설정은 일반적인 Broker 설정과 동일하다.

이기종 DBLink 설정

이기종 데이터베이스(Oracle/MySQL/MariaDB)와 연결하기 위해서는 cubrid_gateway.conf 와 unixODBC 설치, ODBC Driver 정보 설정이 필요 하다.

게이트웨이 설정 파일

CUBRID 설치 시 생성되는 기본 게이트웨이 설정 파일인 `cubrid_gateway.conf` 에서 사용되는 파라미터는 브로커 파라미터와 거의 동일 하며, 추가로 반드시 변경해야 할 일부 파라미터가 포함된다. 기본으로 포함되지 않는 파라미터의 값은 직접 추가/편집해서 사용하면 된다. 다음은 설치 시 기본으로 제공되는 `cubrid_gateway.conf` 파일 내용이다.

```
[gateway]
MASTER_SHM_ID           =50001
ADMIN_LOG_FILE           =log/gateway/cubrid_gateway.log

[%oracle_gateway]
SERVICE                 =OFF
SSL                      =OFF
APPL_SERVER              =CAS_CGW
BROKER_PORT              =53000
MIN_NUM_APPL_SERVER      =5
MAX_NUM_APPL_SERVER      =40
APPL_SERVER_SHM_ID       =53000
LOG_DIR                  =log/gateway/sql_log
ERROR_LOG_DIR            =log/gateway/error_log
SQL_LOG                  =ON
TIME_TO_KILL             =120
SESSION_TIMEOUT          =300
KEEP_CONNECTION          =AUTO
CCI_DEFAULT_AUTOCOMMIT   =ON
APPL_SERVER_MAX_SIZE     =256
CGW_LINK_SERVER          =ORACLE
CGW_LINK_SERVER_IP       =localhost
CGW_LINK_SERVER_PORT     =1521
CGW_LINK_ODBC_DRIVER_NAME =Oracle_ODBC_Driver
CGW_LINK_CONNECT_URL_PROPERTY =

[%mysql_gateway]
SERVICE                 =OFF
SSL                      =OFF
APPL_SERVER              =CAS_CGW
BROKER_PORT              =56000
MIN_NUM_APPL_SERVER      =5
MAX_NUM_APPL_SERVER      =40
APPL_SERVER_SHM_ID       =56000
LOG_DIR                  =log/gateway/sql_log
ERROR_LOG_DIR            =log/gateway/error_log
SQL_LOG                  =ON
TIME_TO_KILL             =120
SESSION_TIMEOUT          =300
KEEP_CONNECTION          =AUTO
CCI_DEFAULT_AUTOCOMMIT   =ON
APPL_SERVER_MAX_SIZE     =256
CGW_LINK_SERVER          =MYSQL
```

```

CGW_LINK_SERVER_IP      =localhost
CGW_LINK_SERVER_PORT    =3306
CGW_LINK_ODBC_DRIVER_NAME  =MySQL_ODBC_Driver
CGW_LINK_CONNECT_URL_PROPERTY
="charset=utf8;PREFETCH=100;NO_CACHE=1"

[%mariadb_gateway]
SERVICE                =OFF
SSL                     =OFF
APPL_SERVER             =CAS_CGW
BROKER_PORT              =59000
MIN_NUM_APPL_SERVER      =5
MAX_NUM_APPL_SERVER      =40
APPL_SERVER_SHM_ID       =59000
LOG_DIR                 =log/gateway/sql_log
ERROR_LOG_DIR            =log/gateway/error_log
SQL_LOG                 =ON
TIME_TO_KILL            =120
SESSION_TIMEOUT          =300
KEEP_CONNECTION          =AUTO
CCI_DEFAULT_AUTOCOMMIT   =ON
APPL_SERVER_MAX_SIZE     =256
CGW_LINK_SERVER          =MARIADB
CGW_LINK_SERVER_IP       =localhost
CGW_LINK_SERVER_PORT     =3306
CGW_LINK_ODBC_DRIVER_NAME  =MariaDB_ODBC_Driver
CGW_LINK_CONNECT_URL_PROPERTY  =

```

게이트웨이 파라미터

이기종 데이터 베이스와 DBLink를 사용하기 위해서 설정하는 파라미터이다.

각각의 파라미터 의미는 이기종 데이터베이스 별로 약간 차이가 있다.

Parameter Name	Type	Value
APPL_SERVER	string	
CGW_LINK_SERVER	string	
CGW_LINK_SERVER_IP	string	
CGW_LINK_SERVER_PORT	int	

Parameter Name	Type	Value
CGW_LINK_ODBC_DRIVER_NAME	string	
CGW_LINK_CONNECT_URL_PROPERTY	string	

APPL_SERVER

APPL_SERVER 는 게이트웨이의 응용 서버 이름을 설정하는 부분으로 반드시 CAS_CGW로 설정해야 한다.

CGW_LINK_SERVER

CGW_LINK_SERVER 는 CAS_CGW로 연결하여 사용할 이기종 데이터베이스의 이름을 설정해야 한다. 현재 지원하는 데이터베이스는 Oracle, MySQL, MariaDB이다.

CGW_LINK_SERVER_IP

CGW_LINK_SERVER_IP 는 CAS_CGW와 연결할 이기종 데이터베이스의 IP 주소를 설정해야 한다.

참고

- Oracle인 경우, tnsnames.ora의 net_service_name을 이용하므로 해당 파라미터는 사용하지 않는다.
- 자세한 내용은 [Oracle Database에 연결을 위한 연결정보 설정](#)을 참고한다.

CGW_LINK_SERVER_PORT

CGW_LINK_SERVER_PORT 는 CAS_CGW와 연결할 이기종 데이터베이스의 Port 번호를 설정해야 한다.

참고

- Oracle인 경우, tnsnames.ora의 net_service_name을 이용하므로 해당 파라미터는 사용하지 않는다.

- 자세한 내용은 [Oracle Database에 연결을 위한 연결정보 설정](#)을 참고한다.

CGW_LINK_ODBC_DRIVER_NAME

CGW_LINK_ODBC_DRIVER_NAME 는 CAS_CGW와 연결할 때 이기종 데이터베이스에서 제공하는 ODBC Driver 이름을 설정해야 한다.

참고

- Windows에서는 해당 이기종 데이터베이스의 ODBC Driver가 설치된 경우, ODBC 데이터 원본 관리자를 통해 Driver 이름을 확인할 수 있다.
- Linux는 `odbcinit.ini`에 직접 Driver 이름을 명시해야 한다.
- 자세한 내용은 [ODBC Driver 정보 설정](#)을 참고한다.

CGW_LINK_CONNECT_URL_PROPERTY

CGW_LINK_CONNECT_URL_PROPERTY 는 이기종 데이터베이스 연결을 위한 연결 문자열(Connection String)에 사용되는 연결 속성(property)을 작성한다.

참고

- 연결 속성(property)는 데이터베이스별로 각각 다르므로 아래의 사이트를 참조한다.
- Oracle : https://docs.oracle.com/cd/B19306_01/server.102/b15658/app_odbc.htm#UNXAR418
- MySQL : <https://dev.mysql.com/doc/connector-odbc/en/connector-odbc-configuration-connection-parameters.html#codbc-dsn-option-flags>
- MariaDB : <https://mariadb.com/docs/connectors/mariadb-connector-odbc/mariadb-connector-odbc-guide#general-connection-parameters>

unixODBC 설치

unixODBC 드라이버 관리자는 Linux 및 UNIX 운영 체제에서 ODBC 드라이버 와 함께 사용할 수 있는 오픈 소스 ODBC 드라이버 관리자이다. 게이트웨이에서는 ODBC를 사용하기 위해서 unixODBC를 반드시 설치해야 한다.

참고

Windows에서는 기본으로 설치된 Microsoft® ODBC 데이터 원본 관리자 를 사용하면 된다.

unixODBC 설치 방법

```
$ wget https://www.unixodbc.org/unixODBC-2.3.9.tar.gz
$ tar xvf unixODBC-2.3.9.tar.gz
$ cd unixODBC-2.3.9
$ ./configure
$ make
$ make install
```

참고

unixODBC 드라이버 관리자 설치 방법은 아래의 url를 참고 바란다.

unixODBC 홈페이지 : <https://www.unixodbc.org/>

ODBC Driver 정보 설정

unixODBC가 설치한 후, 연결할 데이터베이스의 ODBC Driver 정보를 등록해야 한다.

ODBC Driver 정보는 odbcinst.ini를 직접 수정해서 등록한다.

아래의 내용은 MySQL, Oracle, MariaDB ODBC Driver 정보를 설정한 예제이다.

```
[MySQL ODBC 8.0 Unicode Driver]
Description = MySQL ODBC driver v8.0
Driver=/usr/lib64/libmyodbc8w.so

[Oracle 11g ODBC driver]
Description = Oracle ODBC driver v11g
Driver = /home/user/oracle/instantclient/libsqora.so.19.1

[mariadb odbc 3.1.13 driver]
Description= mariadb odbc driver 3.1.13
Driver=/home/user/mariadb-odbc-3.1.13/lib64/mariadb/libmaodbc.so
```

참고

참고로, 위의 예제에서 드라이버 이름은 각각 “MySQL ODBC 8.0 Unicode Driver”, “Oracle 11g ODBC driver” 와 “mariadb odbc 3.1.13 driver” 이다.

DBLink를 위한 Oracle 설정

Oracle 환경설정

DBLink에서 Oracle을 사용하기 위해서는 Oracle Instant Client 설치 및 설정, 연결 정보 설정, Oracle Database 환경변수 설정 및 게이트웨이 설정을 반드시 해야 한다.

오라클 인스턴트 클라이언트 ODBC 설치

Oracle Instant Client 다운로드 사이트에서 ODBC Package와 Basic Package 다운받아 동일한 디렉터리에 압축을 풉니다.

```
unzip instantclient-basic-linux.x64-19.20.0.0.0dbru.zip
unzip instantclient-odbc-linux.x64-19.20.0.0.0dbru.zip
```

Oracle Instant Client 다운로드 사이트: <https://www.oracle.com/database/technologies/instant-client/downloads.html>

오라클 인스턴트 클라이언트 환경변수 설정

```
export ORACLE_INSTANT_CLIENT=/home/user/oracle/instantclient
export PATH=$ORACLE_INSTANT_CLIENT:$PATH
export LD_LIBRARY_PATH=$ORACLE_INSTANT_CLIENT:$LD_LIBRARY_PATH
```

Oracle Database에 연결을 위한 연결정보 설정

Oracle Database에 연결을 하기 위해서는 연결정보를 가지고 있는 tnsnames.ora 파일을 수정해야 한다. 아래의 기본 형식에 HOST, PORT, SERVICE_NAME 이 세 항목에 연결정보를 작성해야 한다. 연결정보를 작성한 tnsnames.ora 파일은 TNS_ADMIN 환경변수에서 디렉터리 경로를 설정해야 한다. TNS_ADMIN 설정 방법은 [TNS_ADMIN 환경변수 설정](#)을 참고한다.

tnsnames.ora 파일의 기본 형식

```
net_service_name =
  (DESCRIPTION=
    (ADDRESS = (PROTOCOL = TCP)(HOST = xxx.xxx.xxx.xxx)(PORT =
1521)
```

```

)
(CONNECT_DATA =
  (SERVICE_NAME=service_name)
)
)

```

- net_service_name: 데이터베이스 연결을 위한 네트 서비스 이름이며, connection url의 db_name에 사용하는 이름이다.
- HOST: 데이터베이스에 연결하려는 IP 주소 또는 서버 이름이다.
- PORT: 연결에 필요한 포트이다. 대부분의 경우 기본 포트는 1521이다.
- SERVICE_NAME: 연결하려는 데이터베이스의 이름이다.

참고

참고로, net_service_name 이 중복으로 작성이 되어도 에러가 발생되지 않는다. 하지만 중복된 다른 서버에 연결될 수 있으므로, net_service_name 이 반드시 중복되지 않게 설정해야 한다.

Oracle Database 환경변수 설정

Oracle database server 에 아래의 환경변수를 설정해야 한다.

```

export ORACLE_SID=XE
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=$ORACLE_BASE/product/11.2.0/xe
export PATH=$ORACLE_HOME/bin:$PATH

```

- ORACLE_SID는 시스템 식별자이다.
- ORACLE_BASE은 오라클 기본 디렉터리 구조이다.
- ORACLE_HOME은 오라클 데이터베이스가 설치된 경로이다.

TNS_ADMIN 환경변수 설정

TNS_ADMIN는 tnsnames.ora 파일이 있는 디렉터리 경로를 가리킨다. 만약 /home/user/myconfigs 에 tnsnames.ora 파일이 있다면 아래와 같이 설정 할 수 있다.

```

export TNS_ADMIN=/home/user/myconfigs

```

Oracle을 위한 cubrid_gateway.conf 설정

게이트웨이에서 oracle에 연결하기 위해서는 아래와 같이 몇 가지 설정이 필요하다.

자세한 내용은 **게이트웨이 설정 파일**을 참고한다.

게이트웨이는 oracle에 연결하기 위해서 tnsnames.ora 의 정보를 이용하기 때문에 CGW_LINK_SERVER_IP, CGW_LINK_SERVER_PORT 는 작성하지 않아도 된다. 해당 정보를 작성한 경우에도 게이트웨이는 참조하지 않는다.

```
APPL_SERVER                =CAS_CGW
.
.
.
CGW_LINK_SERVER            =ORACLE
CGW_LINK_ODBC_DRIVER_NAME  =Oracle 19c ODBC driver
CGW_LINK_CONNECT_URL_PROPERTY =
```

DBLink를 위한 MySQL 설정

MySQL 환경설정

MySQL ODBC Driver 설치

게이트웨이에서 MySQL 연결을 하기 위해서는 MySQL Unicode ODBC Driver가 필요 하다. 아래의 내용은 MySQL ODBC Driver 설치 방법이다.

MySQL Yum 저장소 를 사용하여 Connector/ODBC RPM 패키지를 제공합니다. 시스템의 리포지토리 목록에 MySQL Yum 저장소가 있어야 하며, 없는 경우 MySQL Yum 저장소 다운로드 페이지(<https://dev.mysql.com/downloads/repo/yum/>) 에서 플랫폼에 대한 패키지를 선택하고 다운로드한다.

다운로드한 릴리스 패키지를 설치한다.

```
$ sudo yum install mysql80-community-release-el6-{version-number}.noarch.rpm
```

Yum을 사용하여 저장소를 업데이트한다.

```
$ sudo yum update mysql-community-release
```

아래의 명령으로 Connector/ODBC 를 설치한다.

```
$ sudo yum install mysql-connector-odbc
```


자세한 설치 방법은 <https://dev.mysql.com/doc/connector-odbc/en/connector-odbc-installation-binary-yum.html> 을 참고한다.

MySQL을 위한 cubrid_gateway.conf 설정

게이트웨이에서 MySQL에 연결하기 위해서는 아래와 같이 몇 가지 설정이 필요하다.

자세한 내용은 [게이트웨이 설정 파일](#) 을 참고한다.

```
APPL_SERVER                =CAS_CGW
.
.
.
CGW_LINK_SERVER            =MYSQL
CGW_LINK_SERVER_IP        =localhost
CGW_LINK_SERVER_PORT      =3306
CGW_LINK_ODBC_DRIVER_NAME  =MySQL ODBC 8.0 Unicode Driver
CGW_LINK_CONNECT_URL_PROPERTY = "charset=utf8;PREFETCH=100;NO_CACHE=1"
```

DBLink를 위한 MariaDB 설정

MariaDB 환경설정

MariaDB ODBC Driver 설치

게이트웨이에서 MariaDB 연결을 하기 위해서는 MariaDB ODBC Driver가 필요하다. 아래의 내용은 MariaDB ODBC Driver 설치 방법이다.

MariaDB Connector/ODBC 패키지는 아래의 페이지에서 버전을 선택하여 다운로드할 수 있다.

<https://mariadb.com/downloads/connectors/>

다운로드한 tarball 패키지에서 파일을 추출한다. 그리고 드라이버의 공유 라이브러리를 시스템의 적절한 위치에 설치 한다. 설치한 드라이버는 odbcinst.ini에 드라이버 정보를 설정해야 한다. 설정 방법은 [ODBC Driver 정보 설정](#) 을 참고 한다.

```
$ mariadb-connector-odbc-3.1.13-centos7-amd64.tar.gz -C mariadb-odbc-3.1.13
```

자세한 설치 방법은 <https://mariadb.com/docs/connectors/mariadb-connector-odbc/mariadb-connector-odbc-guide#installing-mariadb-connector-odbc-on-linux> 을 참고한다.

MariaDB 위한 cubrid_gateway.conf 설정

게이트웨이에서 MariaDB에 연결하기 위해서는 아래와 같이 몇 가지 설정이 필요하다.

자세한 내용은 **게이트웨이 설정 파일**을 참고한다.

```
APPL_SERVER                =CAS_CGW
.
.
.
CGW_LINK_SERVER            =MARIADB
CGW_LINK_SERVER_IP        =localhost
CGW_LINK_SERVER_PORT      =3306
CGW_LINK_ODBC_DRIVER_NAME =mariadb odbc 3.1.13 driver
```

Cubrid DBLink 사용 방법

DBLink을 사용하기 위해 연결할 CUBRID의 broker들 정보 파악 또는 이기종 데이터베이스를 위한 게이트웨이 설정이 완료되었다면, DBLink을 이용한 Query문 작성 방법에 대해서 알아본다.

데이터 조회를 위한 DBLINK Query문 작성 방법은 두가지이다.

첫번째, FROM절에 DBLINK 구문을 작성하여 타 데이터베이스의 정보를 조회하는 방법

아래의 Query문은 IP 192.xxx.xxx.xxx의 타 데이터베이스의 remote_t 테이블 정보를 조회하는 Query문이다.

```
SELECT * FROM DBLINK ('192.xxx.xxx.xxx:
53000:testdb:user:password:', 'SELECT col1, col2 FROM remote_t') AS
t(col1 int, col2 varchar(32));
```

참고

Oracle의 경우 원격접속 정보 중 ip와 port는 게이트웨이 접속 정보이고, db_name 항목에는 tnsnames.ora의 net_service_name 을 넣어야 한다. 만약 net_service_name이 ora_test 이라면 아래와 같이 작성하면 된다.

```
SELECT * FROM DBLINK ('192.xxx.xxx.xxx:53000:ora_test:user:password:','SELECT col1, col2 FROM
remote_t') AS t(col1 int, col2 varchar(32));
```

두번째, Query를 작성할 때 마다 매번 연결 정보를 작성해야 하는 번거로움과 사용자 정보(id, password)의 정보 보호를 위해 CREATE SERVER구문을 이용한다.CREATE SERVER 구문을 이용하는 경우 Query문이 간결해지고 사용자 정보 보호에 도움이 된다.

```
CREATE SERVER remote_srv ( HOST='192.xxx.xxx.xxx', PORT=53000,
DBNAME=testdb, USER=user, PASSWORD='password');
```

```
SELECT * FROM DBLINK (remote_srv, 'SELECT col1 FROM remote_t') AS
t(col1 int);
```

참고

DBLINK는 접속 URL에 추가 연결 속성(Properties)을 설정할 수 있다. 상세한 속성 내용은 [CCI드라이버의 cci_connect_with_url 함수](#)를 참조한다.

DBLINK 대상 데이터베이스가 HA 환경으로 구성된 경우 altHosts 속성을 이용해서 아래 예제처럼 설정을 할 수 있다.

```
192.168.0.1:53000:testdb:user:password::?altHosts=192.168.0.2:33000,192.168.0.3:33000
```

예제는 192.168.0.1 서버가 Active 데이터베이스이고 해당 서버에 연결할 수 없는 경우, 192.168.0.2 서버에 연결 요청하는 설정이다. 위의 예시처럼 여러개의 altHosts를 지정할 수 있으며, 나열한 순서대로 연결을 시도한다.

CREATE SERVER 구문의 PROPERTIES 항목에 연결 속성을 설정할 수 있다. 자세한 내용은 [SELECT](#) 와 [서버 정의문](#) 을 참고한다.

유의 사항

- 동의어 생성 : 원격 테이블과 원격 동의어를 대상으로 로컬 동의어가 생성이 가능하다. 단, CUBRID가 아닌 타 DBMS의 경우, user명이나 db명을 병기해야 한다.

```
-- for CUBRID
create synonym synonym_1 for t1@srv1;
create synonym synonym_2 for remote_synonym@srv1;

-- for ORACLE
create synonym synonym_ora for user_ora.t1@srv_ora;

-- for MySQL and MariaDB
create synonym synonym_my for my_db.t1@srv_mysql;
create synonym synonym_maria for maria_db.t1@srv_maria;
```

- 예약어 처리 : 원격DB의 예약어를 식별자로 사용하는 경우 예약어 처리 문자와 함께 반드시 대괄호 ([])로 감싸야 한다. CUBRID의 예약어 처리 문자는 큰따옴표(" ")문자이다.

```
SELECT ["COLUMN"],["ADD"],["ALTER"] FROM ["TABLE"]@srv1 ;
SELECT * FROM dblink(srv1, 'select "COLUMN","ADD","ALTER" from
"TABLE" ') AS t(a varchar, b varchar, c varchar );
```

- 오라클의 예약어 처리 문자는 큰따옴표(" ")문자이다.

```
SELECT ["COLUMN"],["ADD"],["ALTER"] FROM ["TABLE"]@srv1 ;
SELECT * FROM dblink(srv1, 'select "COLUMN","ADD","ALTER" from
"TABLE" ') AS t(a varchar, b varchar, c varchar );
```

- MySQL, MariaDB 예약어 처리 문자는 백쿼트(` `)이다.

```
SELECT [ `COLUMN` ],[ `ADD` ],[ `ALTER` ] FROM [ `TABLE` ]@srv1 ;
SELECT * FROM dblink(srv1, 'select `COLUMN`,`ADD`,`ALTER` from
`TABLE` ') AS t(a varchar, b varchar, c varchar );
```

제약 사항

공통 제약 사항

- 원격 DB의 문자셋(charset)은 유니코드(utf-8) 문자셋만 지원한다.
- 원격 DB 문자열 컬럼의 콜레이션(collation)은 바이너리(binary) 콜레이션(예: utf8_bin)으로 가정한다. 대소문자/엑센트 비구분(case/accent-insensitive) 등 비-바이너리 콜레이션을 사용하는 원격 DB 문자열 컬럼에 대해서는 질의에 따라 결과가 달라질 수 있으므로 올바른 동작을 보장하지 않는다(미지원).
- 테이블 확장 형식 (**object@server**) 지원
 - 테이블, 뷰, 동의어만 지원
 - 시리얼, 내장함수, 저장함수는 미지원

(예 : 원격서버(server1)의 sp_func() 저장 함수는 sp_func@server1(arg1, ...) 형식으로 사용할 수 없음)
- SELECT 질의의 모든 함수들(SYSDATE를 포함한 내장 함수, 저장 함수), serial 관련 함수 및 시스템 상수는 모두 로컬에서 실행된다. (원격DB에서 함수 또는 serial 수행이 필요한 경우에는 DBLINK 구문을 사용해야 한다.) 예를 들어 아래와 같이 원격 테이블 대상 select 질의에 대해 옵티마이저가 재작성한 질의를 보면 DBLINK() 안의 질의만 원격DB에서 실행된다.

```
SELECT A.*, rownum rn, ' ' empty, null null_col, SYSDATE
FROM t1@srv1 A ;
```

```
-- rewritten query
SELECT A.id, A.parentid, A.[text], rownum, '', null, SYS_DATE -- at
local
FROM ( SELECT [_dbl].id, [_dbl].parentid, [_dbl].[text]
      FROM DBLINK( srv1 /* '192.168.1.125:33000:remotedb1:dba::' */
        'SELECT * FROM tree A') AS [_dbl](id integer,
parentid integer, [text] varchar(32)) -- at remote
      ) A (id, parentid, [text])
```

- INSERT/UPDATE/DELETE/MERGE 질의에서 사용하는 모든 함수와, serial 관련 함수 및 시스템 상수는 모두 원격 서버에서 실행되므로 사용 시 주의가 필요하다. (즉, CUBRID의 내장함수가 원격 DBMS에서는 지원하지 않거나 사용법이 달라 오류 등의 오동작이 발생할 수 있으므로 주의 요망)
- 트랜잭션 : 로컬DB와 원격DB의 트랜잭션(commit, rollback)은 하나의 트랜잭션으로 처리되지 않는다. 원격DB의 DML(INSERT/UPDATE/DELETE/MERGE 구문) 질의는 로컬 DB의 트랜잭션과 별개로 auto commit으로 동작한다. 아래 예시 처럼 트랜잭션을 수행하는 경우, 원격DB에는 insert 수행과 함께 커밋 처리되어 데이터가 입력되고, 로컬DB에는 명시적 롤백 질의에 의해 insert 수행했던 데이터가 롤백되어 입력되지 않는다.

```
-- local input
INSERT INTO t1(a, b) VALUES (1, 'local');

-- remote input
INSERT INTO t2@srv1(a, b) VALUES (1, 'remote');

rollback;

SELECT a, b FROM t1, t2@srv1 t2 WHERE t1.a = t2.a;
there's no result
```

- TRUNCATE 구문 미지원
- CREATE TABLE ... LIKE 테이블명@server명 구문 미지원 (참고로 CREATE TABLE ... AS SELECT FROM 테이블명@server명 구문은 지원)
- TRIGGER 구문에서 DBLINK() 혹은 원격 테이블(@server)을 사용한 질의는 오류가 발생한다.
- predicate push 처리: 테이블 확장 형식(@server) 구문으로 작성된 SELECT 구문은 옵티마이저가 DBLINK()구문으로 재작성하는데, 성능 향상을 위해 원격 DB에서 처리 가능한 조건절을 함께 push하여 재작성한다. 단, 조건절에서 사용한 내장함수와 사용자 정의함수는 push에서 제외되고, 로컬 DB에서 실행된다.
- 성능 유의 사항

참고

테이블 확장 형식(@server)의 SELECT 구문에서 connect by절, group by절, having절 및 limit절을 사용한 경우, where조건, group by절, having절 및 limit절이 원격DB에서 실행되지 않고, 전체 데이터를 로컬DB로 가져온 후 해당 조건 처리를 수행함으로 성능이 저하될 수 있다.

아래는 count()를 처리하기 위해서 원격DB의 tree 테이블의 전체 데이터를 로컬DB로 가져온 후에 group by절을 처리하는 예시이다.

```
-- original query
SELECT A.parentid, count()
FROM tree@srv1 A
GROUP BY A.parentid ;

-- rewritten query
SELECT A.parentid, count()
FROM ( SELECT [_dbl].parentid
      FROM DBLINK( srv1 /* '192.168.1.125:33000:remotedb1:dba::'
*/ ,
              'SELECT parentid FROM tree A'
            ) AS [_dbl](parentid integer)
      ) A (parentid)
GROUP BY A.parentid
```

- 테이블 확장 형식(@server)구문에서 사용하는 sysdate 함수는 로컬DB에서 수행되므로 서버간의 시간이 다른 경우 주의가 필요하다.

```
-- original query
SELECT * FROM tbl@srv1 WHERE col1 >= sysdate;

-- rewritten query
SELECT *
FROM ( SELECT col1, col2
      FROM DBLINK( srv1 /* '192.168.1.125:33000:remotedb1:dba::'
*/ ,
              'SELECT col1, col2 FROM tbl'
            ) AS [_dbl](col1 date, col2 varchar)
      ) tbl (col1, col2)
WHERE col1>= SYS_DATE
```

- 테이블 확장 형식(@server)구문으로 co-related 조건의 스칼라 서브쿼리, 서브쿼리 및 EXIST 구문의 부질의 사용하는 경우, 반복되는 원격 질의 수행시 전체 데이터를 로컬 DB로 가지고 온 후 조건에 맞는 데이터를 찾는 작업을 수행하게 되어 급격한 성능 저하가 발생할 수 있다. 아래 예

ODBC SQL Type	Oracle Data Type	MySQL Data Type	MariaDB Type	Data
SQL_WLONGVARCHAR				
SQL_BIT		BIT	BIT	
SQL_BINARY		BINARY	BINARY	
SQL_VARBINARY	RAW	VARBINARY	VARBINARY	
SQL_LONGVARBINARY	LONG RAW	LONG VARBINARY	LONG VARBINARY	
	BLOB	BLOB	BLOB	
	BFILE	TINYBLOB	TINYBLOB	
		MEDIUMBLOB	MEDIUMBLOB	
		LOB	LOB	
		GEOMETRY	GEOMETRY	
		POINT	POINT	
		POLYGON	POLYGON	
		GEOMETRYCOLLECTION	GEOMETRYCOLLECTION	

ODBC SQL Type	Oracle Data Type	MySQL Data Type	MariaDB Type	Data
		MULTILINESTRING	MULTILINESTRING	
		MULTIPOINT	MULTIPOINT	
		MULTIPOLYGON	MULTIPOLYGON	
SQL_INTERVAL	INTERVAL YEAR TO MONTH			
	INTERVAL DAY TO SECOND			

- 1개 컬럼의 문자열 최대 길이는 16M이다. 아래의 표는 DBMS별 Data의 길이가 16M 이상인 Data Type 이다.

DBMS Name	Data Type
Oracle	LONG, NCLOB, CLOB
MySQL	LONGTEXT
MariaDB	LONGTEXT

- INSERT, UPDATE, DELETE, MERGE 와 같은 DML 구문에서 CUBRID가 미지원하는 내장 함수 중 function(파라미터1, ..., 파라미터N)의 형식이 아닌 경우에는 질의 오류가 발생한다.

예시: MySQL, MariaDB의 convert 함수 : convert('binary' using binary) – 오류 발생

Oracle 제약 사항

- SELECT 구문에서 LONG, INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH, BLOB, CLOB 타입은 지원하지 않는다. 자세한 내용은 **이기종 DBMS 공통 제약 사항의 미지원 타입**을 참고한다.
- INTERVAL DAY TO SECOND, INTERVAL YEAR TO MONTH 타입은 Oracle ODBC에서 지원하지 않는 타입이다. 자세한 내용은 “Using the Oracle ODBC Driver”(<https://docs.oracle.com/en/database/oracle/oracle-database/19/adfns/odbc-driver.html#GUID-3FE69BEF-F8D2-4152-9B1A-877186C47028>)를 참고한다.
- Oracle ODBC는 타임존 데이터 조회시 timestamp 타입의 로컬 시간으로 변환하여 반환된다. (타임존 데이터 타입 미지원)

아래는 Oracle DB의 타임존 데이터를 ODBC로 조회시 로컬타임존으로 변환되는 예시이다. 입력한 타임존 “+02:00”이지만, DBLink을 통해 select 하는 경우 로컬타임존 “+09:00”로 변환하여 “PM 08시”로 출력된다.

```
-- oracle input
INSERT INTO tbl VALUES (to_timestamp_tz('2021-07-25 12:34:56
+02:00', 'yyyy-mm-dd hh24:mi:ss tzh:tzm'));

-- local
SELECT t_timestamp_timezone2 FROM tbl@server;
07:34:56.000 PM 07/25/2021
SELECT to_char(t_timestamp_timezone2, 'yyyy-mm-dd hh24:mi:ss.ff
tzh:tzm') FROM tbl@server;
2021-07-25 19:34:56.000 +09:00
```

- 미지원 구문인 REPLACE 구문은 사용시 오류 발생
- CUBRID의 타입 범위를 벗어나는 오라클의 date, number 타입 데이터는 입력은 가능하나 조회 시에는 오류가 발생한다.

참고

MySQL/MariaDB 제약 사항

- MySQL에서 cache를 사용하는 경우 게이트웨이 프로세스(cub_cas_cgw)의 메모리 사용량이 증가하므로 PREFETCH, NO_CACHE=1 사용을 권장한다.
- MySQL/MariaDB에서 repeat() 함수가 포함된 질의 수행 시 문자열의 일부가 잘리거나, 문자열을 읽어오지 못할 수 있다.

- SELECT 구문에서 LONGTEXT, BIT, BLOB, LONGBLOB 타입은 지원하지 않는다. 자세한 내용은 [이 기종 DBMS 공통 제약 사항의 미지원 타입](#)을 참고한다.

클래스 상속

이 장에서는 상속 개념을 설명하기 위해 테이블은 클래스(class), 칼럼은 속성(attribute), 타입은 도메인(domain)으로 표현한다.

CUBRID 데이터베이스에 있는 클래스들은 클래스 계층 구조를 가질 수 있으며, 계층 구조를 통해 속성과 메서드를 상속할 수 있다. 예를 들면, *Employee* 클래스로부터 속성과 메서드를 상속하는 *Manager* 클래스를 생성할 수 있다. 이때 *Manager* 클래스는 *Employee* 클래스의 서브클래스라고 하고, *Employee* 클래스는 *Manager* 클래스의 수퍼클래스라고 한다. 상속을 이용하면 이미 존재하는 클래스의 구조를 재사용하므로 간단하게 클래스를 생성할 수 있다.

CUBRID는 다중 상속을 허용하므로, 하나의 클래스는 두 개 이상의 클래스로부터 속성과 메서드를 상속할 수 있다. 그러나 다중 상속을 이용하면 메서드나 속성, 클래스를 삭제하거나 추가할 때 충돌이 발생할 수 있다.

클래스 충돌은 클래스와 수퍼클래스, 또는 두 개 이상의 수퍼클래스에 같은 이름의 속성이나 메서드가 존재할 때 발생한다. 예를 들어 클래스가 두 개 이상의 클래스로부터 이름과 도메인이 같은 속성을 상속할 가능성이 있다면, 어떤 클래스의 속성을 상속할 것인지 지정해야 한다. 이때 지정된 수퍼클래스가 삭제되면, 다른 수퍼클래스로부터 이름과 도메인이 같은 속성을 상속하도록 지정해야 한다. 일반적으로는 데이터베이스 시스템이 자동으로 이와 같은 문제를 해결하지만, 시스템과 다른 방법으로 해결하고 싶다면 상속 구문을 통해 직접 지정할 수 있다.

클래스가 두 개 이상의 클래스로부터 속성을 상속할 때 속성의 이름은 같지만 도메인은 다른 경우, 서브클래스는 좀 더 상세한 도메인을 가지는 속성을 자동으로 상속한다. 예를 들어, 이름이 같고 도메인이 클래스인 속성을 두 개의 수퍼클래스가 가지고 있다면, 클래스 계층 구조에서 더 하위에 존재하는 클래스의 속성을 상속한다. 이때 해당 속성의 도메인이 각각 문자열, 정수와 같이 시스템이 제공하는 기본 타입이라면, 상속은 불가능하다.

상속 시 발생하는 충돌과 이에 대한 해결 방법은 [클래스 충돌 해결](#)에서 다룬다.

참고

상속 시 주의 사항은 다음과 같다.

- 클래스 이름은 데이터베이스 내에서 고유해야 한다. 존재하지 않는 클래스로부터 상속하는 클래스를 생성하면 에러가 발생한다.

- 한 클래스 내에서 메서드 이름과 속성의 이름은 고유해야 한다. 이름은 공백을 포함할 수 없으며 CUBRID의 예약어와 중복될 수 없다. 알파벳, '_', '#', '%' 등은 클래스 이름에 포함될 수 있으나 첫 글자가 '_' 가 될 수는 없다. 클래스 이름은 대/소문자를 구분하지 않고 소문자로 변환되어 시스템에 저장된다.
- 클래스의 암호화 (TDE) 여부는 상속되지 않는다. 자세한 내용은 **테이블 암호화 (TDE)** 을 참고한다.

참고

상속 구문에서 수퍼클래스의 이름을 알기 쉽게 명시하기 위해서 수퍼클래스 이름 앞에 사용자의 이름을 붙일 수 있다.

클래스 속성과 클래스 메서드

클래스 객체나 클래스에 저장된 모든 인스턴스의 공통적인 특징을 저장하기 위한 클래스 속성을 생성할 수 있다. 클래스 메서드는 클래스 객체에 대한 연산을 위해 생성된다. 클래스 속성이나 메서드를 생성하기 위해서는 속성이나 메서드 이름 앞에 **CLASS** 라는 키워드를 사용한다. 클래스 속성은 인스턴스와 관련이 있다기 보다는 클래스 자체와 관련이 있기 때문에 클래스 속성의 값은 하나의 값만 존재한다. 예를 들어, 클래스 속성은 클래스 메서드에 의해 결정되는 평균값을 저장하거나, 클래스가 생성된 타임스탬프 값을 저장할 수 있다. 클래스 메서드는 클래스 객체 자체에서 수행된다. 클래스 메서드는 클래스에 저장된 인스턴스 값을 집계하기 위해 사용될 수 있다.

서브클래스가 수퍼클래스를 상속할 때, 각 클래스는 클래스 속성을 위한 별도의 저장 공간을 가지므로 두 클래스의 클래스 속성은 서로 다른 값을 가질 수 있다. 따라서 수퍼클래스의 클래스 속성에 대한 값 변경은 서브클래스에 영향을 주지 않는다.

클래스 속성의 이름은 동일한 클래스 내의 인스턴스 속성의 이름과 같을 수 있다. 마찬가지로, 클래스 메서드의 이름도 동일한 클래스 내의 인스턴스 메서드의 이름과 같을 수 있다.

상속을 위한 순서 규칙

상속이 적용된 경우 다음 규칙들이 적용된다. 클래스라는 용어는 데이터베이스 내에서 클래스와 가상 클래스간 상속 개념을 서술할 때 일반적으로 사용된다.

- 수퍼클래스가 없는 객체의 경우, 속성 정의 순서는 **CREATE** 구문에서 속성을 정의한 순서와 동일하다(이 특징은 ANSI 표준이다).
- 수퍼클래스가 하나인 경우, 수퍼클래스의 속성들 다음에 지역적으로 생성된 속성이 위치한다. 수퍼클래스로부터 상속받은 속성들 간의 순서는 수퍼클래스 정의 시 정해진 순서를 따른다. 다중 상속인 경우, 클래스 정의 시 지정된 수퍼클래스의 순서에 따라 수퍼클래스의 속성 순서가 결정된다.

- 두 개 이상의 수퍼클래스가 동일한 클래스로부터 상속된 클래스라면, 두 수퍼클래스에 동시에 존재하는 속성은 한 번만 서브클래스에 상속된다. 이 때, 충돌이 발생하면 서브클래스는 첫 번째 수퍼클래스의 속성을 상속한다.
- 두 개 이상의 수퍼클래스들 사이에 이름 충돌이 발생할 경우, 이름 충돌을 해결하기 위해 **INHERIT** 구문을 사용하여 수퍼클래스의 속성 중 원하는 것만 상속받을 수 있다.
- 수퍼클래스의 속성의 이름이 **INHERIT** 구문의 별칭 기능을 통해 변경된 경우, 이름이 변경된 속성의 위치는 전과 동일하게 유지된다.

INHERIT 절

한 클래스의 서브클래스로 클래스를 생성하면 자동으로 클래스 계층 구조의 상위 클래스들의 모든 속성과 메서드를 상속받는다. 상속 시 발생할 수 있는 이름 충돌은 시스템에 의해 자동으로 처리되거나 사용자가 직접 해결할 수 있다. 사용자가 이름 충돌을 해결하고자 한다면 **CREATE CLASS** 구문에 **INHERIT** 구문을 추가하여 해결할 수 있다.

```
CREATE CLASS
.
.
.
INHERIT resolution [{, resolution}] ;

resolution:
[column_name | method_name] OF [schema_name.]superclass_name [AS
alias]
```

INHERIT 구문에 상속하고 싶은 수퍼클래스의 속성이나 메서드 이름을 지정한다. **AS** 절을 사용하여 새로운 이름으로 상속받을 수도 있으므로, 다중 상속 구문에서 이름 충돌이 발생할 경우에서 충돌을 해결할 수 있다.

ADD SUPERCLASS 절

클래스의 상속은 클래스에 수퍼클래스를 추가하여 확장할 수 있다. 이미 존재하는 클래스에 수퍼클래스를 추가하여 두 클래스 사이에 관계를 생성한다. 수퍼클래스를 추가한다는 것이 새로운 클래스를 추가한다는 것을 의미하지는 않는다.

```
ALTER CLASS
.
.
.
ADD SUPERCLASS [schema_name.]superclass_name [{,
[schema_name.]superclass_name}]
[INHERIT resolution [{, resolution}]] ;
```

```
resolution:
[column_name | method_name] OF [schema_name.]superclass_name [AS
alias]
```

추가할 수퍼클래스의 이름을 *superclass_name*에 지정한다. 위 구문을 사용하여 수퍼클래스의 속성과 메서드를 상속할 수 있다.

새로운 수퍼클래스를 추가할 경우 이름 충돌이 발생할 수 있다. 데이터베이스 시스템의 의해서 이름 충돌이 자동으로 해결될 수 없는 경우, **INHERIT** 구문을 사용하여 수퍼클래스에서 상속받을 속성이나 메서드를 지정할 수 있다. 충돌이 발생한 속성이나 메서드를 모두 상속 받기 위해서는 별칭을 사용할 수 있다. 수퍼클래스에서 발생하는 이름 충돌에 대한 자세한 설명은 **클래스 충돌 해결**을 참조한다.

*demodb*에 포함되어 있는 *event* 클래스를 상속하여 *female_event* 클래스를 생성한다면 다음과 같은 클래스 생성 문장이 수행된다.

```
CREATE CLASS female_event UNDER event;
```

DROP SUPERCLASS 절

클래스로부터 수퍼클래스를 삭제하는 것은 두 클래스 사이의 관계를 제거하는 것이다. 클래스에서 수퍼클래스를 삭제하면, 해당 클래스뿐만 아니라 그 클래스의 모든 서브클래스의 상속 관계 수정을 의미한다.

```
ALTER CLASS
•
•
•
DROP SUPERCLASS [schema_name.]superclass_name [{,
[schema_name.]superclass_name}]
[INHERIT resolution [{, resolution}]] ;

resolution:
[column_name | method_name] OF [schema_name.]superclass_name [AS
alias]
```

삭제할 수퍼클래스의 이름을 *superclass_name*에 지정한다. 수퍼클래스의 삭제에 의해 이름 충돌이 발생할 경우, 해결 방법은 **클래스 충돌 해결**을 참조한다.

다음은 *female_event* 클래스가 *event* 클래스를 상속받은 예이다.

```
CREATE CLASS female_event UNDER event;
```

다음 **ALTER** 구문은 *female_event* 클래스에서 수퍼클래스 *event* 를 삭제하는 예이다. *female_event* 클래스가 *event* 클래스로부터 상속받은 모든 속성은 더 이상 존재하지 않는다.

```
ALTER CLASS female_event DROP SUPERCLASS event;
```

클래스 충돌 해결

데이터베이스의 스키마를 변경하면 상속 관련 클래스들 사이의 속성이나 메서드에서 충돌이 발생할 수 있다. 충돌하면 대부분, CUBRID에서 자동으로 해결되지만 그렇지 않은 경우에는 사용자가 직접 충돌을 해결해야 한다. 따라서 스키마를 변경하기 전에, 충돌이 발생할 가능성을 면밀히 조사해야 한다.

두 가지 형태의 충돌이 데이터베이스 스키마를 손상시킬 수 있다. 하나는 서브클래스의 스키마가 변경되어 서브클래스와 충돌이 발생하는 경우이고 또 다른 하나는 수퍼클래스가 변경되어 서브클래스와 충돌이 발생하는 것이다. 다음은 클래스들 간 충돌을 유발하는 연산들이다:

- 속성 추가
- 속성 삭제
- 수퍼클래스의 추가
- 수퍼클래스의 삭제
- 클래스 삭제

위의 연산들로 인해 서브클래스와 충돌이 발생할 경우, CUBRID는 충돌이 발생한 서브클래스에 대해 기본 해결 방법을 적용한다. 따라서 데이터베이스 스키마는 항상 일관된 상태를 유지한다.

해결 지시자

데이터베이스 스키마를 변경하면, 기존 클래스나 속성 간의 충돌이나 상속 충돌이 발생할 수 있다. 시스템이 자동으로 충돌을 해결하지 못하거나 시스템의 해결 방법이 마음에 들지 않으면 **ALTER** 구문의 **INHERIT** 절을 사용하여 충돌을 해결하는 방법을 제시할 수 있다(흔히 해결 지시자라고 한다).

시스템이 자동적으로 충돌을 해결할 때는 상속이 존재한다면 기본적으로 앞의 상속을 유지한다. 스키마 변경으로 인해 앞의 해결 방법이 무효화된다면 시스템은 또 다른 해결 방법을 임의로 선택할 것이다. 따라서 시스템이 충돌을 해결하는 방법을 항상 예측할 수는 없으므로 가급적이면 스키마 설계 단계에서 속성이나 메서드의 과도한 재사용을 피해야 한다.

다음에서 충돌과 관련하여 논의하고 있는 사항은 속성과 메서드에 공통적으로 적용된다.

```
ALTER [class_type] [schema_name.]class_name alter_clause  
[INHERIT resolution [{, resolution }]] ;
```

```
resolution:
[column_name | method_name] OF [schema_name.]superclass_name [AS
alias]
```

수퍼클래스 충돌

수퍼클래스 추가

ALTER CLASS 구문에서 **INHERIT** 절은 선택 사항이지만 클래스의 변경에 의해 충돌이 발생할 경우에는 반드시 사용해야 하는 문장이다. **INHERIT** 절 다음에 하나 이상의 해결방법을 명시할 수 있다.

*superclass_name*에는 충돌이 발생했을 때 새로 상속받을 속성(칼럼)이나 메서드를 가지는 수퍼클래스의 이름을 명시하고, *column_name*이나 *method_name*에는 상속받을 속성이나 메서드의 이름을 명시한다. 상속받을 속성이나 메서드의 이름을 변경할 필요가 있는 경우에는 **AS** 절을 이용하여 별칭을 지정할 수 있다.

다음 예는 *demodb*의 *event* 클래스와 *stadium* 클래스를 상속받아서 *soccer_stadium* 클래스를 만든다. *event* 클래스와 *stadium* 클래스는 모두 *name*, *code* 속성을 가지고 있기 때문에 **INHERIT** 을 사용하여 상속받을 속성을 지정해야 한다.

```
CREATE CLASS soccer_stadium UNDER event, stadium
INHERIT name OF stadium, code OF stadium;
```

두 수퍼클래스 *event*, *stadium*이 *name*이라는 속성을 가지고 있고, *soccer_stadium* 클래스가 두 속성을 모두 상속받으려면, *stadium*의 *name*은 그대로 상속 받고 *event* 클래스의 *name*은 **INHERIT**의 **alias** 절을 사용하여 이름을 변경하여 상속받을 수 있다.

아래 예는 *stadium* 클래스의 *name*은 그대로 *name*으로 상속받고, *event* 클래스의 *name*은 *purpose*라는 별명으로 상속받는다.

```
ALTER CLASS soccer_stadium
INHERIT name OF event AS purpose;
```

수퍼클래스 삭제

INHERIT을 사용하여 명시적으로 속성이나 메서드를 상속한 수퍼클래스를 삭제하면 서브클래스에서 다시 이름 충돌이 발생할 수 있다. 이 경우에는 삭제할 때 명시적으로 상속받을 속성이나 메서드를 지정해야 한다.

```
CREATE CLASS a_tbl(a INT PRIMARY KEY, b INT);
CREATE CLASS b_tbl(a INT PRIMARY KEY, b INT, c INT);
CREATE CLASS c_tbl(b INT PRIMARY KEY, d INT);
```



```
CREATE CLASS a_b_c UNDER a_tbl, b_tbl, c_tbl INHERIT a OF b_tbl, b
OF b_tbl;

ALTER CLASS a_b_c
DROP SUPERCLASS b_tbl
INHERIT b OF a_tbl;
```

위의 예는 *a_tbl*, *b_tbl*, *c_tbl* 클래스를 상속받아서 *a_b_c* 클래스를 만들고, 그 중 *b_tbl* 클래스를 수퍼클래스에서 제거한다. *b_tbl* 클래스에서 *a*와 *b*를 명시적으로 상속받았기 때문에, 수퍼클래스에서 제거하기 전에 *a*와 *b*의 이름 충돌을 해결해야 한다. 하지만, *a*는 삭제할 *b_tbl* 클래스 외에 *a_tbl* 클래스에만 존재하므로 명시적으로 지정할 필요는 없다.

호환되는 도메인

충돌하는 속성이 호환되는 도메인이 아니면, 클래스 상속 구문을 수행할 수 없다.

예를 들어, 정수 타입의 *phone*이라는 속성을 가지는 수퍼클래스를 상속받은 클래스에는 문자열 타입의 *phone* 속성을 가지는 또 다른 수퍼클래스를 추가할 수 없다. 두 수퍼클래스의 *phone* 속성의 타입이 모두 문자열이거나 정수라면 **INHERIT** 구문을 이용하여 충돌을 해결하면서 수퍼클래스를 추가할 수 있다.

이름은 같지만 도메인이 다른 속성을 상속할 때 도메인 호환성이 점검된다. 이 경우, 클래스 상속 계층 구조의 하위 클래스를 도메인으로 갖는 속성이 자동으로 상속된다. 상속받을 속성들의 도메인이 호환 가능할 때, 상속 관계가 만들어지는 클래스에서 충돌이 해결되어야 한다.

서브클래스 충돌

클래스의 변경 사항은 모든 서브클래스에 자동으로 전파된다. 변화된 내용으로 인해 서브클래스에 문제가 발생한다면, CUBRID가 문제되는 서브클래스 충돌(subclass conflict)을 처리하고 시스템이 자동으로 충돌을 해결했다는 경고 메시지를 보여준다.

수퍼클래스의 추가, 속성과 메서드의 생성, 삭제로 인해 서브클래스 충돌이 발생할 수 있다. 클래스의 변경 사항은 모든 서브클래스에 영향을 미친다. 변경된 사항이 자동 전파되는 특징으로 인해 정상적인 변경도 하위 서브클래스들에 부작용을 유발할 수 있다.

속성과 메서드의 추가

서브클래스 충돌의 가장 단순한 형태는 속성을 추가할 때 발생한다. 한 수퍼클래스에 추가된 속성이 또 다른 수퍼클래스에서 이미 상속 받고 있는 속성의 이름과 동일하다면 서브클래스 충돌이 발생할 것이다. 이러한 경우 CUBRID는 이 문제를 자동으로 해결한다. 즉, 추가된 속성은 동일한 이름의 속성을 이미 상속하고 있는 모든 서브클래스에 상속되지 않는다.

다음은 *event* 클래스에 속성을 추가하는 예이다. *soccer_stadium* 클래스는 수퍼클래스로 *event*와 *stadium* 클래스를 가지며, *stadium* 클래스에는 *nation_code* 속성이 이미 존재한다. 따라서 *event* 클

래스에 *nation_code* 속성을 추가하면 *soccer_stadium* 클래스에서는 *nation_code* 속성과 관련하여 충돌이 발생하지만, CUBRID는 이 충돌을 자동으로 해결한다.

```
ALTER CLASS event
ADD ATTRIBUTE nation_code CHAR(3);
```

만약 *event* 가 *soccer_stadium* 의 수퍼클래스에서 제거되면, *stadium* 클래스의 *cost* 속성이 자동으로 상속될 것이다.

속성과 메서드의 삭제

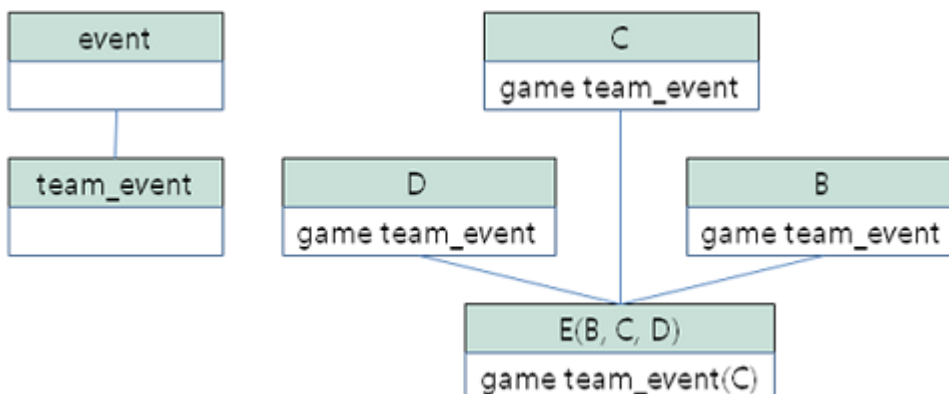
속성이 삭제되면, **INHERIT** 구문을 사용하여 그 속성을 상속받도록 한 문장의 효력 역시 사라진다. 속성이 삭제됨으로써 충돌이 발생한다면 시스템은 새로운 상속 계층 구조를 결정할 것이다. 만약, 시스템이 결정한 상속 계층 구조가 마음에 들지 않으면 **ALTER** 구문의 **INHERIT** 절을 사용하여 사용자가 계층 구조를 정할 수도 있다. 아래의 경우가 이러한 충돌에 해당할 것이다.

세 개의 서로 다른 수퍼클래스로부터 속성을 상속 받는 서브클래스가 있다고 가정하자. 모든 수퍼클래스에서 이름 충돌이 발생하였고, 이 문제를 해결하기 위해 명시적으로 상속된 속성이 삭제되었다면 나머지 두 개의 속성 중 하나가 자동으로 상속될 것이다.

다음은 서브클래스 충돌의 예이다. 클래스 *B*, *C*, *D* 는 클래스 *E* 의 수퍼클래스이고 세 개의 수퍼클래스는 이름이 *team* 이고 도메인이 *team_event* 인 속성을 가진다. 클래스 *E* 는 다음과 같이 *C* 클래스의 *place* 속성을 상속받으며 생성되었다.

```
create class E under B, C, D
inherit place of C;
```

이 경우의 상속 계층 구조는 다음과 같다:



클래스 *C* 를 수퍼클래스에서 삭제하기로 결정했다고 가정하자. 이 삭제는 상속 계층 구조의 변경을 요구할 것이다. 나머지 *B*, *D* 클래스의 *game* 속성의 도메인이 동일 레벨이므로 시스템은 둘 중 하나를 임의로 선택하여 상속할 것이다. 시스템의 임의 선택을 원하지 않으면 클래스 변경 시에 **INHERIT** 구문을 사용하여 상속받을 클래스를 지정할 수 있다.

```
ALTER CLASS E INHERIT game OF D;
ALTER CLASS C DROP game;
```

참고

한 슈퍼클래스의 *game* 속성의 도메인이 *event* 이고, 또 다른 슈퍼클래스의 속성이 *team_event* 인 경우, *team_event* 가 *event* 에 비해 더 상세하므로(상속 계층 구조상 더 하위에 존재하므로) *team_event* 를 도메인으로 가지는 속성이 상속될 것이다. 이 경우 사용자가 강제로 *event* 를 도메인으로 가지는 속성을 상속할 수는 없다. *event* 클래스는 *team_event* 보다 상속 계층 구조의 상위에 존재하기 때문이다.

스키마 불변성

데이터베이스 스키마 불변성은 항상(스키마 변경 전/후) 스키마가 지켜야 하는 스키마의 특징이다, 클래스 계층 불변성, 이름 불변성, 상속 불변성, 일관성의 불변성 등 네 가지 유형의 불변성이 존재한다.

클래스 계층 불변성

하나의 루트를 가지며 연결된 클래스들이 방향성을 갖는 비순환 그래프(DAG: directed acyclic graph)인 클래스 계층 구조를 정의한다. 즉, 루트를 제외한 모든 클래스는 하나 이상의 슈퍼클래스를 가지고 자기 자신이 슈퍼클래스가 될 수 없다. DAG의 루트는 *object*라는 시스템 정의 클래스이다.

이름 불변성

클래스 계층 구조상의 모든 클래스는 고유한 이름을 가져야 하고, 클래스 내의 모든 속성 역시 고유한 이름을 가져야 함을 의미한다. 즉, 동일한 이름의 클래스를 생성하거나 한 클래스에서 동일한 이름의 속성, 메서드를 생성하는 것은 규칙에 어긋나므로 거부된다.

이름 불변성은 **RENAME** 한정어(qualifier)에 의해 재정의된다. **RENAME** 한정어는 속성 또는 메서드의 이름이 변경될 수 있도록 한다.

상속 불변성

한 클래스는 모든 슈퍼클래스의 모든 속성들과 메서드들을 상속해야 한다는 것이다. 이 불변성은 출처 한정어, 충돌 한정어, 도메인 한정어 등 세 개의 한정어로 구분될 수 있다. 상속 이후, 상속된 속성들과 메서드들은 이름이 변경될 수 있다. 기본값 또는 공유값 속성의 경우에, 기본값과 공유값은 수정될 수 있다. 상속 불변성은 이러한 변경들이 속성들과 메서드들을 상속한 모든 클래스에 전파될 것이라는 것을 의미한다.

· 출처 한정어

클래스 S 라는 클래스를 상속한 클래스들을 클래스 C 가 다시 상속받을 경우, 클래스 S 로부터 각각의 클래스에 상속된 속성(메서드)들은 오직 하나씩만 클래스 C 에 상속될 수 있다는 것을 의미한다. 다시 말하면, 만일 한 속성(메서드)이 클래스 S 에 먼저 정의되었고, 다른 클래스들에 의해 상속되었다면, 그 속성(메서드)이 여러 개의 서브클래스에 존재하지만 실질적으로는 한 속성(메서드)인 것이다. 따라서, 한 클래스가 출처가 같은 속성(메서드)를 가지는 클래스들로부터 다중 상속 받는 경우, 오직 한 속성(메서드)의 모습만을 상속한다.

· 충돌 한정어

출처는 다르지만 동일한 이름을 가지는 속성(메서드)을 가지는 두 개 이상의 클래스를 클래스 C 가 상속한다면, 클래스 C 는 하나 이상의 클래스를 모두 상속받을 수 있다는 것이다. 동일한 이름의 속성(메서드)를 상속받으려면 이름 불변성을 위반하므로 이름 변경이 필요하다.

· 도메인 한정어

상속된 속성의 도메인이 그 도메인의 서브클래스로 변환될 수도 있음을 의미한다.

일치 불변성

데이터베이스 스키마는 스키마를 변경하는 순간을 제외하고 항상 스키마 불변성과 모든 규칙들을 준수해야 한다는 것이다.

스키마 변경 규칙

스키마 불변성에서 항상 유지되어야 하는 스키마의 특성들에 대해 언급하였다.

스키마를 변경하는 방법은 몇 가지가 존재하며 이 방법들은 스키마 불변성을 유지해야 한다. 예를 들어, 수퍼클래스를 하나만 가지는 클래스에서 그 수퍼클래스와의 관계를 제거한다고 가정하자. 수퍼클래스와의 관계가 삭제되면 그 클래스는 object 클래스의 직속 서브클래스가 되거나 만약 사용자가 그 클래스는 적어도 하나의 수퍼클래스를 가져야 한다고 명시했다면 그 삭제는 거부될 것이다. 이러한 선택은 임의적인 측면이 있지만, 스키마를 변경하는 방법 중 하나를 선택하기 위한 몇 가지 규칙을 가지는 것은 사용자나 데이터베이스 설계자에게 분명 유용할 것이다.

충돌 해결 규칙(conflict-resolution rules), 도메인 변경 규칙(domain-change rule), 클래스 계층 규칙(class-hierarchy rule)의 세 가지 형태 규칙이 적용된다.

일곱 개의 충돌 해결 규칙은 상속 불변성을 강화한다. 대부분의 스키마 변경 규칙은 이름 충돌 때문에 필요하다. 도메인 변경 규칙은 상속 불변성의 도메인 해결을 강화한다. 클래스 계층 규칙은 클래스 계층 불변성을 강화한다.

충돌 해결 규칙

- **규칙 1:** 클래스 C 의 속성(메서드) 이름이 수퍼클래스 S 의 속성 이름과 충돌이 발생한다면(이름이 같다면), 클래스 C 의 속성이 사용된다. S 의 속성은 상속되지 않는다.

어떤 클래스가 하나 이상의 수퍼클래스를 가지는 경우, 속성들이 의미적으로 같은지, 어떤 속성을 상속받을 것인지를 결정하기 위해 각 수퍼클래스가 가지는 속성(메서드)들의 세가지 측면이 고려되어야 한다. 속성(메서드)의 세 가지 측면은 이름, 도메인, 출처이다. 아래 표는 세 가지 측면에서 두 수퍼클래스에서 발생할 수 있는 여덟 가지 조합이다. 사례 1의 경우(두 개의 서로 다른 수퍼클래스의 속성이 이름, 도메인, 출처가 모두 같은 경우), 두 속성은 동일하므로 서브클래스는 둘 중 하나만 상속받아야 한다. 사례 8의 경우(두 개의 서로 다른 수퍼클래스의 속성이 이름, 도메인, 출처가 모두 다른 경우), 두 속성은 완전히 다른 속성이므로 모두 상속받아야 한다.

사례	이름	도메인	출처
1	같음	같음	같음
2	같음	같음	다름
3	같음	다름	같음
4	같음	다름	다름
5	다름	같음	같음
6	다름	같음	다름
7	다름	다름	같음
8	다름	다름	다름

8개의 사례 중 5개(1, 5, 6, 7, 8)는 명확한 의미를 가지고 있다. 상속 불변성은 이러한 경우의 충돌을 해결하기 위한 가이드 라인이다. 나머지 사례(2, 3, 4)의 경우, 충돌을 자동으로 해결하는 것은 매우 어렵다. 규칙 2, 규칙 3이 이러한 충돌의 해결 방안이 될 수 있다.

- **규칙 2:** 두 개 이상의 수퍼클래스가 출처는 다르지만 같은 이름과 도메인의 속성(메서드)을 가질 때, 사용자가 충돌 해결 구문을 사용할 경우 하나 이상의 속성(메서드)을 상속할 수 있다. 충돌 해결 구문을 사용하지 않는다면 시스템은 임의의 어느 한 속성을 선택하여 상속할 것이다.

이 규칙은 위 표의 사례 2 형태의 충돌을 해결하기 위한 가이드 라인이다.

- **규칙 3:** 두 개 이상의 수퍼클래스가 출처와 도메인은 다르지만 이름이 같은 속성(메서드)을 가질 때, 더 상세한 도메인(상속 계층 구조의 하위에 있는)을 가지는 속성(메서드)이 상속될 것이다. 도메인들 사이에 상속 관계가 없으면 스키마 변경은 허용되지 않는다.

이 규칙은 사례 3, 4 형태의 충돌을 해결하기 위한 가이드 라인이다. 규칙 3과 규칙 4가 충돌하는 경우, 규칙 3이 규칙 4보다 우선한다.

- **규칙 4:** 사용자는 사례 3, 4의 경우를 제외하면 어떠한 변경도 가능하다. 뿐만 아니라, 서브클래스에 대한 충돌 해결이 수퍼클래스에 대한 변경을 초래할 수 없다.

규칙 4의 철학은 “상속은 서브클래스가 수퍼클래스로부터 부여받은 권리로 서브클래스의 변경이 수퍼클래스에 영향을 줄 수 없다”라는 것이다. 규칙 4는 클래스 C와 수퍼클래스들 사이에 발생하는 충돌을 해결하기 위해 수퍼클래스의 포함된 속성(메서드)의 이름을 변경할 수 없다는 것을 의미한다. 규칙 4의 예외는 스키마 변경이 사례 3, 4의 충돌을 유발하는 경우이다.

- 예를 들어, 클래스 A가 클래스 B의 수퍼클래스이고, 클래스 B가 타입이 **DATE**인 *playing_date*라는 속성을 가진다고 가정하자. 클래스 A에 **STRING** 타입의 *playing_date*라는 이름의 속성을 추가하면, 클래스 B의 *playing_date* 속성과 충돌이 발생할 것이다. 이것이 사례 4의 경우다. 이 충돌을 해결하는 정확한 방법은 사용자가 클래스 B가 클래스 A의 *playing_date* 속성을 상속하도록 명시하는 것이다. 메서드가 속성을 참조한다면, 클래스 B의 사용자는 올바른 *playing_date* 속성을 참조하도록 메서드를 적절히 변경할 필요가 있다. 클래스 A의 스키마 변경이 허용되지 않는 이유는 클래스 B의 사용자가 스키마 변경으로 인해 발행하는 충돌을 해결하기 위해 명시적인 구문을 기술하지 않으면, 스키마가 일관되지 않은 상태가 되기 때문이다.



- **규칙 5:** 수퍼클래스의 스키마를 변경함으로써 충돌이 발생하면, 그 변경이 규칙들을 위반하지 않는 한 원래의 해결 방법이 유지된다. 그러나 스키마 변경이 원래의 해결 방법을 무효화한다면 시스템은 다른 해결 방법을 적용할 것이다.

규칙 5는 충돌이 없는 클래스에 충돌을 유발하거나, 앞의 충돌을 해결하는 방법을 무효화하는 상황을 책임지는 규칙이다.

이러한 경우는 수퍼클래스에 속성(메서드)이 추가되거나 수퍼클래스로부터 상속받은 속성(메서드)이 삭제될 때, 속성(메서드)의 이름 또는 도메인이 변경되거나, 수퍼클래스가 삭제되는 상황이다. 규칙 5는 규칙 4의 철학과 일치한다. 즉, 사용자는 그 클래스를 상속한 서브클래스가 상속받은 속성(메서드)에 어떠한 영향을 미칠지 신경 쓰지 않고 자유롭게 클래스를 변경할 수 있다.

클래스 *C*의 수퍼클래스의 스키마를 변경할 때, 예전에 다른 클래스와 충돌이 발생하여 그 클래스의 속성을 상속하기로 결정했다면 클래스 *C*의 속성(메서드) 손실을 초래할 수 있다. 이 경우, 예전에 충돌했던 속성(메서드) 중 하나를 대신 상속 받아야 한다.

수퍼클래스의 스키마 변경은 속성(메서드)과 클래스 *C*의 (지역적으로 선언되거나 상속받은) 속성(메서드)의 충돌을 일으킬 수 있다. 이 경우, 시스템은 규칙 2나 규칙 3을 적용하여 충돌을 자동으로 해결하고 사용자에게 알릴 수도 있다.

수퍼클래스와의 관계를 추가하거나 삭제함으로써 새로운 충돌이 발생하는 상황은 규칙 5를 적용할 수 없다. 클래스에 대한 수퍼클래스 추가/삭제는 클래스 내에서 제어되어야 한다. 즉, 사용자가 명시적인 해결 방법을 제시해야 한다.

- **규칙 6:** 속성이나 메서드의 변경은 충돌이 발생하지 않는 서브클래스들에게만 전파된다.

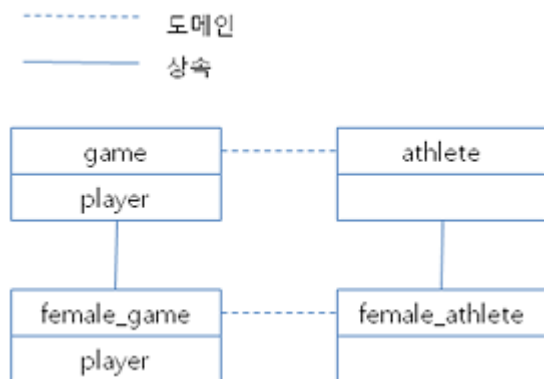
이 규칙은 규칙 5와 상속 불변성의 적용을 제한한다. 규칙 2, 규칙 3을 적용하여 충돌을 탐지하고 해결할 수 있다.

- **규칙 7:** 클래스 *R*의 속성이 클래스 *C*를 도메인으로 사용해도 클래스 *C*를 삭제할 수 있다. 이 경우, 클래스 *C*를 도메인으로 사용하는 속성의 도메인이 object로 변경될 수 있다.

도메인 변경 규칙

- **규칙 8:** 클래스 *C*의 한 속성의 도메인이 *D*에서 *D*의 수퍼클래스로 변경되었다면 새로운 도메인은 클래스 *C*가 속성을 상속받은 수퍼클래스의 대응하는 도메인보다 더 일반적이지 않다. 다음 예는 이 규칙의 원리를 설명한다.

데이터베이스에 *player*라는 속성을 가지는 *game* 클래스와 *game*을 상속한 *female_game* 클래스가 존재한다고 가정하자. *game*의 *player* 속성의 도메인은 *athlete* 클래스이지만 *female_game*의 *player* 속성의 도메인은 *athlete*의 서브클래스인 *female_athlete* 클래스로 변경되었다. 다음 그림이 이러한 관계를 보여주고 있다. 그러나 *female_game*의 *player* 속성의 도메인은 *female_athlete*의 수퍼클래스인 *athlete*로 다시 변경될 수 있다.



클래스 계층 규칙

- **규칙 9:** 수퍼클래스가 없는 클래스는 object의 직속 서브클래스가 된다. 클래스 계층 규칙은 수퍼클래스가 없는 클래스의 특성을 정의한다. 수퍼클래스 없이 클래스를 생성한다면 object를 수퍼클래스가 갖게 된다. 만약 클래스 C의 고유한 수퍼클래스인 S를 삭제하면 클래스 C는 object의 직속 서브클래스가 된다.

데이터베이스 관리

사용자 관리

데이터베이스 사용자

사용자 이름 작성 원칙은 **식별자** 절을 참고한다.

CUBRID는 기본적으로 **DBA, PUBLIC, INFORMATION_SCHEMA** 세 가지 내장 사용자를 제공한다. 제품을 처음 설치하면 **DBA**와 **PUBLIC** 사용자의 비밀번호가 설정되어 있지 않다.

- 데이터베이스의 모든 사용자는 **PUBLIC**의 멤버가 되므로, **PUBLIC** 사용자에게 부여된 권한을 소유한다. 모든 사용자에게 권한을 부여하려면 **PUBLIC** 사용자에게 권한을 부여하면 된다.
- **DBA**는 데이터베이스 관리자 권한을 가진 사용자로, 자동으로 모든 사용자와 그룹의 멤버가 된다. 이로 인해 **DBA**는 모든 테이블에 접근할 수 있으며, 별도로 권한을 부여할 필요가 없다.
- **INFORMATION_SCHEMA**는 표준 메타데이터 뷰([Information Schema](#) 참조)를 소유하는 사용자로, 해당 목적으로만 예약되어 있으며 로그인 불가하다.

데이터베이스 사용자는 고유한 이름을 갖으며, 관리자는 **cubrid createdb** 유틸리티를 사용하여 사용자를 일괄적으로 생성할 수 있다(자세한 내용은 [cubrid 유틸리티](#) 참조).

사용자는 동일한 권한을 가지는 다른 멤버를 직접 소유할 수 없지만, 한 사용자에게 권한이 부여되면 해당 사용자의 모든 멤버가 동일한 권한을 자동으로 상속 받는다.

참고

CUBRID 11.3 버전부터 **DBA**의 멤버도 시스템 카탈로그 가상 클래스에서 모든 사용자의 객체를 조회할 수 있다.

CREATE USER

CREATE USER 문을 사용하여 사용자를 생성할 수 있다. 기본으로 제공되는 **DBA**, **PUBLIC** 사용자는 비밀번호가 없이 생성된다.

```
CREATE USER user_name
[PASSWORD password]
[LOGIN | NOLOGIN]
[GROUPS user_name [{, user_name } ... ]]
[MEMBERS user_name [{, user_name } ... ]]
[COMMENT 'comment_string'];
```

- *user_name*: 생성할 사용자 이름을 지정한다.
- *password*: 생성할 사용자의 비밀번호를 지정한다.
- **[LOGIN | NOLOGIN]**: 생성할 사용자의 로그인 허용 여부를 지정한다. 지정하지 않으면 기본값은 **LOGIN** 이 된다. 자세한 내용은 **사용자의 로그인 허용 여부** 를 참고한다.
- *comment_string*: 생성할 사용자에 대한 커멘트를 지정한다.

참고

DBA 와 **DBA** 의 멤버만 CREATE USER 문을 사용하여 사용자를 생성할 수 있다.

다음은 test_user1 사용자를 생성하고 비밀번호를 설정하는 예제이다.

```
CREATE USER test_user1 PASSWORD 'password';
```

다음은 사용자 생성과 함께 멤버를 추가하는 예제이다. 다음 문장을 통해 *company* 는 *engineering*, *marketing*, *design* 을 멤버로 가지는 그룹이 된다. *marketing* 은 *smith*, *jones* 를, *design* 은 *smith* 를, *engineering* 은 *brown* 을 멤버로 가지는 그룹이 된다.

```
CREATE USER company;
CREATE USER engineering GROUPS company;
CREATE USER marketing GROUPS company;
CREATE USER design GROUPS company;
CREATE USER smith GROUPS design, marketing;
CREATE USER jones GROUPS marketing;
CREATE USER brown GROUPS engineering;
```

다음은 위와 동일한 그룹을 생성하는 예이지만 **GROUPS** 대신 **MEMBERS** 문을 사용하는 예제이다.

```
CREATE USER smith;
CREATE USER brown;
CREATE USER jones;
CREATE USER engineering MEMBERS brown;
CREATE USER marketing MEMBERS smith, jones;
CREATE USER design MEMBERS smith;
CREATE USER company MEMBERS engineering, marketing, design;
```

사용자의 커멘트

다음은 test_user1 사용자를 생성하고 비밀번호와 커멘트를 설정하는 예제이다.

```
CREATE USER test_user1 PASSWORD 'password' COMMENT 'new user';
```

사용자의 커멘트를 확인하려면 다음의 구문을 실행한다.

```
SELECT name, comment FROM db_user;
```

사용자의 커멘트 변경은 ALTER USER 문의 설명을 참고한다.

ALTER USER

ALTER USER 문을 사용하여 사용자의 비밀번호, 멤버, 로그인 허용 여부 및 커멘트를 변경할 수 있다.

```
ALTER USER user_name
[PASSWORD password] [LOGIN | NOLOGIN] |
[ADD MEMBERS user_name [{, user_name } ... ]] |
[DROP MEMBERS user_name [{, user_name } ... ]]
[COMMENT 'comment_string'];
```

- *user_name*: 변경할 사용자 이름을 지정한다.
- *password*: 변경할 사용자의 비밀번호를 지정한다.
- **[LOGIN | NOLOGIN]**: 변경할 사용자의 로그인 허용 여부를 지정한다. 자세한 내용은 **사용자의 로그인 허용 여부** 를 참고한다.
- *comment_string*: 변경할 사용자에 대한 커멘트를 지정한다.

참고

DBA 와 **DBA** 의 멤버는 ALTER USER 문을 사용하여 모든 사용자 의 비밀번호, 멤버 및 커멘트를 변경할 수 있다.

일반 사용자 는 ALTER USER 문을 사용하여 본인 의 비밀번호, 멤버 및 커멘트를 변경할 수 있다.

다음은 test_user1 사용자를 생성하고 비밀번호를 변경하는 예제이다.

```
CREATE USER test_user1;  
ALTER USER test_user1 PASSWORD '1234';
```

다음은 사용자를 생성하고 생성된 사용자에게 멤버를 추가하는 예제이다. CREATE USER .. MEMBERS .. 의 예제와 동일하게 수행하는 예제이다.

```
CREATE USER company;  
CREATE USER engineering;  
CREATE USER marketing;  
CREATE USER design;  
CREATE USER smith;  
CREATE USER jones;  
CREATE USER brown;  
  
ALTER USER engineering ADD MEMBERS brown;  
ALTER USER marketing ADD MEMBERS smith, jones;  
ALTER USER design ADD MEMBERS smith;  
ALTER USER company ADD MEMBERS engineering, marketing, design;
```

다음은 생성된 사용자 그룹의 멤버를 삭제하는 예제이다. 다음 문장을 통해 *company* 그룹에서 *marketing* 멤버를 삭제하고, *marketing* 그룹은 *smith, jones* 를 멤버에서 삭제한다.

```
ALTER USER company DROP MEMBERS marketing;  
ALTER USER marketing DROP MEMBERS smith, jones;
```

사용자의 로그인 허용 여부

다음은 로그인을 허용하지 않는 사용자를 생성하는 예제이다.

```
CREATE USER test_user1 PASSWORD 'password' NOLOGIN;
```

test_user1 사용자로 데이터베이스에 접속하면 다음과 같은 에러가 발생한다.

```
ERROR: Login is disabled for user "test_user1".
```

다음은 test_user1 사용자의 로그인을 다시 허용하는 예제이다.

```
ALTER USER test_user1 LOGIN;
```

사용자의 로그인 허용 여부를 확인하려면 다음의 구문을 실행한다.

```
SELECT name, is_loginable FROM db_user;
```

다음은 비밀번호와 로그인 허용 여부, 커멘트를 한 문장으로 함께 변경하는 예제이다. **ADD MEMBERS, DROP MEMBERS**와는 함께 사용할 수 없다.

```
ALTER USER test_user1 PASSWORD '1234' NOLOGIN COMMENT 'blocked temporarily';
```

참고

로그인 허용 여부는 로그인할 때 확인한다. 따라서 사용자를 **NOLOGIN**으로 변경해도 그 사용자의 기존 연결은 끊어지지 않으며, 이후의 신규 접속과 **login () 메서드**를 통한 사용자 전환만 거부된다.

참고

DBA와 **DBA**의 멤버만 사용자의 로그인 허용 여부를 변경할 수 있다.

단, **DBA**, **INFORMATION_SCHEMA**, 현재 사용자는 변경 대상이 될 수 없으며, 대상으로 지정하면 에러가 발생한다.

경고

PUBLIC을 **NOLOGIN**으로 변경하면 **PUBLIC** 사용자의 접속이 거부된다. 사용자 이름을 지정하지 않은 접속도 **PUBLIC**으로 처리되므로 함께 거부된다.

로그인 허용 여부는 로그인만 제어한다. 권한과 소유권은 영향받지 않으므로, **PUBLIC**이 **NOLOGIN**인 상태에서도 **GRANT** 문으로 부여한 권한과 그룹을 통한 권한 상속은 그대로 동작한다.

사용자의 커멘트 변경

다음은 생성된 사용자의 커멘트를 변경하는 예제이다.

```
CREATE USER test_user1 COMMENT 'new user';
ALTER USER test_user1 COMMENT 'old user';
```

DROP USER

DROP USER 문을 사용하여 사용자를 삭제할 수 있다. 테이블, 뷰, 트리거, 저장 함수/프로시저, 시리얼, 동의어, 서버의 객체를 소유한 사용자는 삭제할 수 없다.

```
DROP USER user_name;
```

· *user_name*: 삭제할 사용자 이름을 지정한다.

참고

DBA 와 **DBA** 의 멤버만 DROP USER 문을 사용하여 사용자를 삭제할 수 있다.

다음은 test_user1 사용자를 생성하고 비밀번호 변경 후 test_user1 사용자를 삭제하는 예제이다.

```
CREATE USER test_user1;
ALTER USER test_user1 PASSWORD '1234';
DROP USER test_user1;
```

GRANT

CUBRID에서는 다음 데이터베이스 객체에 대한 권한을 부여할 수 있다.

- 테이블
- 뷰

· 프로시저

자신이 생성한 데이터베이스 객체에 다른 사용자(그룹)의 접근을 허용하려면 해당 사용자(그룹)에게 적절한 권한을 부여해야 한다. 권한이 부여된 그룹의 모든 멤버는 동일한 권한을 소유하므로 모든 멤버에게 개별적으로 권한을 부여할 필요는 없다. 단, DBA와 소유자 그룹의 멤버가 아닌 WITH GRANT OPTION 권한을 부여받은 그룹의 멤버는 부여받은 권한을 다른 사용자에게 부여할 수 없다.

PUBLIC 사용자가 생성한 데이터베이스 객체는 모든 사용자에게 접근이 허용된다.

다음의 **GRANT** 문을 사용하여 사용자에게 접근 권한을 부여할 수 있다.

(1) 테이블과 뷰:

```
GRANT operation [ { ,operation } ... ] ON
[schema_name.]object_name [ { , [schema_name.]object_name } ... ]
TO user [ { ,user } ... ] [ WITH GRANT OPTION ];
```

(2) 저장 프로시저와 저장 함수:

```
GRANT EXECUTE ON PROCEDURE [schema_name.]object_name
TO user [ { ,user } ... ];
```

• *operation*: 권한을 부여할 때 각 데이터베이스 객체에 대하여 사용 가능한 연산을 나타낸다.

◦ (1) 테이블과 뷰

- **SELECT**: 테이블 정의 내용을 읽을 수 있고 인스턴스 조회가 가능. 가장 일반적인 유형의 권한.
- **INSERT**: 테이블의 인스턴스를 생성할 수 있는 권한.
- **UPDATE**: 테이블에 이미 존재하는 인스턴스를 수정할 수 있는 권한.
- **DELETE**: 테이블의 인스턴스를 삭제할 수 있는 권한.
- **ALTER**: 테이블의 정의를 수정할 수 있고, 테이블의 이름을 변경하거나 삭제할 수 있는 권한.
- **INDEX**: 검색 속도의 향상을 위해 칼럼에 인덱스를 생성할 수 있는 권한.
- **EXECUTE**: 테이블 메서드 혹은 인스턴스 메서드를 호출할 수 있는 권한.
- **ALL PRIVILEGES**: 앞서 설명한 7가지 권한을 모두 포함.

◦ (2) 저장 프로시저와 저장 함수

- **EXECUTE ON PROCEDURE**: 저장 프로시저 또는 저장 함수를 호출할 수 있는 권한.

• *schema_name*: 데이터베이스 객체의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.

- *object_name*: 권한을 부여할 데이터베이스 객체의 이름을 지정한다.
- *user*: 권한을 부여할 사용자나 그룹의 이름을 지정한다. 데이터베이스 사용자의 로그인 이름을 입력하거나 시스템 정의 사용자인 **PUBLIC** 을 입력할 수 있다. **PUBLIC** 이 명시되면 데이터베이스의 모든 사용자는 부여한 권한을 가진다.

• WITH GRANT OPTION

- **WITH GRANT OPTION** 을 이용하면 권한을 부여받은 사용자가 부여받은 권한을 다른 사용자에게 부여할 수 있다.
- 저장 프로시저와 저장 함수에 대한 **EXECUTE ON PROCEDURE** 권한은 **WITH GRANT OPTION** 을 지원하지 않는다.

다음은 *smith* (*smith* 의 모든 멤버 포함)에게 *olympic* 테이블의 검색 권한을 부여한 예제이다.

```
GRANT SELECT ON olympic TO smith;
```

다음은 *brown* 와 *jones* (두 사용자에게 속한 모든 멤버)에게 *nation* 과 *athlete* 테이블에 대해 **SELECT, INSERT, UPDATE, DELETE** 권한을 부여한 예제이다.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON nation, athlete TO brown, jones;
```

다음은 모든 사용자(public)에게 *tbl1*, *tbl2* 테이블에 대해 모든 권한을 부여하는 예제이다.

```
CREATE TABLE tbl1 (a INT);
CREATE TABLE tbl2 (a INT);
GRANT ALL PRIVILEGES ON tbl1, tbl2 TO public;
```

다음 **GRANT** 문은 *brown*에게 *record*, *history* 테이블에 대한 검색 권한을 부여하고 *brown*이 다른 사용자에게 검색 권한을 부여할 수 있도록 **WITH GRANT OPTION** 절을 사용한 예제이다. 이후 *brown*은 다른 사용자에게 자신이 받은 권한 내에서 권한을 부여할 수 있다.

```
GRANT SELECT ON record, history TO brown WITH GRANT OPTION;
```

다음은 *DBA* 가 일반 사용자 *u1* 이 생성한 *tbl3* 테이블에 대하여 사용자 *u2* 에게 **SELECT** 권한을 부여한 예제이다. *db_auth*의 *grantor_name* (권한을 부여한 사용자명) 을 조회하면 권한을 부여한 *DBA* 가 아닌, 소유자(*u1*)이 표시된다.

```
CALL LOGIN ('DBA', '') ON CLASS db_user;
CREATE USER u1;
CREATE USER u2;
CREATE TABLE u1.tbl3 (a INT);
```

```
GRANT SELECT ON u1.tbl3 TO u2;

SELECT * FROM db_auth WHERE object_name = 'tbl3';
```

grantor_name	grantee_name	object_type
object_name	owner_name	auth_type
is_grantable		
=====		
=====		
=====		
'U1'	'U2'	'CLASS'
'tbl3'	'U1'	'SELECT'
'NO'		

참고

- 권한을 부여하는 사용자는 권한 부여 전에 나열된 모든 테이블의 소유자, 소유자 멤버, DBA, DBA 멤버 및 **WITH GRANT OPTION** 을 가지고 있어야 한다.
- DBA, DBA 멤버, 소유자 멤버는 권한을 부여하는 경우, *grantor_name* 컬럼에 권한을 부여한 사용자가 아닌 객체 소유자로 표시된다. 단, **WITH GRANT OPTION**을 가진 사용자가 권한을 부여하는 경우, **grantor_name** 컬럼에 권한을 부여한 사용자로 표시된다.
- 뷰에 대한 **SELECT, UPDATE, DELETE, INSERT** 권한을 부여하기 전에 뷰의 소유자는 뷰의 질의 명세부에 포함되어 있는 모든 테이블에 대해서 **SELECT** 권한과 **GRANT** 권한을 가져야 한다. **DBA** 사용자와 **DBA** 그룹에 속한 멤버는 자동으로 모든 테이블에 대한 모든 권한을 가진다.
- **TRUNCATE** 문을 수행하려면 **ALTER, INDEX, DELETE** 권한이 필요하다.

다음은 *smith* 에게 *my_sp* 프로시저를 실행할 수 있는 권한을 부여하는 예제이다.

```
CREATE OR REPLACE PROCEDURE my_sp ()
IS
BEGIN
    DBMS_OUTPUT.put_line('grant test');
END;
GRANT EXECUTE ON PROCEDURE my_sp TO smith;
```

참고

- 저장 프로시저와 함수에 대하여 **GRANT** 구문으로 한번에 하나의 객체에 대한 권한만 부여할 수 있다.

- 저장 프로시저와 함수에 대하여 모두 **EXECUTE ON PROCEDURE** 연산을 사용한다.
- 현재 저장 프로시저와 함수에 대해서 **WITH GRANT OPTION** 을 이용한 권한 부여 기능에서 지원하지 않으므로, 사용시 문법 오류를 반환한다.

REVOKE

REVOKE 문을 사용하여 권한을 해지할 수 있다. 사용자에게 부여된 권한은 언제든지 해지가 가능하다. 한 사용자에게 두 종류 이상의 권한을 부여했다면 권한 중 일부 또는 전부를 해지할 수 있다. 또한 하나의 **GRANT** 문으로 여러 사용자에게 여러 데이터베이스 객체에 대한 권한을 부여한 경우라도 일부 사용자와 일부 데이터베이스 객체에 대해 선택적인 권한 해지가 가능하다.

권한을 부여한 사용자에게서 권한(**WITH GRANT OPTION**)을 해지하면, 권한이 해지된 사용자로부터 권한을 받은 사용자도 권한이 해지된다.

다음의 **REVOKE** 문을 사용하여 사용자에게 부여된 권한을 해지할 수 있다.

```
(1) 테이블과 뷰:
    REVOKE operation [ { ,operation } ... ] ON
    [schema_name.]object_name [ { , [schema_name.]object_name } ... ]
    FROM user [ { ,user } ... ];

(2) 저장 프로시저와 저장 함수:
    REVOKE EXECUTE ON PROCEDURE [schema_name.]object_name
    FROM user [ { ,user } ... ];
```

- *operation*: 권한을 해지할 때 사용 가능한 연산의 종류이다(자세한 내용은 **GRANT** 참조).
- *schema_name*: 데이터베이스 객체의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *object_name*: 권한을 해지할 데이터베이스 객체의 이름을 지정한다.
- *user*: 권한을 해지할 사용자나 그룹의 이름을 지정한다.

다음은 *smith, jones* 사용자에게 *nation, athlete* 두 테이블에 대해 **SELECT, INSERT, UPDATE, DELETE** 권한을 부여하는 예제이다.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON nation, athlete TO smith,
jones;
```

다음은 *jones* 에게 조회 권한만을 부여하기 위해 **REVOKE** 문장을 수행하는 예제이다. 만약 *jones* 가 다른 사용자에게 권한을 부여했다면 권한받은 사용자 또한 조회만 가능하다.

```
REVOKE INSERT, UPDATE, DELETE ON nation, athlete FROM jones;
```

다음은 *smith* 에게 부여한 모든 권한을 해지하기 위해 **REVOKE** 문을 수행하는 예제이다. 이 문장이 수행되면 *smith* 는 *nation*, *athlete* 테이블에 대한 어떠한 연산도 허용되지 않는다.

```
REVOKE ALL PRIVILEGES ON nation, athlete FROM smith;
```

다음은 *u1* 사용자가 *tbl1**에 대해 **u2* 사용자에게 **WITH GRANT OPTION** 과 함께 **SELECT** 권한을 부여하고, *u2* 사용자는 *u3* 사용자에게 *tbl1* 테이블의 **SELECT** 권한 부여한 후, *DBA* 가 *u1* 사용자의 *tbl1* 에 부여된 *u2* 의 권한을 해지하는 예시이다. *tbl1* 에 대한 *u2* 의 권한 해지시, *u2* 가 **WITH GRANT OPTION** 을 이용해서 부여한 권한에 대해서도 함께 해지된다.

```
CREATE USER u1;
CREATE USER u2;
CREATE USER u3;
CREATE TABLE u1.tbl1 (a INT);

CALL LOGIN ('u1', '') ON CLASS db_user;
GRANT SELECT ON u1.tbl1 TO u2 WITH GRANT OPTION;

CALL LOGIN ('u2', '') ON CLASS db_user;
GRANT SELECT ON u1.tbl1 TO u3 WITH GRANT OPTION;

CALL LOGIN ('dba', '') ON CLASS db_user;
SELECT * FROM db_auth WHERE object_name = 'tbl1';
```

grantor_name	grantee_name	object_type
object_name	owner_name	auth_type
is_grantable		
=====		
=====		
=====		
'U1'	'U2'	'CLASS'
'tbl1'	'U1'	'SELECT'
'YES'		
'U2'	'U3'	'CLASS'
'tbl1'	'U1'	'SELECT'
'YES'		

```
REVOKE SELECT ON u1.tbl1 FROM u2;
```

```
SELECT * FROM db_auth WHERE object_name = 'tbl1';
```

```
There are no results.
0 row selected.
```

참고

- DBA, DBA 멤버, 소유자 멤버는 권한을 해지할 때, 소유자와 동일하게 권한을 해지할 수 있다.

ALTER ... OWNER

데이터베이스 관리자(**DBA**) 또는 **DBA** 그룹의 멤버는 다음 질의를 통해 테이블, 뷰, 트리거, 저장 함수/프로시저 및 시리얼의 소유자를 변경할 수 있다.

```
ALTER (TABLE | CLASS | VIEW | VCLASS | TRIGGER | PROCEDURE |
FUNCTION | SERIAL) [schema_name.]name OWNER TO user_id;
```

- *schema_name*: 객체의 스키마 이름을 지정한다. 생략하면 현재 세션의 스키마 이름을 사용한다.
- *name*: 소유자를 변경할 스키마 객체의 이름
- *user_id*: 사용자 ID

```
ALTER TABLE test_tbl OWNER TO public;
ALTER VIEW test_view OWNER TO public;
ALTER TRIGGER test_trigger OWNER TO public;
ALTER FUNCTION test_function OWNER TO public;
ALTER PROCEDURE test_procedure OWNER TO public;
ALTER SERIAL test_serial OWNER TO public;
```

경고

소유자 변경 시, 이전 소유자가 부여한 권한은 변경된 소유자에게 이관된다. 단, 이전 소유자가 변경된 소유자에게 부여했던 해당 객체의 권한은 자동으로 해지된다.

사용자 권한 관리 메서드

데이터베이스 관리자(**DBA**)는 데이터베이스 사용자에게 대한 정보를 저장하는 **db_user** 또는 시스템 권한 클래스인 **db_root**에 정의된 권한 관련 메서드들을 호출하여 사용자 권한을 조회 및 수정할 수

있다. 호출하고자 하는 메서드에 따라 **db_user** 또는 **db_root** 클래스를 명시할 수 있으며, 메서드의 리턴 값을 변수에 저장할 수 있다. 또한, 일부 메서드는 **DBA** 와 **DBA** 그룹의 멤버에 의해서만 호출될 수 있음에 유의한다.

참고

HA 환경에서 마스터 노드에서의 메서드 호출은 슬레이브 노드에 반영되지 않으므로 이에 주의한다.

```
CALL method_definition ON CLASS auth_class [ TO variable ] [ ; ]
CALL method_definition ON variable [ ; ]
```

login() 메서드

login () 메서드는 **db_user** 클래스의 클래스 메서드로서, 현재 데이터베이스에 접속한 사용자를 변경하고자 할 때 사용된다. 새로 접속할 사용자 이름과 비밀번호가 인자로 주어지며, 문자열 타입이어야 한다. 비밀번호가 없는 경우 인자에 공백 문자(" ")를 입력할 수 있다. **DBA** 나 **DBA** 그룹의 멤버는 비밀번호를 입력하지 않고 **login ()** 메서드를 호출할 수 있다.

```
-- 비밀번호가 없는 DBA 사용자로 접속하기
CALL login ('dba', '') ON CLASS db_user;

-- 비밀번호가 cubrid인 user_1 사용자로 접속하기
CALL login ('user_1', 'cubrid') ON CLASS db_user;
```

add_user() 메서드

add_user () 메서드는 **db_user** 클래스의 클래스 메서드로서, 새로운 사용자를 추가할 때 사용된다. 새로 추가할 사용자 이름과 비밀번호가 인자로 주어지며, 문자열 타입이어야 한다. 이때, 추가할 사용자 이름은 이미 등록된 데이터베이스 사용자 이름과 중복되어서는 안 된다. 한편, **add_user ()** 메서드는 **DBA** 사용자와 **DBA** 그룹에 속한 멤버만 호출할 수 있다.

```
-- 비밀번호가 없는 user_2 추가하기
CALL add_user ('user_2', '') ON CLASS db_user;

-- 비밀번호가 없는 user_3 추가하고, 메서드 리턴 값을 admin 변수에 저장하기
CALL add_user ('user_3', '') ON CLASS db_user to admin;
```

drop_user() 메서드

drop_user () 메서드는 **db_user** 클래스의 클래스 메서드로서, 기존 사용자를 삭제할 때 사용된다. 삭제하고자 하는 사용자 이름만 인자로 주어지며, 문자열 타입이어야 한다. 이때, 클래스의 소유자는

삭제할 수 없으므로, **DBA** 는 관련 클래스의 소유자를 변경한 후, 해당 사용자를 삭제할 수 있다. **drop_user()** 메서드 역시 **DBA** 사용자와 **DBA** 그룹에 속한 멤버만 호출할 수 있다.

```
-- user_2 삭제하기
CALL drop_user ('user_2') ON CLASS db_user;
```

find_user() 메서드

find_user() 메서드는 **db_user** 클래스의 클래스 메서드로서, 인자로 주어진 사용자를 검색할 때 사용된다. 찾고자 하는 사용자 이름이 인자로 주어지며, **TO** 뒤에 지정된 변수에 메서드의 리턴 값을 저장하여 다음 질의 수행 시 변수에 저장된 값을 이용할 수 있다.

```
-- user_2를 찾아서 admin이라는 변수에 저장하기
CALL find_user ('user_2') ON CLASS db_user TO admin;
```

set_password() 메서드

set_password() 메서드는 사용자 인스턴스 각각에 대해 호출할 수 있는 인스턴스 메서드로서, 사용자의 비밀번호를 변경할 때 사용된다. 지정된 사용자의 새로운 비밀번호가 인자로 주어진다. **DBA** 와 **DBA** 그룹의 멤버를 제외한 일반 사용자는 자신의 비밀번호만 변경할 수 있다.

```
-- user_4 를 추가하고 user_common 변수에 저장하기
CALL add_user ('user_4', '') ON CLASS db_user to user_common;

-- user_4의 비밀번호를 'abcdef'로 변경하기
CALL set_password('abcdef') on user_common;
```

change_owner() 메서드

change_owner() 메서드는 **db_root** 클래스의 클래스 메서드로서, 클래스 소유자를 변경할 때 사용된다. 소유자를 변경하고자 하는 클래스 이름과 새로운 소유자의 이름이 각각 인자로 주어진다. 테이블 이름은 스키마 이름을 접두사로 사용해야 한다. 생략하면 현재 세션의 스키마 이름을 사용한다. 이때, 데이터베이스에 존재하는 클래스와 소유자가 인자로 지정되어야 하며, 그렇지 않은 경우 에러가 발생한다. **change_owner()** 메서드는 **DBA** 와 **DBA** 그룹의 멤버만 호출할 수 있다. 이 메서드와 같은 역할을 하는 질의로 **ALTER ... OWNER** 가 있다. 이에 대한 내용은 **ALTER ... OWNER** 절을 참고한다.

```
-- user_10이 소유하고 있는 table_1의 소유자를 user_4로 변경하기
CALL change_owner ('user_1.table_1', 'user_4') ON CLASS db_root;
```

다음 예제는 특정 데이터베이스 사용자의 존재 여부를 판단하기 위해 시스템 클래스인 **db_user** 에 등록된 메서드인 **find_user** 를 호출하는 **CALL** 문의 수행을 보여준다. 첫 번째 문장은 **db_user** 클래스에 정의된 클래스 메서드를 호출한다. 찾고자 하는 대상 사용자가 데이터베이스에 등록되어 있을 경우 x에는 해당 클래스 이름(여기에서는 **db_user**)이 저장되고, 없을 경우엔 **NULL** 이 저장된다.

두 번째 문장은 변수 `x`에 저장된 값을 출력하는 방법이다. 이 질의문에서 **db_root**는 시스템 클래스로서, 하나의 인스턴스만이 존재하여 `sys_date`나 등록된 변수의 값을 출력하는 데 사용할 수 있다. 이러한 용도로 쓰일 경우 **db_root**는 인스턴스가 하나인 다른 테이블로 대체할 수 있다.

```
CALL find_user('dba') ON CLASS db_user to x;
```

```
Result
=====
db_user
```

```
SELECT x FROM db_root;
```

```
x
=====
db_user
```

find_user 메서드를 이용하면 결과값이 **NULL** 인지 아닌지에 따라 해당 사용자가 데이터베이스에 존재하는지 여부를 판단할 수 있다.

SET

SET 문은 시스템 파라미터의 값을 지정하거나 사용자 정의 변수의 값을 지정하는 구문이다.

시스템 파라미터

SQL 문을 이용하여 CSQL 인터프리터나 CUBRID 매니저의 질의 편집기에서 시스템 파라미터의 값을 설정할 수 있다. 단, 갱신할 수 있는 파라미터는 한정되어 있으므로 주의한다. 갱신할 수 있는 파라미터를 확인하려면 [cubrid.conf 설정 파일과 기본 제공 파라미터](#)를 참고한다.

```
SET SYSTEM PARAMETERS 'parameter_name=value [{; name=value}...]'
```

value: **call_stack_dump_activation_list** 파라미터를 제외하고 해당 파라미터의 *value*를 **DEFAULT**로 설정하면 기본값으로 재설정된다.

```
SET SYSTEM PARAMETERS 'lock_timeout=DEFAULT';
```

사용자 변수

사용자 정의 변수는 다음 두 가지 방법으로 생성할 수 있다. 하나는 **SET** 문을 사용하는 것이고, 다른 하나는 SQL문 내에서 사용자 정의 변수에 값을 할당하는 방식입니다.

정의된 사용자 정의 변수는 **DEALLOCATE** 또는 **DROP VARIABLE** 구문을 사용하여 삭제할 수 있다.

사용자 정의 변수는 하나의 응용 프로그램이 연결을 유지하는 동안 사용되는 변수이므로 세션 변수라고도 한다. 이 변수는 연결 세션 범위 내에서만 유효하며, 하나의 응용 프로그램에서 정의된 변수는 다른 응용 프로그램에서 접근할 수 없다.

응용 프로그램이 연결을 종료하면, 모든 변수는 자동으로 제거된다.

사용자 정의 변수는 세션당 최대 20개까지 정의할 수 있으며, 이 제한을 초과하여 새 변수를 생성하려면 **DROP VARIABLE** 문을 사용해 기존의 사용하지 않는 변수를 먼저 제거해야 한다.

대부분의 SQL 문에서 사용자 정의 변수를 사용할 수 있지만, SELECT 문 내에서 변수의 지정과 참조 순서는 보장되지 않는다. 예를 들어, SELECT 목록에 지정된 사용자 정의 변수를 **HAVING**, **GROUP BY**, **ORDER BY** 절에서 참조하면 예상한 값이 반환되지 않을 수 있다. 또한, 사용자 정의 변수는 컬럼 이름이나 테이블 이름과 같은 식별자(identifier)로는 사용할 수 없다.

사용자 정의 변수는 대소문자를 구분하지 않는다. 사용자 정의 변수의 타입은 **SHORT**, **INTEGER**, **BIGINT**, **FLOAT**, **DOUBLE**, **NUMERIC**, **CHAR**, **VARCHAR**, **BIT**, **BIT VARYING** 중 하나가 될 수 있으며, 그 밖의 타입은 **VARCHAR** 타입으로 변환된다.

```
SET @v1 = 1, @v2=CAST(1 AS BIGINT), @v3 = '123', @v4 =
DATE'2010-01-01';

SELECT typeof(@v1), typeof(@v2), typeof(@v3), typeof(@v4);
```

```
      typeof(@v1)      typeof(@v2)      typeof(@v3)
typeof(@v4)
=====
'integer'             'bigint'             'character (-1)'
'character varying (1073741823)
```

사용자 정의 변수의 타입은 사용자가 값을 지정할 때 변경될 수 있다.

```
SET @v = 'a';
SET @v1 = 10;

SELECT @v := 1, typeof(@v1), @v1:='1', typeof(@v1);
```

```
@v := 1      typeof(@v1)      @v1 := '1'
typeof(@v1)
```

```
=====
=====
1                'integer'                '1'
'character (-1)'
```

```
<set_statement>
    : <set_statement>, <udf_assignment>
    | SET <udv_assignment>
    ;

<udv_assignment>
    : @<name> = <expression>
    | @<name> := <expression>
    ;

{DEALLOCATE|DROP} VARIABLE <variable_name_list>
<variable_name_list>
    : <variable_name_list> ',' @<name>
```

- 사용자 정의 변수의 이름은 영숫자(alphanumeric)와 언더바(_)로 정의한다.
- SQL 문 내에서 사용자 정의 변수를 선언할 때에는 '=' 연산자를 사용한다.

사용자 정의 변수 a를 선언하고, 값 1을 할당한다.

```
SET @a = 1;
SELECT @a;
```

```
@a
=====
1
```

사용자 정의 변수를 사용하여 **SELECT** 문에서 행의 개수를 카운트한다.

```
CREATE TABLE t (i INTEGER);
INSERT INTO t(i) VALUES(2),(4),(6),(8);

SET @a = 0;

SELECT @a := @a+1 AS row_no, i FROM t;
```

```
row_no                i
=====
1                    2
2                    4
```



```

3                               6
4                               8

4 rows selected.

```

사용자 정의 변수를 prepared statement에서 지정한 바인드 파라미터의 입력으로 사용한다.

```

SET @a:=3;

PREPARE stmt FROM 'SELECT i FROM t WHERE i < ?';
EXECUTE stmt USING @a;

```

```

          i
=====
          2

```

SQL 문 내에서 ':' 연산자를 사용하여 사용자 정의 변수를 선언한다.

```

SELECT @a := 1, @user_defined_variable := 'user defined variable';
UPDATE t SET i = (@var := 1);

```

사용자 정의 변수 *a* 와 *user_defined_variable* 를 삭제한다.

```

DEALLOCATE VARIABLE @a, @user_defined_variable;
DROP VARIABLE @a, @user_defined_variable;

```

참고

SET 문에 의해 정의되는 사용자 정의 변수는 응용 프로그램이 서버에 연결하면서 시작되어 응용 프로그램이 연결을 종료할 때까지 유지되며, 이 기간동안 유지되는 연결을 세션(session)이라고 한다. 사용자 정의 변수는 응용 프로그램이 연결을 종료하거나 일정 기간 동안 요청이 없어 세션 기간이 만료될(expired) 때 삭제된다. 세션 기간은 **cubrid.conf** 의 **session_state_timeout** 파라미터로 설정할 수 있으며, 기본값은 **21600** 초(=6시간)이다.

세션에서 관리하는 데이터에는 **PREPARE** 문 외에 사용자 정의 변수, 가장 마지막에 삽입한 ID(**LAST_INSERT_ID**), 그리고 가장 마지막에 실행한 문장으로 인해 영향을 받은 레코드의 개수(**ROW_COUNT**)가 포함한다.

KILL

KILL 구문은 **TRANSACTION** 또는 **QUERY** 수정자를 사용하여 트랜잭션을 종료한다.

```
KILL [TRANSACTION | QUERY] tran_index, ... ;
```

- **KILL TRANSACTION** 은 **KILL** 질의문의 기본값이다. 수정자가 없는 **KILL** 과 같다. 해당 *tran_index* 와 관련된 트랜잭션을 종료한다.
- **KILL QUERY** 는 트랜잭션에서 수행 중인 질의문을 종료한다.

DBA와 DBA 그룹의 사용자는 시스템의 모든 트랜잭션을 종료할 수 있으며, DBA 외 사용자는 자신의 트랜잭션만 종료할 수 있다.

```
KILL TRANSACTION 1;
```

SHOW

목차

SHOW

1222

● DESC, DESCRIBE	1223
● EXPLAIN	1224
● SHOW TABLES	1224
● SHOW COLUMNS	1225
● SHOW INDEX	1228
● SHOW COLLATION	1231
● SHOW TIMEZONES	1234
● SHOW GRANTS	1236
● SHOW CREATE TABLE	1236

● SHOW CREATE VIEW	1237
● SHOW ACCESS STATUS	1238
● SHOW EXEC STATISTICS	1239
● 진단(Diagnostics)	1241
● SHOW VOLUME HEADER	1241
● SHOW LOG HEADER	1244
● SHOW ARCHIVE LOG HEADER	1249
● SHOW HEAP HEADER	1250
● SHOW HEAP CAPACITY	1256
● SHOW SLOTTED PAGE HEADER	1260
● SHOW SLOTTED PAGE SLOTS	1262
● SHOW INDEX HEADER	1263
● SHOW INDEX CAPACITY	1265
● SHOW CRITICAL SECTIONS	1270
● SHOW TRANSACTION TABLES	1273
● SHOW THREADS	1280
● SHOW JOB QUEUES	1285
● SHOW PAGE BUFFER STATUS	1285

DESC, DESCRIBE

테이블의 칼럼 정보를 출력하며 **SHOW COLUMNS** 문과 같다. 보다 자세한 사항은 **SHOW COLUMNS**를 참고한다.

```
DESC [schema_name.]table_name;
DESCRIBE [schema_name.]table_name;
```

EXPLAIN

테이블의 칼럼 정보를 출력하며 **SHOW COLUMNS** 문과 같다. 보다 자세한 사항은 **SHOW COLUMNS**를 참고한다.

```
EXPLAIN [schema_name.]table_name;
```

SHOW TABLES

데이터베이스의 전체 테이블 이름 목록을 출력한다. 결과 칼럼의 이름은 `tables_in_<데이터베이스 이름>` 이 되며 하나의 칼럼을 지닌다. **LIKE** 절을 사용하면 이와 매칭되는 테이블 이름을 검색할 수 있으며, **WHERE** 절을 사용하면 좀더 일반적인 조건으로 테이블 이름을 검색할 수 있다. **SHOW FULL TABLES** 는 `owner` 칼럼과 `table_type` 칼럼을 함께 출력한다. `owner` 컬럼은 소유자 이름을 값으로 가진다. `table_type` 칼럼은 테이블이면 **BASE TABLE**, 뷰이면 **VIEW**를 값으로 가진다.

```
SHOW [ FULL ] TABLES [ LIKE 'pattern' | WHERE expr ];
```

다음은 이 구문을 수행한 예이다.

```
SHOW TABLES;
```

```
Tables_in_demodb
=====
'athlete'
'code'
'event'
'game'
'history'
'nation'
'olympic'
'participant'
'record'
'stadium'
```

```
SHOW FULL TABLES;
```

Tables_in_demodb	Owner	Table_type
'athlete'	'PUBLIC'	'BASE TABLE'
'code'	'PUBLIC'	'BASE TABLE'
'event'	'PUBLIC'	'BASE TABLE'
'game'	'PUBLIC'	'BASE TABLE'
'history'	'PUBLIC'	'BASE TABLE'
'nation'	'PUBLIC'	'BASE TABLE'
'olympic'	'PUBLIC'	'BASE TABLE'
'participant'	'PUBLIC'	'BASE TABLE'
'record'	'PUBLIC'	'BASE TABLE'
'stadium'	'PUBLIC'	'BASE TABLE'

```
SHOW FULL TABLES LIKE '%c%';
```

Tables_in_demodb	Owner	Table_type
'code'	'PUBLIC'	'BASE TABLE'
'olympic'	'PUBLIC'	'BASE TABLE'
'participant'	'PUBLIC'	'BASE TABLE'
'record'	'PUBLIC'	'BASE TABLE'

```
SHOW FULL TABLES WHERE table_type = 'BASE TABLE' and
TABLES_IN_demodb LIKE '%co%';
```

Tables_in_demodb	Owner	Table_type
'code'	'PUBLIC'	'BASE TABLE'
'record'	'PUBLIC'	'BASE TABLE'

SHOW COLUMNS

테이블의 칼럼 정보를 출력한다. **LIKE** 절을 사용하면 이와 매칭되는 칼럼 이름을 검색할 수 있다. **WHERE** 절을 사용하면 “모든 **SHOW** 문에 대한 일반적인 고려 사항”과 같이 좀 더 일반적인 조건으로 칼럼 이름을 검색할 수 있다.

```
SHOW [FULL] COLUMNS (FROM | IN) [schema_name.]table_name [LIKE
'pattern' | WHERE expr];
```

FULL 키워드를 사용하면 **collation** 및 **comment** 를 추가로 출력한다.

DESCRIBE (또는 줄여서 **DESC**) 문과 **EXPLAIN** 문은 **SHOW COLUMNS**와 같은 정보를 제공하지만, **LIKE** 절 또는 **WHERE** 절은 지원하지 않는다.

해당 구문은 다음과 같은 칼럼을 출력한다.

칼럼 이름	타입	설명
Field	VARCHAR	칼럼 이름
Type	VARCHAR	칼럼의 데이터 타입
Null	VARCHAR	NULL 을 저장할 수 있으면 YES, 불가능하면 NO
Key	VARCHAR	칼럼에 인덱스가 걸려있는지 여부. 테이블의 주어진 칼럼에 하나 이상의 키 값이 존재하면 PRI, UNI, MUL의 순서 중 가장 먼저 나타나는 것 하나만 출력한다. • 공백이면 인덱스를 타지 않거나 다중 칼럼 인덱스에서 첫번째 칼럼이 아니거나, 비고유(non-unique) 인덱스이다. • PRI 값이면 기본 키이거나 다중 칼럼 기본 키이다. • UNI 값이면 고유(unique) 인덱스이다. (고유 인덱스는 여러개의 NULL값을 허용하지만, NOT NULL 제약 조건을 설정할 수도 있다.) • MUL 값이면 주어진 값이 칼럼 내에서 여

칼럼 이름	타입	설명
		러 번 나타나는 것을 허용하는 비고유 인덱스의 첫번째 칼럼이다. 복합 고유 인덱스를 구성하는 칼럼이면 MUL 값이 된다. 칼럼 값들의 결합은 고유일 수 있으나 각 칼럼의 값은 여러 번 나타날 수 있기 때문이다.
Default	VARCHAR	칼럼에 정의된 기본값
Extra	VARCHAR	주어진 칼럼에 대해 가능한 추가 정보. AUTO_INCREMENT 속성인 칼럼은 'auto_increment'라는 값을 갖는다.

다음은 이 구문을 수행한 예이다.

```
SHOW COLUMNS FROM athlete;
```

```

Field      Type      Null
Key        Default   Extra
=====
=====
'code'     'INTEGER' 'NO'
'PRI'      NULL      'auto_increment'
'name'     'VARCHAR(40)' 'NO'
''         NULL      ''
'gender'   'CHAR(1)' 'YES'
''         NULL      ''
'nation_code' 'CHAR(3)' 'YES'
''         NULL      ''
'event'    'VARCHAR(30)' 'YES'
''         NULL      ''

```

```
SHOW COLUMNS FROM athlete WHERE field LIKE '%c%';
```

Field	Type	Null
Key	Default	Extra
'code'	'INTEGER'	'NO'
'PRI'	NULL	'auto_increment'
'nation_code'	'CHAR(3)'	'YES'
''	NULL	''

```
SHOW COLUMNS FROM athlete WHERE "type" = 'INTEGER' and "key"='PRI'
AND extra='auto_increment';
```

Field	Type	Null
Key	Default	Extra
'code'	'INTEGER'	'NO'
'PRI'	NULL	'auto_increment'

```
SHOW FULL COLUMNS FROM athlete WHERE field LIKE '%c%';
```

Field	Type	Collation
Null	Key	Default
Extra	Comment	
'code'	'INTEGER'	NULL
'NO'	'PRI'	NULL
'auto_increment'	NULL	
'nation_code'	'CHAR(3)'	'iso88591_bin'
'YES'	''	NULL
''	NULL	

SHOW INDEX

인덱스 정보를 출력한다.


```
SHOW (INDEX | INDEXES | KEYS) (FROM | IN) [schema_name.]table_name;
```

해당 질의는 다음과 같은 칼럼을 가진다.

칼럼 이름	타입	설명
Table	VARCHAR	테이블 이름
Non_unique	INTEGER	중복 가능 여부 • 0: 데이터 중복 불가능 • 1: 데이터 중복 가능
Key_name	VARCHAR	인덱스 이름
Seq_in_index	INTEGER	인덱스에 있는 칼럼의 일련번호. 1부터 시작한다.
Column_name	VARCHAR	칼럼 이름
Collation	VARCHAR	칼럼이 인덱스에서 정렬되는 방법. 'A'는 오름차순 (Ascending), NULL 은 비정렬을 의미한다.
Cardinality	INTEGER	인덱스에서 고유한 값의 개수를 측정한 수치. 카디널리티가 높을수록 해당 인덱스를 선택 가능성이 높아진다. 이 값은 SHOW INDEX 가 실행되면 매번 업데이트된다. 이 값은 근사치임에 유의한다.
Sub_part	INTEGER	칼럼의 일부만 인덱스된 경우 인덱스된 문자의 바이트 수.

칼럼 이름	타입	설명
		칼럼 전체가 인덱스되면 NULL 이다.
Packed		키가 어떻게 팩되었는지 (packed)를 나타냄. 팩되지 않은 경우 NULL . 현재 지원 안 함.
Null	VARCHAR	칼럼이 NULL 을 포함할 수 있으면 YES, 그렇지 않으면 NO.
Index_type	VARCHAR	사용되는 인덱스(현재 BTREE 만 지원한다).
Func	VARCHAR	함수 인덱스에서 사용되는 함수
Comment	VARCHAR	인덱스를 설명하기 위한 주석
Visible	VARCHAR	인덱스의 가시성을 보여준다 (YES/NO)

다음은 이 구문을 수행한 예이다.

```
SHOW INDEX IN athlete;
```

```

Table          Non_unique  Key_name          Seq_in_index
Column_name    Collation  Cardinality  Sub_part  Packed  Null
Index_type     func      Comment      Visible
=====
=====
=====
'public.athlete'      0  'pk_athlete_code'      1
'code'                'A'      6677      NULL  NULL  'NO'
'BTREE'              NULL  NULL  'YES'
```

```

CREATE TABLE tbl1 (i1 INTEGER , i2 INTEGER NOT NULL, i3 INTEGER
UNIQUE, s1 VARCHAR(10), s2 VARCHAR(10), s3 VARCHAR(10) UNIQUE);

CREATE INDEX i_tbl1_i1 ON tbl1 (i1 DESC);
CREATE INDEX i_tbl1_s1 ON tbl1 (s1 (7));
CREATE INDEX i_tbl1_i1_s1 ON tbl1 (i1, s1);
CREATE UNIQUE INDEX i_tbl1_i2_s2 ON tbl1 (i2, s2);

ALTER INDEX i_tbl1_s1 ON tbl1 INVISIBLE;

SHOW INDEXES FROM tbl1;

```

Table	Non_unique	Key_name	Seq_in_index
Column_name	Collation	Cardinality	Sub_part
Index_type	Func	Comment	Visible
'public.tbl1'		1	'i_tbl1_i1'
'i1'	'D'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		1	'i_tbl1_i1_s1'
'i1'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		1	'i_tbl1_i1_s1'
's1'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		0	'i_tbl1_i2_s2'
'i2'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		0	'i_tbl1_i2_s2'
's2'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		1	'i_tbl1_s1'
's1'	'A'	0	7
'BTREE'	NULL	NULL	'NO'
'public.tbl1'		0	'u_tbl1_i3'
'i3'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'
'public.tbl1'		0	'u_tbl1_s3'
's3'	'A'	0	NULL
'BTREE'	NULL	NULL	'YES'

SHOW COLLATION

데이터베이스에서 지원하는 콜레이션 리스트를 출력한다. LIKE 절은 콜레이션 이름이 매칭되는 정보를 출력한다.

```
SHOW COLLATION [LIKE 'pattern'];
```

해당 질의는 다음과 같은 칼럼을 가진다.

칼럼 이름	타입	설명
Collation	VARCHAR	콜레이션 이름
Charset	CHAR(1)	문자셋 이름
Id	INTEGER	콜레이션 ID
Built_in	CHAR(1)	내장 콜레이션 여부. 내장 콜레이션들은 하드-코딩되어 있어 추가 혹은 삭제가 불가능하다.
Expansions	CHAR(1)	확장이 있는 콜레이션인지 여부. 자세한 내용은 확장 을 참조한다.
Strength	CHAR(1)	문자 간 비교를 위한 기준. 이 기준에 따라 문자 순서가 달라질 수 있다. 이에 대한 설명은 콜레이션 속성 을 참고한다.

다음은 이 구문을 수행한 예이다.

```
SHOW COLLATION;
```

```

Collation      Charset      Id
Built_in      Expansions   Strength
=====
'euckr_bin'   'euckr'      8
'Yes'         'No'         'Not applicable'
'iso88591_bin' 'iso88591'   0

```

'Yes'	'No'	'Not applicable'
'iso88591_en_ci'	'iso88591'	3
'Yes'	'No'	'Not applicable'
'iso88591_en_cs'	'iso88591'	2
'Yes'	'No'	'Not applicable'
'utf8_bin'	'utf8'	1
'Yes'	'No'	'Not applicable'
'utf8_de_exp'	'utf8'	76
'No'	'Yes'	'Tertiary'
'utf8_de_exp_ai_ci'	'utf8'	72
'No'	'Yes'	'Primary'
'utf8_en_ci'	'utf8'	5
'Yes'	'No'	'Not applicable'
'utf8_en_cs'	'utf8'	4
'Yes'	'No'	'Not applicable'
'utf8_es_cs'	'utf8'	85
'No'	'No'	'Quaternary'
'utf8_fr_exp_ab'	'utf8'	94
'No'	'Yes'	'Tertiary'
'utf8_gen'	'utf8'	32
'No'	'No'	'Quaternary'
'utf8_gen_ai_ci'	'utf8'	37
'No'	'No'	'Primary'
'utf8_gen_ci'	'utf8'	44
'No'	'No'	'Secondary'
'utf8_ja_exp'	'utf8'	124
'No'	'Yes'	'Tertiary'
'utf8_ja_exp_cbm'	'utf8'	125
'No'	'Yes'	'Tertiary'
'utf8_km_exp'	'utf8'	132
'No'	'Yes'	'Quaternary'
'utf8_ko_cs'	'utf8'	7
'Yes'	'No'	'Not applicable'
'utf8_ko_cs_uca'	'utf8'	133
'No'	'No'	'Quaternary'
'utf8_tr_cs'	'utf8'	6
'Yes'	'No'	'Not applicable'
'utf8_tr_cs_uca'	'utf8'	205
'No'	'No'	'Quaternary'
'utf8_vi_cs'	'utf8'	221
'No'	'No'	'Quaternary'

```
SHOW COLLATION LIKE '%_ko_%';
```

Collation	Charset	Id
Built_in	Expansions	Strength
=====	=====	=====
'utf8_ko_cs'	'utf8'	7

```
'Yes'      'No'      'Not applicable'
'utf8_ko_cs_uca'  'utf8'      133
'No'      'No'      'Quaternary'
```

SHOW TIMEZONES

현재 CUBRID에 설정된 타임 존 정보를 출력한다.

```
SHOW [FULL] TIMEZONES [LIKE 'pattern'];
```

FULL이 명시되지 않으면 타임 존의 영역 이름을 가진 하나의 칼럼을 출력한다. 칼럼의 이름은 timezone_region이다.

FULL이 명시되면 4개의 칼럼을 가진 타임존 정보를 출력한다.

LIKE 절을 사용하면 이와 매칭되는 timezone_region 을 검색할 수 있다.

칼럼 이름	타입	설명
timezone_region	VARCHAR(32)	타임존 영역 이름
region_offset	VARCHAR(32)	일광 절약 시간을 고려하지 않은 타임존 영역의 오프셋
dst_offset	VARCHAR(32)	일광 절약 시간을 고려한 타임존 영역의 오프셋
dst_abbreviation	VARCHAR(32)	일광 절약 시간이 적용된 영역의 약어

region_offset, dst_offset, dst_abbreviation 값은 현재 날짜와 시간을 기준으로 출력된다.

타임 존 영역이 일광 절약 시간(daylight saving time) 규칙을 적용하지 않는다면, dst_offset과 dst_abbreviation 값은 NULL 값이 된다.

현재의 날짜에 일광 절약 시간이 적용되지 않으면 dst_offset 값은 '0'이 되고 dst_abbreviation 값은 빈 문자열("")이 된다.

WHERE 조건 없는 LIKE 조건은 첫 번째 칼럼에 적용된다. WHERE 조건은 결과를 필터링하기 위해 사용될 수 있다.

```
SHOW TIMEZONES;
```

```
timezone_region
=====
'Africa/Abidjan'
'Africa/Accra'
'Africa/Addis_Ababa'
'Africa/Algiers'
'Africa/Asmara'
'Africa/Asmera'
...
'US/Michigan'
'US/Mountain'
'US/Pacific'
'US/Samoa'
'UTC'
'Universal'
'W-SU'
'WET'
'Zulu'
```

```
SHOW FULL TIMEZONES;
```

timezone_region dst_abbreviation	region_offset	dst_offset
=====		
'Africa/Abidjan'	' +00:00 '	' +00:00 '
'GMT'		
'Africa/Accra'	' +00:00 '	NULL
NULL		
'Africa/Addis_Ababa'	' +03:00 '	' +00:00 '
'EAT'		
'Africa/Algiers'	' +01:00 '	' +00:00 '
'CET'		
'Africa/Asmara'	' +03:00 '	' +00:00 '
'EAT'		
'Africa/Asmera'	' +03:00 '	' +00:00 '
'EAT'		
...		
'US/Michigan'	' -05:00 '	' +00:00 '
'EST'		
'US/Mountain'	' -07:00 '	' +00:00 '
'MST'		
'US/Pacific'	' -08:00 '	' +00:00 '
'PST'		
'US/Samoa'	' -11:00 '	' +00:00 '

```
'SST'
'UTC'          '+00:00'          '+00:00'
'UTC'
'Universal'    '+00:00'          '+00:00'
'UTC'
'W-SU'         '+04:00'          '+00:00'
'MSK'
'WET'          '+00:00'          '+00:00'
'WET'
'Zulu'         '+00:00'          '+00:00'
'UTC'
```

```
SHOW FULL TIMEZONES LIKE '%Paris%';
```

```
timezone_region      region_offset      dst_offset
dst_abbreviation
=====
=====
'Europe/Paris'      '+01:00'          '+00:00'
'CET'
```

SHOW GRANTS

데이터베이스의 사용자 계정에 부여된 권한을 출력한다.

```
SHOW GRANTS FOR user_name;
```

다음은 이 구문을 수행한 예이다.

```
CREATE TABLE testgrant (id INT);
CREATE USER user1;
GRANT INSERT,SELECT ON testgrant TO user1;

SHOW GRANTS FOR user1;
```

```
Grants for USER1
=====
'GRANT INSERT, SELECT ON dba.testgrant TO USER1'
```


SHOW CREATE TABLE

테이블 이름을 지정하면 해당 테이블의 **CREATE TABLE** 문을 출력한다.

```
SHOW CREATE TABLE [schema_name.]table_name;
```

```
SHOW CREATE TABLE nation;
```

```
TABLE                CREATE TABLE
=====
'public.nation'      'CREATE TABLE [nation] ([code] CHARACTER(3)
NOT NULL, [name] CHARACTER VARYING(40) NOT NULL, [continent]
CHARACTER VARYING(10), [capital] CHARACTER VARYING(30), CONSTRAINT
[pk_nation_code] PRIMARY KEY ([code])) DONT_REUSE_OID, COLLATE
iso88591_bin'
```

SHOW CREATE TABLE 문은 사용자가 입력한 구문을 그대로 출력하지는 않는다. 예를 들어, 사용자가 입력한 커멘트를 출력하지 않으며, 테이블 명이나 칼럼 명은 항상 소문자로 출력한다.

SHOW CREATE VIEW

뷰 이름을 지정하면 해당 **CREATE VIEW** 문을 출력한다.

```
SHOW CREATE VIEW view_name;
```

다음은 이 구문을 수행한 예이다.

```
SHOW CREATE VIEW db_class;
```

```
View                Create View
=====
'db_class'          'SELECT [c].[class_name], CAST([c].[owner].
[name] AS VARCHAR(255)), CASE [c].[class_type] WHEN 0 THEN 'CLASS'
WHEN 1 THEN 'VCLASS' ELSE 'UNKNOWN' END, CASE WHEN MOD([c].
[is_system_class], 2) = 1 THEN 'YES' ELSE 'NO' END, CASE [c].
[tde_algorithm] WHEN 0 THEN 'NONE' WHEN 1 THEN 'AES' WHEN 2 THEN
'ARIA' END, CASE WHEN [c].[sub_classes] IS NULL THEN 'NO' ELSE
NVL((SELECT 'YES' FROM [_db_partition] [p] WHERE [p].[class_of] =
[c] and [p].[pname] IS NULL), 'NO') END, CASE WHEN MOD([c].
[is_system_class] / 8, 2) = 1 THEN 'YES' ELSE 'NO' END, [coll].
[coll_name], [c].[comment] FROM [_db_class] [c], [_db_collation]
[coll] WHERE [c].[collation_id] = [coll].[coll_id] AND (CURRENT_USER
= 'DBA' OR {[c].[owner].[name]} SUBSETEQ(SELECT SET{CURRENT_USER} +
```

```
COALESCE(SUM(SET{[t].[g].[name]}), SET{}) FROM [db_user] [u],
TABLE([groups]) AS [t]([g]) WHERE [u].[name] = CURRENT_USER) OR
{[c]} SUBSETEQ ( SELECT SUM(SET{[au].[class_of]}) FROM [_db_auth]
[au] WHERE {[au].[grantee].[name]} SUBSETEQ ( SELECT
SET{CURRENT_USER} + COALESCE(SUM(SET{[t].[g].[name]}), SET{}) FROM
[db_user] [u], TABLE([groups]) AS [t]([g]) WHERE [u].[name] =
CURRENT_USER) AND [au].[auth_type] = 'SELECT'))'
```

SHOW ACCESS STATUS

SHOW ACCESS STATUS 문은 데이터베이스 사용자의 최근 로그인 정보를 출력한다. 이 명령은 데이터베이스 계정이 DBA인 사용자만 사용할 수 있다.

```
SHOW ACCESS STATUS [LIKE 'pattern' | WHERE expr];
```

해당 구문은 다음과 같은 칼럼을 출력한다.

칼럼 이름	타입	설명
user_name	VARCHAR(32)	DB 사용자 계정
last_access_time	DATETIME	DB 사용자가 마지막으로 접속한 시간
last_access_host	VARCHAR(32)	마지막으로 접속한 호스트
program_name	VARCHAR(32)	클라이언트 프로그램 이름 (broker_cub_cas_1, csqll..)

다음은 해당 구문을 실행한 결과이다.

```
SHOW ACCESS STATUS;
```

```
user_name last_access_time last_access_host program_name
=====
=====
'DBA' 08:19:31.000 PM 02/10/2014 127.0.0.1 'csqll'
'PUBLIC' NULL NULL NULL
```

i 참고

SHOW ACCESS STATUS의 로그인 정보는 데이터베이스가 재시작되면 초기화되며, HA 환경에서 복제되지 않으므로 각 노드마다 다른 결과를 출력된다.

SHOW EXEC STATISTICS

실행한 질의의 실행 통계 정보를 출력한다.

- 통계 정보 수집을 시작하려면 세션 변수 **@collect_exec_stats** 의 값을 1로 설정하며, 종료하려면 0으로 설정한다.
- 통계 정보 수집 결과를 출력한다.
 - **SHOW EXEC STATISTICS**는 data_page_fetches, data_page_dirties, data_page_ioreads, data_page_iowrites 이렇게 4가지 항목의 데이터 페이지 통계 정보를 출력하며, 결과 칼럼은 통계 정보 이름과 값에 해당하는 variable 칼럼과 value 칼럼으로 구성된다. **SHOW EXEC STATISTICS** 문을 실행하면 그동안 누적된 통계 정보가 초기화된다.
 - **SHOW EXEC STATISTICS ALL**은 모든 항목의 통계 정보를 출력한다.

통계 정보 각 항목에 대한 자세한 설명은 [statdump](#)을 참고한다.

```
SHOW EXEC STATISTICS [ALL];
```

다음은 이 구문을 수행한 예이다.

```
-- set session variable @collect_exec_stats as 1 to start collecting
the statistical information.
SET @collect_exec_stats = 1;
SELECT * FROM db_class;

-- print the statistical information of the data pages.
SHOW EXEC STATISTICS;
```

variable	value
'data_page_fetches'	332
'data_page_dirties'	85
'data_page_ioreads'	18
'data_page_iowrites'	28

```
SELECT * FROM db_index;
```

```
-- print all of the statistical information.
```

```
SHOW EXEC STATISTICS ALL;
```

variable	value
=====	
'file_creates'	0
'file_removes'	0
'file_ioreads'	6
'file_iowrites'	0
'file_iosynches'	0
'data_page_fetches'	548
'data_page_dirties'	34
'data_page_ioreads'	6
'data_page_iowrites'	0
'log_page_ioreads'	0
'log_page_iowrites'	0
'log_append_records'	0
'log_archives'	0
'log_start_checkpoints'	0
'log_end_checkpoints'	0
'log_wals'	0
'page_locks_acquired'	13
'object_locks_acquired'	9
'page_locks_converted'	0
'object_locks_converted'	0
'page_locks_re-requested'	0
'object_locks_re-requested'	8
'page_locks_waits'	0
'object_locks_waits'	0
'tran_commits'	3
'tran_rollbacks'	0
'tran_savepoints'	0
'tran_start_topops'	6
'tran_end_topops'	6
'tran_interrupts'	0
'btree_inserts'	0
'btree_deletes'	0
'btree_updates'	0
'btree_covered'	0
'btree_noncovered'	2
'btree_resumes'	0
'btree_multirange_optimization'	0
'query_selects'	4
'query_inserts'	0
'query_deletes'	0
'query_updates'	0
'query_sscans'	2
'query_iscans'	4

```

'query_lscans'          0
'query_setscans'        2
'query_methscans'       0
'query_nljoins'         2
'query_mjoins'          0
'query_objfetches'      0
'query_holdable_cursors' 0
'sort_io_pages'         0
'sort_data_pages'       0
'network_requests'      88
'adaptive_flush_pages'  0
'adaptive_flush_log_pages' 0
'adaptive_flush_max_pages' 0
'prior_lsa_list_size'   0
'prior_lsa_list_maxed'  0
'prior_lsa_list_removed' 0
'heap_stats_bestspace_entries' 0
'heap_stats_bestspace_maxed' 0

```

진단(Diagnostics)

다음 진단(Diagnostics) SHOW 명령은 데이터베이스 계정이 DBA 또는 DBA 멤버인 사용자만 사용할 수 있다.

SHOW VOLUME HEADER

명시한 볼륨의 헤더 정보를 출력한다.

```
SHOW VOLUME HEADER OF volume_id;
```

해당 구문은 다음과 같은 칼럼을 출력한다.

칼럼 이름	타입	설명
Volume_id	INT	볼륨 식별자
Magic_symbol	VARCHAR(100)	볼륨 파일의 매직 값
Io_page_size	INT	DB 볼륨의 페이지
Purpose	VARCHAR(32)	

칼럼 이름	타입	설명
		볼륨 사용 목적 : '영구적 데이터 목적' 또는 '일시적 데이터 목적'
Type	VARCHAR(32)	볼륨 타입, '영구적 볼륨' 또는 '일시적 볼륨'
Sector_size_in_pages	INT	페이지 내 섹터의 크기
Num_total_sectors	INT	섹터 전체 개수
Num_free_sectors	INT	여유 섹터 개수
Num_max_sectors	INT	섹터 수의 최대값
Hint_alloc_sector	INT	할당될 다음 섹터에 대한 힌트
Sector_alloc_table_size_in_pages	INT	페이지 내 섹터 할당 테이블 크기
Sector_alloc_table_first_page	INT	섹터 할당 테이블의 첫번째 페이지
Page_alloc_table_size_in_pages	INT	페이지 내 페이지 할당 테이블의 크기
Page_alloc_table_first_page	INT	페이지 할당 테이블의 첫번째 페이지
Last_system_page	INT	마지막 시스템 페이지

칼럼 이름	타입	설명
Creation_time	DATETIME	볼륨 생성 시간
Db_charset	INT	데이터베이스 문자셋번호
Checkpoint_lsa	VARCHAR(64)	이 볼륨의 복구 절차를 시작하는 가장 작은 로그 일련 주소
Boot_hfid	VARCHAR(64)	다중 볼륨과 데이터베이스 기동을 위한 시스템 힙 파일ID
Full_name	VARCHAR(255)	볼륨의 전체 경로
Next_volume_id	INT	다음 볼륨의 ID
Next_vol_full_name	VARCHAR(255)	다음 볼륨의 전체 경로
Remarks	VARCHAR(64)	볼륨에 대한 설명

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW VOLUME HEADER OF 0;
```

```
<00001> Volume_id           : 0
        Magic_symbol        : 'MAGIC SYMBOL = CUBRID/'
Volume at disk location = 32'
        Io_page_size         : 16384
        Purpose              : 'Permanent data purpose'
        Type                 : 'Permanent Volume'
        Sector_size_in_pages : 64
        Num_total_sectors    : 512
        Num_free_sectors     : 459
        Num_max_sectors      : 512
        Hint_alloc_sector    : 0
        Sector_alloc_table_size_in_pages: 1
```

```

Sector_alloc_table_first_page : 1
Last_system_page              : 1
Creation_time                  : 09:46:41.000 PM 05/23/2017
Db_charset                     : 3
Checkpoint_lsa                 : '(0|12832)'
Boot_hfid                      : '(0|41|50)'
Full_name                      : '/home1/brightest/CUBRID/
databases/demodb/demodb'
Next_volume_id                 : -1
Next_vol_full_name             : ''
Remarks                       : ''

```

SHOW LOG HEADER

활성 로그(active log) 파일의 헤더 정보를 출력한다.

```
SHOW LOG HEADER [OF file_path];
```

OF file_path를 생략하면 메모리의 헤더 정보를 출력하며, OF file_path를 포함하면 file_path의 헤더 정보를 출력한다. file_path에는 활성 로그 파일의 상대경로 또는 절대경로를 입력할 수 있다. 상대경로 입력 시 기준이 되는 경로는 활성 로그 파일이 존재하는 디렉터리이다.

해당 구문은 다음의 칼럼을 출력한다.

Column name	Type	Description
Volume_id	INT	볼륨 식별자
Magic_symbol	VARCHAR(32)	로그 파일의 매직 값
Magic_symbol_location	INT	로그 페이지의 매직 심볼 위치
Creation_time	DATETIME	볼륨 생성 시간
Release	VARCHAR(32)	CUBRID 릴리즈 버전
Compatibility_disk_version	VARCHAR(32)	현재 릴리즈 버전에 대한 DB의 호환성

Column name	Type	Description
Db_page_size	INT	DB 페이지의 크기
Log_page_size	INT	로그 페이지의 크기
Shutdown	INT	로그 셧다운의 여부
Next_trans_id	INT	다음 트랜잭션 ID
Num_avg_trans	INT	평균 트랜잭션 개수
Num_avg_locks	INT	평균 객체 잠금 개수
Num_active_log_pages	INT	활성로그 부분에서 페이지 개수
Db_charset	INT	DB의 문자셋 번호
First_active_log_page	BIGINT	활성 로그에서 물리적 위치 1에 대한 논리 페이지
Current_append	VARCHAR(64)	현재의 추가된 위치
Checkpoint	VARCHAR(64)	복구 프로세스를 시작하는 가장 작은 로그 일련 주소
Next_archive_page_id	BIGINT	보관할 다음 논리 페이지
Active_physical_page_id	INT	보관할 논리 페이지의 물리적 위치

Column name	Type	Description
Next_archive_num	INT	다음 보관 로그 번호
Last_archive_num_for_syscrashes	INT	시스템 비정상 종료 대비하여 필요한 최종 보관 로그 번호
Last_deleted_archive_num	INT	최종 삭제된 보관 로그 번호
Backup_lsa_level0	VARCHAR(64)	백업 수준 0의 LSA(log sequence number)
Backup_lsa_level1	VARCHAR(64)	백업 수준 1의 LSA
Backup_lsa_level2	VARCHAR(64)	백업 수준 2의 LSA
Log_prefix	VARCHAR(256)	로그 prefix 이름
Has_logging_been_skipped	INT	로깅의 생략 여부
Perm_status	VARCHAR(64)	현재 사용하지 않음
Backup_info_level0	VARCHAR(128)	백업 수준 0의 상세 정보. 현재는 백업 시작 시간만 저장됨
Backup_info_level1	VARCHAR(128)	백업 수준 1의 상세 정보. 현재는 백업 시작 시간만 저장됨
Backup_info_level2	VARCHAR(128)	백업 수준 2의 상세 정보. 현재는 백업 시작 시간만 저장됨
Ha_server_state	VARCHAR(32)	HA 서버 상태. 다음 값 중 하나: na, idle, active, to-be-

Column name	Type	Description
		active, standby, to-be-standby, maintenance, dead
Ha_file	VARCHAR(32)	HA 복제 상태. 다음 값 중 하나: clear, archived, sync
Eof_lsa	VARCHAR(64)	LSA 파일의 끝
Smallest_lsa_at_last_checkpoint	VARCHAR(64)	맨 마지막 체크포인트의 가장 작은 LSA, NULL 값이 될 수 있음
Next_mvcc_id	BIGINT	다음 트랜잭션에서 사용될 다음 MVCCID 값
Mvcc_op_log_lsa	VARCHAR(32)	MVCC 작업을 위한 로그 항목을 연결하는 데 사용되는 LSA
Last_block_oldest_mvcc_id	BIGINT	로그 데이터 블록에서 가장 오래된 MVCC 를 찾기 위한 ID 값, NULL 값이 될 수 있음
Last_block_newest_mvcc_id	BIGINT	로그 데이터 블록에서 가장 최신의 MVCC 를 찾기 위한 ID 값, NULL 값이 될 수 있음

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW LOG HEADER;
```

```
<00001> Volume_id           : -2
        Magic_symbol       : 'CUBRID/LogActive'
        Magic_symbol_location : 16
```

```

Creation_time           : 09:46:41.000 PM 05/23/2017
Release                 : '10.0.0'
Compatibility_disk_version : '10'
Db_page_size            : 16384
Log_page_size           : 16384
Shutdown                : 0
Next_trans_id           : 17
Num_avg_trans           : 3
Num_avg_locks           : 30
Num_active_log_pages    : 1279
Db_charset               : 3
First_active_log_page   : 0
Current_append           : '(102|5776)'
Checkpoint               : '(101|7936)'
Next_archive_page_id    : 0
Active_physical_page_id : 1
Next_archive_num         : 0
Last_archive_num_for_syscrashes : -1
Last_deleted_archive_num : -1
Backup_lsa_level0       : '(-1|-1)'
Backup_lsa_level1       : '(-1|-1)'
Backup_lsa_level2       : '(-1|-1)'
Log_prefix              : 'mvccdb'
Has_logging_been_skipped : 0
Perm_status             : 'LOG_PSTAT_CLEAR'
Backup_info_level0      : 'time: N/A'
Backup_info_level1      : 'time: N/A'
Backup_info_level2      : 'time: N/A'
Ha_server_state         : 'idle'
Ha_file                 : 'UNKNOWN'
Eof_lsa                 : '(102|5776)'
Smallest_lsa_at_last_checkpoint : '(101|7936)'
Next_mvcc_id            : 6
Mvcc_op_log_lsa         : '(102|5488)'
Last_block_oldest_mvcc_id : 4
Last_block_newest_mvcc_id : 5

```

```
SHOW LOG HEADER OF 'demodb_lgat';
```

```

<00001> Volume_id       : -2
        Magic_symbol     : 'CUBRID/LogActive'
        Magic_symbol_location : 16
        Creation_time      : 09:46:41.000 PM 05/23/2017
        Release            : '10.0.0'
        Compatibility_disk_version : '10'
        Db_page_size       : 16384
        Log_page_size      : 16384
        Shutdown           : 0
        Next_trans_id      : 15

```

```

Num_avg_trans           : 3
Num_avg_locks           : 30
Num_active_log_pages    : 1279
Db_charset              : 3
First_active_log_page   : 0
Current_append           : '(101|8016)'
Checkpoint              : '(101|7936)'
Next_archive_page_id    : 0
Active_physical_page_id : 1
Next_archive_num        : 0
Last_archive_num_for_syscrashes: -1
Last_deleted_archive_num: -1
Backup_lsa_level0       : '(-1|-1)'
Backup_lsa_level1       : '(-1|-1)'
Backup_lsa_level2       : '(-1|-1)'
Log_prefix              : 'mvccdb'
Has_logging_been_skipped: 0
Perm_status             : 'LOG_PSTAT_CLEAR'
Backup_info_level0      : 'time: N/A'
Backup_info_level1      : 'time: N/A'
Backup_info_level2      : 'time: N/A'
Ha_server_state         : 'idle'
Ha_file                 : 'UNKNOWN'
Eof_lsa                 : '(101|8016)'
Smallest_lsa_at_last_checkpoint: '(101|7936)'
Next_mvcc_id            : 4
Mvcc_op_log_lsa         : '(-1|-1)'
Last_block_oldest_mvcc_id: NULL
Last_block_newest_mvcc_id: NULL

```

SHOW ARCHIVE LOG HEADER

보관 로그(archive log) 파일의 헤더 정보를 출력한다.

```
SHOW ARCHIVE LOG HEADER OF file_path;
```

file_path에는 보관 로그 파일의 상대경로 또는 절대경로를 입력할 수 있다. 상대경로 입력 시 기준이 되는 경로는 보관 로그 파일이 존재하는 디렉터리이다.

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Volume_id	INT	로그 볼륨 ID

칼럼 이름	타입	설명
Magic_symbol	VARCHAR(32)	보관 로그 파일의 매직 값
Magic_symbol_location	INT	로그 페이지로부터 매직 심볼 위치
Creation_time	DATETIME	볼륨 생성 시간
Next_trans_id	BIGINT	다음 트랜잭션 ID
Num_pages	INT	보관 로그에서 페이지의 개수
First_page_id	BIGINT	보관 로그에서 물리적 위치 1에 대한 논리 페이지 ID
Archive_num	INT	보관 로그 번호

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW ARCHIVE LOG HEADER OF 'demodb_lgar001';
```

```
<00001> Volume_id      : -20
        Magic_symbol   : 'CUBRID/LogArchive'
        Magic_symbol_location: 16
        Creation_time    : 04:42:28.000 PM 12/11/2013
        Next_trans_id    : 22695
        Num_pages        : 1278
        First_page_id    : 1278
        Archive_num      : 1
```

SHOW HEAP HEADER

명시한 테이블의 헤더 페이지를 출력한다.

```
SHOW [ALL] HEAP HEADER OF [schema_name.]table_name;
```

· ALL: 분할 테이블에서 “ALL” 키워드가 주어지면 기반 테이블과 분할 테이블이 같이 출력된다.

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Class_name	VARCHAR(256)	테이블 이름
Class_oid	VARCHAR(64)	포맷: (volid pageid slotid)
Volume_id	INT	파일이 위치해 있는 볼륨의 식별자
File_id	INT	파일 식별자
Header_page_id	INT	첫 페이지 식별자(헤더 페이지)
Overflow_vfid	VARCHAR(64)	오버플로우 파일 식별자(존재하는 경우)
Next_vpid	VARCHAR(64)	다음 페이지 (예: 힙 파일의 두 번째 페이지)
Unfill_space	INT	페이지 공간이 이 값보다 작을 때 INSERT 중지. UPDATE 시에는 이 값을 사용 안 함
Estimates_num_pages	BIGINT	힙 페이지 개수의 추정치
Estimates_num_recs	BIGINT	힙 내 객체 개수의 추정치

칼럼 이름	타입	설명
Estimates_avg_rec_len	INT	레코드 전체 길이의 추정치
Estimates_num_high_best	INT	최소의 HEAP_DROP_FREE_SPACE를 가진 것으로 추정되는 베스트 페이지의 배열에 있는 페이지 개수. 이 숫자가 0이고 최소한 다른 HEAP_NUM_BEST_SPACESTAT S 개수만큼의 베스트 페이지 가 있으면, 그것을 찾는다.
Estimates_num_others_high_best	INT	베스트 페이지로 알려진 것으 로 추정되는 전체 개수. 이 베스트 페이지는 베스트 배 열에는 포함되어 있지 않고 최 소한 HEAP_DROP_FREE_SPACE를 가진 것으로 추정한다.
Estimates_head	INT	베스트 순환 배열의 헤드
Estimates_best_list	VARCHAR(512)	포맷: '((best[0].vpid.volid best[0].vpid.pageid), best[0].freespace), ... , ((best[9].vpid.volid best[9].vpid.pageid), best[9].freespace)'
Estimates_num_second_best	INT	두번째 베스트 힌트의 개수. 이 힌트는 두번째 베스트 배열 에 존재한다. 이들은 새로운 베스트 페이지를 찾을 때 사용 됨.

칼럼 이름	타입	설명
Estimates_head_second_best	INT	두번째 베스트 힌트의 헤드의 인덱스. 새로운 두번째 베스트 힌트는 이 인덱스에 저장된다.
Estimates_num_substitutions	INT	페이지 대체(substitution) 개수. 새로운 두번째 베스트 페이지를 두번째 베스트 힌트로 입력하기 위해 사용된다.
Estimates_second_best_list	VARCHAR(512)	포맷: '(second_best[0].vpid.valid second_best[0].vpid.pageid), ... (second_best[9].vpid.valid second_best[9].vpid.pageid)'
Estimates_last_vpid	VARCHAR(64)	포맷: '(valid pageid)'
Estimates_full_search_vpid	VARCHAR(64)	포맷: '(valid pageid)'

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW HEAP HEADER OF athlete;
```

```
<00001> Class_name           : 'athlete'
        Class_oid           : '(0|463|8)'
        Volume_id           : 0
        File_id             : 147
        Header_page_id      : 590
        Overflow_vfid       : '(-1|-1)'
        Next_vpid           : '(0|591)'
        Unfill_space        : 1635
        Estimates_num_pages  : 27
        Estimates_num_recs   : 6677
        Estimates_avg_rec_len : 54
```

```

Estimates_num_high_best      : 1
Estimates_num_others_high_best: 0
Estimates_head               : 0
Estimates_best_list          : '((0|826), 14516), ((-1|-1),
0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0),
((-1|-1), 0), ((-1|-1), 0), ((-1|-1),0), ((-1|-1), 0)')
Estimates_num_second_best    : 0
Estimates_head_second_best   : 0
Estimates_tail_second_best   : 0
Estimates_num_substitutions   : 0
Estimates_second_best_list    : '(-1|-1), (-1|-1), (-1|-1),
(-1|-1), (-1|-1), (-1|-1), (-1|-1), (-1|-1)'
Estimates_last_vpid          : '(0|826)'
Estimates_full_search_vpid    : '(0|590)'

```

```

CREATE TABLE participant2 (
    host_year INT,
    nation CHAR(3),
    gold INT,
    silver INT,
    bronze INT
)
PARTITION BY RANGE (host_year) (
    PARTITION before_2000 VALUES LESS THAN (2000),
    PARTITION before_2008 VALUES LESS THAN (2008)
);

```

```
SHOW ALL HEAP HEADER OF participant2;
```

```

<00001> Class_name           : 'participant2'
Class_oid                   : '(0|467|6)'
Volume_id                   : 0
File_id                     : 374
Header_page_id              : 940
Overflow_vfid                : '(-1|-1)'
Next_vpid                   : '(-1|-1)'
Unfill_space                 : 1635
Estimates_num_pages         : 1
Estimates_num_recs          : 0
Estimates_avg_rec_len       : 0
Estimates_num_high_best     : 1
Estimates_num_others_high_best: 0
Estimates_head              : 1
Estimates_best_list          : '((0|940), 16308), ((-1|-1),
0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0),
((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0)')
Estimates_num_second_best    : 0
Estimates_head_second_best   : 0

```

```

Estimates_tail_second_best      : 0
Estimates_num_substitutions     : 0
Estimates_second_best_list      : '(-1|-1), (-1|-1), (-1|-1),
(-1|-1), (-1|-1), (-1|-1), (-1|-1), (-1|-1)'
Estimates_last_vpid             : '(0|940)'
Estimates_full_search_vpid      : '(0|940)'
<00002> Class_name              :
'participant2__p__before_2000'
Class_oid                       : '(0|467|7)'
Volume_id                      : 0
File_id                        : 376
Header_page_id                 : 950
Overflow_vfid                   : '(-1|-1)'
Next_vpid                      : '(-1|-1)'
Unfill_space                    : 1635
Estimates_num_pages             : 1
Estimates_num_recs              : 0
Estimates_avg_rec_len           : 0
Estimates_num_high_best         : 1
Estimates_num_others_high_best : 0
Estimates_head                  : 1
Estimates_best_list             : '((0|950), 16308), ((-1|-1),
0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0),
((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0)'
Estimates_num_second_best       : 0
Estimates_head_second_best      : 0
Estimates_tail_second_best      : 0
Estimates_num_substitutions     : 0
Estimates_second_best_list      : '(-1|-1), (-1|-1), (-1|-1),
(-1|-1), (-1|-1), (-1|-1), (-1|-1)'
Estimates_last_vpid             : '(0|950)'
Estimates_full_search_vpid      : '(0|950)'
<00003> Class_name              :
'participant2__p__before_2008'
Class_oid                       : '(0|467|8)'
Volume_id                      : 0
File_id                        : 378
Header_page_id                 : 960
Overflow_vfid                   : '(-1|-1)'
Next_vpid                      : '(-1|-1)'
Unfill_space                    : 1635
Estimates_num_pages             : 1
Estimates_num_recs              : 0
Estimates_avg_rec_len           : 0
Estimates_num_high_best         : 1
Estimates_num_others_high_best : 0
Estimates_head                  : 1
Estimates_best_list             : '((0|960), 16308), ((-1|-1),
0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0),
((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0)'
Estimates_num_second_best       : 0
Estimates_head_second_best      : 0

```

```

Estimates_tail_second_best      : 0
Estimates_num_substitutions     : 0
Estimates_second_best_list      : '(-1|-1), (-1|-1), (-1|-1),
(-1|-1), (-1|-1), (-1|-1), (-1|-1), (-1|-1)'
Estimates_last_vpid             : '(0|960)'
Estimates_full_search_vpid      : '(0|960)'

```

```
SHOW HEAP HEADER OF participant2__p__before_2008;
```

```

<00001> Class_name              :
'participant2__p__before_2008'
      Class_oid                  : '(0|467|8)'
      Volume_id                  : 0
      File_id                    : 378
      Header_page_id             : 960
      Overflow_vfid              : '(-1|-1)'
      Next_vpid                  : '(-1|-1)'
      Unfill_space                : 1635
      Estimates_num_pages         : 1
      Estimates_num_recs         : 0
      Estimates_avg_rec_len       : 0
      Estimates_num_high_best     : 1
      Estimates_num_others_high_best : 0
      Estimates_head              : 1
      Estimates_best_list         : '((0|960), 16308), ((-1|-1),
0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0),
((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0), ((-1|-1), 0)'
      Estimates_num_second_best   : 0
      Estimates_head_second_best  : 0
      Estimates_tail_second_best  : 0
      Estimates_num_substitutions : 0
      Estimates_second_best_list  : '(-1|-1), (-1|-1), (-1|-1),
(-1|-1), (-1|-1), (-1|-1), (-1|-1)'
      Estimates_last_vpid         : '(0|960)'
      Estimates_full_search_vpid  : '(0|960)'

```

SHOW HEAP CAPACITY

명시한 테이블의 용량을 출력한다.

```
SHOW [ALL] HEAP CAPACITY OF [schema_name.] table_name;
```

· ALL: 분할 테이블에서 “ALL” 키워드가 주어지면 기반 테이블과 분할된 테이블이 같이 출력된다.

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Table_name	VARCHAR(256)	테이블 이름
Class_oid	VARCHAR(64)	힙 파일 식별자
Volume_id	INT	파일이 존재하는 볼륨 식별자
File_id	INT	파일 식별자
Header_page_id	INT	첫번째 페이지 식별자(헤더 페이지)
Num_recs	BIGINT	객체의 전체 개수
Num_relocated_recs	BIGINT	재할당된 레코드의 개수
Num_overflowed_recs	BIGINT	큰 레코드의 개수
Num_pages	BIGINT	힙 페이지의 전체 개수
Avg_rec_len	INT	평균 객체 길이
Avg_free_space_per_page	INT	페이지 당 평균 여유 공간
Avg_free_space_per_page_without_last_page	INT	마지막 페이지를 고려하지 않은 페이지 당 평균 여유 공간
Avg_overhead_per_page	INT	페이지 당 평균 오버헤드
Repr_id	INT	현재 캐시된 카탈로그 칼럼 정보

칼럼 이름	타입	설명
Num_total_attrs	INT	칼럼의 전체 개수
Num_fixed_width_attrs	INT	고정 길이 칼럼의 개수
Num_variable_width_attrs	INT	가변 길이 칼럼의 개수
Num_shared_attrs	INT	공유(shared) 칼럼의 개수
Num_class_attrs	INT	테이블 칼럼 개수
Total_size_fixed_width_attrs	INT	고정 길이 칼럼의 전체 크기

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW HEAP CAPACITY OF athlete;
```

```
<00001> Table_name           : 'athlete'
      Class_oid             : '(0|463|8)'
      Volume_id             : 0
      File_id               : 147
      Header_page_id        : 590
      Num_recs              : 6677
      Num_relocated_recs    : 0
      Num_overflowed_recs   : 0
      Num_pages             : 27
      Avg_rec_len           : 53
      Avg_free_space_per_page : 2139
      Avg_free_space_per_page_except_last_page: 1663
      Avg_overhead_per_page  : 993
      Repr_id               : 1
      Num_total_attrs       : 5
      Num_fixed_width_attrs  : 3
      Num_variable_width_attrs : 2
      Num_shared_attrs      : 0
      Num_class_attrs       : 0
      Total_size_fixed_width_attrs : 8
```

```
SHOW ALL HEAP CAPACITY OF participant2;
```

```
<00001> Table_name           : 'participant2'
        Class_oid           : '(0|467|6)'
        Volume_id           : 0
        File_id             : 374
        Header_page_id      : 940
        Num_recs             : 0
        Num_relocated_recs   : 0
        Num_overflowed_recs  : 0
        Num_pages            : 1
        Avg_rec_len          : 0
        Avg_free_space_per_page : 16016
        Avg_free_space_per_page_except_last_page: 0
        Avg_overhead_per_page : 4
        Repr_id              : 1
        Num_total_attrs      : 5
        Num_fixed_width_attrs : 5
        Num_variable_width_attrs : 0
        Num_shared_attrs     : 0
        Num_class_attrs      : 0
        Total_size_fixed_width_attrs : 20
<00002> Table_name           :
        'participant2__p__before_2000'
        Class_oid           : '(0|467|7)'
        Volume_id           : 0
        File_id             : 376
        Header_page_id      : 950
        Num_recs             : 0
        Num_relocated_recs   : 0
        Num_overflowed_recs  : 0
        Num_pages            : 1
        Avg_rec_len          : 0
        Avg_free_space_per_page : 16016
        Avg_free_space_per_page_except_last_page: 0
        Avg_overhead_per_page : 4
        Repr_id              : 1
        Num_total_attrs      : 5
        Num_fixed_width_attrs : 5
        Num_variable_width_attrs : 0
        Num_shared_attrs     : 0
        Num_class_attrs      : 0
        Total_size_fixed_width_attrs : 20
<00003> Table_name           :
        'participant2__p__before_2008'
        Class_oid           : '(0|467|8)'
        Volume_id           : 0
        File_id             : 378
        Header_page_id      : 960
        Num_recs             : 0
```

```

Num_relocated_recs      : 0
Num_overflowed_recs     : 0
Num_pages               : 1
Avg_rec_len             : 0
Avg_free_space_per_page : 16016
Avg_free_space_per_page_except_last_page: 0
Avg_overhead_per_page   : 4
Repr_id                 : 1
Num_total_attrs         : 5
Num_fixed_width_attrs   : 5
Num_variable_width_attrs: 0
Num_shared_attrs        : 0
Num_class_attrs         : 0
Total_size_fixed_width_attrs: 20

```

SHOW SLOTTED PAGE HEADER

명시한 슬롯 페이지의 헤더 정보를 출력한다.

```
SHOW SLOTTED PAGE HEADER (WHERE | OF) VOLUME = volume_num AND PAGE =
page_num;
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Volume_id	INT	페이지의 볼륨 식별자
Page_id	INT	페이지 식별자
Num_slots	INT	페이지에 할당된 슬롯 개수
Num_records	INT	페이지에 대한 레코드 개수
Anchor_type	VARCHAR(32)	다음 값 중 하나: ANCHORED, ANCHORED_DONT_REUSE_SLOTS, UNANCHORED_ANY_SEQUENCE,

칼럼 이름	타입	설명
		UNANCHORED_KEEP_SEQUENCE
Alignment	VARCHAR(8)	레코드에 대한 정렬 (alignment), 다음 값 중 하나: CHAR, SHORT, INT, DOUBLE
Total_free_area	INT	페이지 전체 여유 공간
Contiguous_free_area	INT	페이지 내 연속된 여유 공간
Free_space_offset	INT	페이지의 처음부터 페이지 내 첫번째 여유 공간 바이트 영역까지의 바이트 오프셋
Need_update_best_hint	INT	undo 복구를 위해 저장에 필요하면 true
Is_saving	INT	이 페이지를 위해 베스트 페이지를 업데이트해야 되면 true
Flags	INT	페이지의 플래그 값

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW SLOTTED PAGE HEADER OF VOLUME=0 AND PAGE=140;
```

```
<00001> Volume_id      : 0
        Page_id       : 140
        Num_slots      : 3
        Num_records    : 3
        Anchor_type    : 'ANCHORED_DONT_REUSE_SLOTS'
        Alignment      : 'INT'
        Total_free_area : 15880
```

```

Contiguous_free_area : 15880
Free_space_offset    : 460
Need_update_best_hint : 1
Is_saving             : 0
Flags                : 0

```

SHOW SLOTTED PAGE SLOTS

명시한 슬롯 페이지의 모든 슬롯 정보를 출력한다.

```
SHOW SLOTTED PAGE SLOTS (WHERE | OF) VOLUME = volume_num AND PAGE =
page_num;
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Volume_id	INT	페이지의 볼륨 식별자
Page_id	INT	페이지 식별자
Slot_id	INT	슬롯 식별자
Offset	INT	페이지의 시작부터 레코드의 시작까지의 바이트 오프셋
Type	VARCHAR(32)	레코드 타입, 다음 값 중 하나: REC_UNKNOWN, REC_ASSIGN_ADDRESS, REC_HOME, REC_NEWHOME, REC_RELOCATION, REC_BIGONE, REC_MARKDELETED, REC_DELETED_WILL_REUSE
Length	INT	레코드 길이

칼럼 이름	타입	설명
Waste	INT	버릴 것인지 여부

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW SLOTTED PAGE SLOTS OF VOLUME=0 AND PAGE=140;
```

```
<00001> Volume_id: 0
        Page_id  : 140
        Slot_id   : 0
        Offset    : 40
        Type      : 'HOME'
        Length    : 292
        Waste     : 0
<00002> Volume_id: 0
        Page_id  : 140
        Slot_id   : 1
        Offset    : 332
        Type      : 'HOME'
        Length    : 64
        Waste     : 0
<00003> Volume_id: 0
        Page_id  : 140
        Slot_id   : 2
        Offset    : 396
        Type      : 'HOME'
        Length    : 64
        Waste     : 0
```

SHOW INDEX HEADER

특정 테이블 내 인덱스의 헤더 페이지 정보를 출력한다.

```
SHOW INDEX HEADER OF [schema_name.]table_name.index_name;
```

ALL 키워드를 사용하면 특정 테이블에 속한 모든 인덱스의 헤더 정보를 출력한다.

```
SHOW ALL INDEXES HEADER OF [schema_name.]table_name;
```

해당 구문은 다음의 칼럼을 출력한다.

Column name	Type	Description
Table_name	VARCHAR(256)	테이블명
Index_name	VARCHAR(256)	인덱스명
Btid	VARCHAR(64)	BTID (volid fileid root_pageid)
Node_level	INT	노드 수준 (1 은 단말, 2 이상은 비단말)
Max_key_len	INT	서브트리의 최대 키 길이
Num_oids	INT	B트리에 저장된 OID 개수
Num_nulls	INT	NULL 의 개수
Num_keys	INT	B트리에 있는 고유 키의 개수
Topclass_oid	VARCHAR(64)	최상위 클래스의 oid 또는 NULL OID (고유 인덱스가 아님)(volid pageid slotid)
Unique	INT	고유값 유무
Overflow_vfid	VARCHAR(32)	VFID (volid fileid)
Key_type	VARCHAR(256)	타입명
Columns	VARCHAR(256)	인덱스를 구성하는 칼럼 리스트

다음은 이 구문을 수행한 예이다.

```
-- Prepare test environment
CREATE TABLE tbl1(a INT, b VARCHAR(5));
CREATE INDEX index_ab ON tbl1(a ASC, b DESC);
```

```
-- csql> ;line on
SHOW INDEX HEADER OF tbl1.index_ab;
```

```
<00001> Table_name      : 'tbl1'
        Index_name     : 'index_ab'
        Btid           : '(0|378|950)'
        Node_level     : 'LEAF'
        Max_key_len    : 0
        Num_oids       : -1
        Num_nulls      : -1
        Num_keys       : -1
        Topclass_oid   : '(0|469|4)'
        Unique         : 0
        Overflow_vfid  : '(-1|-1)'
        Key_type       : 'midxkey(integer,character varying(5))'
        Columns        : 'a,b DESC'
```

SHOW INDEX CAPACITY

테이블의 인덱스 용량 정보를 출력한다.

```
SHOW INDEX CAPACITY OF [schema_name.]table_name.index_name;
```

ALL 키워드를 사용하고 인덱스 이름을 생략하면 해당 테이블의 전체 인덱스의 용량 정보를 출력한다.

```
SHOW ALL INDEXES CAPACITY OF [schema_name.]table_name;
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Table_name	VARCHAR(256)	테이블 이름

칼럼 이름	타입	설명
Index_name	VARCHAR(256)	인덱스 이름
Btid	VARCHAR(64)	BTID (valid fileid root_pageid)
Num_distinct_key	INT	단말 노드(leaf) 페이지의 Distinct key 개수
Total_value	INT	트리에 저장된 값의 총 개수
Deduplicate_distinct_key	INT	단말 노드(leaf) 페이지의 Deduplicated Distinct key 개수
Num_fence_key	INT	펜스키(Fence-key)의 개수
Avg_num_value_per_key	INT	키당 OID 값의 평균 개수
Avg_num_value_per_deduplicate_key	INT	Deduplicated된 키당 OID 값의 평균 개수
Num_leaf_page	INT	단말 노드(leaf) 페이지 개수
Num_non_leaf_page	INT	비단말(NonLeaf) 노드 페이지 개수
Num_ovf_page	INT	단말 노드의 오버플로우 페이지 개수
Num_total_page	INT	전체 페이지 개수

칼럼 이름	타입	설명
Height	INT	트리의 높이
Avg_key_len	INT	평균 키 길이
Avg_rec_len	INT	평균 페이지 레코드 길이
Total_space	VARCHAR(64)	인덱스에 의해 점유되는 전체 공간
Total_used_space_non_ovf	VARCHAR(64)	할당된 페이지에서 사용된 총 공간(단말 노드의 오버플로우 페이지는 제외)
Total_free_space_non_ovf	VARCHAR(64)	할당된 페이지에서 사용되지 않은 총 공간(단말 노드의 오버플로우 페이지는 제외)
Total_used_space_ovf	VARCHAR(64)	단말 노드의 할당된 오버플로우 페이지에서 사용된 총 공간
Total_free_space_ovf	VARCHAR(64)	단말 노드의 할당된 오버플로우 페이지에서 사용되지 않은 총 공간
Avg_num_key_per_page_non_ovf	INT	단말 노드 페이지에서 페이지당 평균 키 개수
Avg_free_space_per_page_non_ovf	VARCHAR(64)	단말 노드 페이지에서 페이지당 평균 여유 공간
Avg_num_key_per_page_ovf	INT	

칼럼 이름	타입	설명
		단말 노드의 오버플로우 페이지에서 페이지 당 평균 키 개수
Avg_free_space_per_page_ovf	VARCHAR(64)	단말 노드의 오버플로우 페이지 당 평균 여유 공간
Max_num_ovf_page_a_key	INT	하나의 키에 대해 연결된 단말 노드의 오버플로우 페이지의 최대 개수

참고

Fence key는 B-tree 인덱스의 운용을 돕기 위해 단말 노드(leaf)에 추가되는 가상의 키이다.

다음은 이 구문을 수행한 예이다.

```
-- Prepare test environment
CREATE TABLE tbl1(a INT, b VARCHAR(5));
CREATE INDEX index_a ON tbl1(a ASC);
CREATE INDEX index_b ON tbl1(b ASC);
```

```
-- csql> ;line on
SHOW INDEX CAPACITY OF tbl1.index_a;
```

```
<00001> Table_name           : 'dba.tbl1'
        Index_name          : 'index_a'
        Btid                : '(0|4160|4161)'
        Num_distinct_key     : 0
        Total_value          : 0
        Deduplicate_distinct_key : 0
        Num_fence_key        : 0
        Avg_num_value_per_key : 0
        Avg_num_value_per_deduplicate_key : 0
        Num_leaf_page        : 1
        Num_non_leaf_page    : 0
        Num_ovf_page         : 0
```



```

Num_total_page      : 1
Height              : 1
Avg_key_len         : 0
Avg_rec_len         : 0
Total_space         : '16.0K'
Total_used_space_non_ovf : '120.0B'
Total_free_space_non_ovf : '15.8K'
Total_used_space_ovf : '0.0B'
Total_free_space_ovf : '0.0B'
Avg_num_key_per_page_non_ovf : 0
Avg_free_space_per_page_non_ovf : '15.8K'
Avg_num_ovf_page_per_key : 0
Avg_free_space_per_page_ovf : '0.0B'
Max_num_ovf_page_a_key : 0

```

```
SHOW ALL INDEXES CAPACITY OF tbl1;
```

```

<00001> Table_name      : 'dba.tbl1'
        Index_name     : 'index_a'
        Btid           : '(0|4160|4161)'
        Num_distinct_key : 0
        Total_value     : 0
        Deduplicate_distinct_key : 0
        Num_fence_key   : 0
        Avg_num_value_per_key : 0
        Avg_num_value_per_deduplicate_key : 0
        Num_leaf_page    : 1
        Num_non_leaf_page : 0
        Num_ovf_page     : 0
        Num_total_page   : 1
        Height          : 1
        Avg_key_len      : 0
        Avg_rec_len      : 0
        Total_space      : '16.0K'
        Total_used_space_non_ovf : '120.0B'
        Total_free_space_non_ovf : '15.8K'
        Total_used_space_ovf : '0.0B'
        Total_free_space_ovf : '0.0B'
        Avg_num_key_per_page_non_ovf : 0
        Avg_free_space_per_page_non_ovf : '15.8K'
        Avg_num_ovf_page_per_key : 0
        Avg_free_space_per_page_ovf : '0.0B'
        Max_num_ovf_page_a_key : 0
<00002> Table_name      : 'dba.tbl1'
        Index_name     : 'index_b'
        Btid           : '(0|4224|4225)'
        Num_distinct_key : 0
        Total_value     : 0
        Deduplicate_distinct_key : 0

```

```

Num_fence_key           : 0
Avg_num_value_per_key   : 0
Avg_num_value_per_deduplicate_key: 0
Num_leaf_page           : 1
Num_non_leaf_page       : 0
Num_ovf_page            : 0
Num_total_page          : 1
Height                  : 1
Avg_key_len             : 0
Avg_rec_len             : 0
Total_space              : '16.0K'
Total_used_space_non_ovf : '124.0B'
Total_free_space_non_ovf : '15.8K'
Total_used_space_ovf     : '0.0B'
Total_free_space_ovf     : '0.0B'
Avg_num_key_per_page_non_ovf : 0
Avg_free_space_per_page_non_ovf : '15.8K'
Avg_num_ovf_page_per_key : 0
Avg_free_space_per_page_ovf : '0.0B'
Max_num_ovf_page_a_key  : 0

```

SHOW CRITICAL SECTIONS

특정 데이터베이스의 전체 크리티컬 섹션(critical section, 이하 CS) 정보를 출력한다.

```
SHOW CRITICAL SECTIONS;
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Index	INT	CS 인덱스
Name	VARCHAR(32)	CS 이름
Num_holders	VARCHAR(16)	해당 CS 보유자의 개수. 다음 값 중 하나: 'N readers', '1 writer', 'none'
Num_waiting_readers	INT	읽기 대기자의 개수

칼럼 이름	타입	설명
Num_waiting_writers	INT	쓰기 대기자의 개수
Owner_thread_index	INT	CS 쓰기 소유자의 스레드 인덱스. 소유자 없으면 NULL
Owner_tran_index	INT	CS 쓰기 소유자의 트랜잭션 인덱스. 소유자 없으면 NULL
Total_enter_count	BIGINT	진입자의 전체 개수
Total_waiter_count	BIGINT	대기자의 전체 개수
Waiting_promoter_thread_index	INT	승격 대기자의 스레드 인덱스. 승격 대기자 없으면 NULL
Max_waiting_msecs	NUMERIC(10,3)	최대 대기 시간(밀리 초)
Total_waiting_msecs	NUMERIC(10,3)	전체 대기 시간(밀리초)

다음은 이 구문을 수행한 예이다.

```
SHOW CRITICAL SECTIONS;
```

```

Index  Name                               Num_holders
Num_waiting_readers Num_waiting_writers Owner_thread_index
Owner_tran_index    Total_enter_count Total_waiter_count
Waiting_promoter_thread_index Max_waiting_msecs Total_waiting_msecs
=====
=====
=====
=====
0      'ER_LOG_FILE'                     'none' 0
0      NULL                             NULL 217
0      NULL                             0.000 0.000
1      'ER_MSG_CACHE'                   'none' 0

```

```

0      NULL      NULL 0
0      NULL 0.000 0.000
0      2 'WFG'      'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      3 'LOG'      'none' 0
0      NULL      NULL 11
0      NULL 0.000      0.000
0      4 'LOCATOR_CLASSNAME_TABLE' 'none' 0
0      NULL      NULL 33
0      NULL 0.000      0.000
0      5 'QPROC_QUERY_TABLE' 'none' 0
0      NULL      NULL 3
0      NULL 0.000 0.000
0      6 'QPROC_LIST_CACHE' 'none' 0
0      NULL      NULL 1
0      NULL 0.000 0.000
0      7 'DISK_CHECK' 'none' 0
0      NULL      NULL 3
0      NULL 0.000 0.000
0      8 'CNV_FMT_LEXER' 'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      9 'HEAP_CHNGUESS' 'none' 0
0      NULL      NULL 10
0      NULL 0.000      0.000
0      10 'TRAN_TABLE' 'none' 0
0      NULL      NULL 7
0      NULL 0.000 0.000
0      11 'CT_OID_TABLE' 'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      12 'HA_SERVER_STATE' 'none' 0
0      NULL      NULL 2
0      NULL 0.000 0.000
0      13 'COMPACTDB_ONE_INSTANCE' 'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      14 'ACL'      'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      15 'PARTITION_CACHE' 'none' 0
0      NULL      NULL 1
0      NULL 0.000 0.000
0      16 'EVENT_LOG_FILE' 'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      17 'LOG_ARCHIVE' 'none' 0
0      NULL      NULL 0
0      NULL 0.000 0.000
0      18 'ACCESS_STATUS' 'none' 0

```

```
0          NULL          NULL 1
0          NULL 0.000 0.000
```

SHOW TRANSACTION TABLES

각 트랜잭션을 관리하는 데이터 구조인 트랜잭션 디스크립터(transaction descriptor)의 내부 정보를 출력한다. 유효한 트랜잭션만 출력되므로, 출력되는 트랜잭션 디스크립터의 스냅샷이 일관되지 않을 수도 있다.

```
SHOW { TRAN | TRANSACTION } TABLES [ WHERE expr ];
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Tran_index	INT	트랜잭션 테이블의 인덱스 또는 할당되지 않은 트랜잭션 슬롯일 경우 NULL 값
Tran_id	INT	트랜잭션 식별자
Is_loose_end	INT	0 : 완료된 트랜잭션일 경우 , 1 : 완료되지 않은 트랜잭션
State	VARCHAR(64)	트랜잭션의 상태. 다음 값 중 하나: 'TRAN_RECOVERY', 'TRAN_ACTIVE', 'TRAN_UNACTIVE_COMMITTED', 'TRAN_UNACTIVE_WILL_COMMIT', 'TRAN_UNACTIVE_COMMITTED_WITH_POSTPONE', 'TRAN_UNACTIVE_ABORTED', 'TRAN_UNACTIVE_UNILATERALLY_ABORTED', 'TRAN_UNACTIVE_2PC_PREPA

칼럼 이름	타입	설명
		RE', 'TRAN_UNACTIVE_2PC_COLLECTING_PARTICIPANT_VOTES', 'TRAN_UNACTIVE_2PC_ABORT_DECISION', 'TRAN_UNACTIVE_2PC_COMMIT_DECISION', 'TRAN_UNACTIVE_COMMITTED_INFORMING_PARTICIPANTS' ', 'TRAN_UNACTIVE_ABORTED_INFORMING_PARTICIPANTS';TRAN_STATE_UNKNOWN'
Isolation	VARCHAR(64)	트랜잭션의 격리 수준. 다음 중 하나: 'SERIALIZABLE', 'REPEATABLE READ', 'COMMITTED READ', 'TRAN_UNKNOWN_ISOLATION'
Wait_msecs	INT	잠금 상태로 대기 (milliseconds)
Head_lsa	VARCHAR(64)	트랜잭션 로그의 처음 주소
Tail_lsa	VARCHAR(64)	트랜잭션 로그의 마지막 주소
Undo_next_lsa	VARCHAR(64)	UNDO 트랜잭션의 다음 로그 주소
Postpone_next_lsa	VARCHAR(64)	실행 될 연기된 레코드의 다음 로그 주소

칼럼 이름	타입	설명
Savepoint_lsa	VARCHAR(64)	마지막 세이프 포인트의 로그 주소
Topop_lsa	VARCHAR(64)	마지막 최상위 동작의 로그 주소
Tail_top_result_lsa	VARCHAR(64)	마지막 부분 취소 또는 커밋의 로그 주소
Client_id	INT	클라이언트의 트랜잭션 고유 식별자
Client_type	VARCHAR(40)	클라이언트 타입. 다음 중 하나 값 'SYSTEM_INTERNAL', 'DEFAULT', 'CSQL', 'READ_ONLY_CSQL', 'BROKER', 'READ_ONLY_BROKER', 'SLAVE_ONLY_BROKER', 'ADMIN_UTILITY', 'ADMIN_CSQL', 'LOG_COPIER', 'LOG_APPLIER', 'RW_BROKER_REPLICA_ONLY', 'RO_BROKER_REPLICA_ONLY', 'SO_BROKER_REPLICA_ONLY', ADMIN_CSQL_WOS', 'UNKNOWN'
Client_info	VARCHAR(256)	클라이언트의 정보
Client_db_user	VARCHAR(40)	클라이언트의 데이터베이스 로그인 계정

칼럼 이름	타입	설명
Client_program	VARCHAR(256)	클라이언트의 프로그램명
Client_login_user	VARCHAR(16)	클라이언트를 수행 중인 OS 로그인 계정
Client_host	VARCHAR(64)	클라이언트의 호스트명
Client_pid	INT	클라이언트의 프로세스 id
Topop_depth	INT	최상위 동작의 단계
Num_unique_btrees	INT	unique_stat_info 배열에 포함된 고유한 btree 의 개수
Max_unique_btrees	INT	unique_stat_info_array 의 크기
Interrupt	INT	수행 중인 트랜잭션의 인터럽트 유무, 0 : 무, 1 : 유
Num_transient_classnames	INT	트랜잭션에 의해 임시 생성되는 클래스의 개수
Repl_max_records	INT	복제 레코드 배열의 크기
Repl_records	VARCHAR(20)	복제 레코드 버퍼 배열, 주소 포인터를 0x12345678 처럼 나타냄, NULL은 0x00000000을 의미함
Repl_current_index	INT	복제 레코드의 현재 위치

칼럼 이름	타입	설명
Repl_append_index	INT	추가 레코드의 현재 위치
Repl_flush_marked_index	INT	플러시 표시된 복제 레코드의 인덱스
Repl_insert_lsa	VARCHAR(64)	쓰기 복제의 로그 주소
Repl_update_lsa	VARCHAR(64)	갱신 복제의 로그 주소
First_save_entry	VARCHAR(20)	트랜잭션의 처음 세이브 포인트 시작점. 주소 포인터를 0x12345678 처럼 나타냄, NULL은 0x00000000 을 의미함
Tran_unique_stats	VARCHAR(20)	다중 열에 대한 로컬 통계 정보. 주소 포인터를 0x12345678 처럼 나타냄, NULL은 0x00000000 을 의미함
Modified_class_list	VARCHAR(20)	더티 클래스의 목록, 주소 포인터를 0x12345678 처럼 나타냄, NULL은 0x00000000 을 의미함
Num_temp_files	INT	임시 파일의 개수
Waiting_for_res	VARCHAR(20)	대기 리소스, 주소 포인터를 0x12345678 처럼 나타냄, NULL은 0x00000000 을 의미함

칼럼 이름	타입	설명
Has_deadlock_priority	INT	데드락 우선순위 유무, 0 : 무, 1 : 유
Suppress_replication	INT	플래그가 세팅 될 때 복제 로그 쓰기를 생략
Query_timeout	DATETIME	query_timeout 시간 내에 쿼리는 수행되어야 함. NULL일 경우 질의가 끝날 때 까지 기다림.
Query_start_time	DATETIME	질의 시작 시간, 질의 완료시 NULL
Tran_start_time	DATETIME	트랜잭션 시작 시간, 트랜잭션 완료시 NULL
Xasl_id	VARCHAR(64)	vpid:(valid pageid),vfid:(valid pageid), 질의 완료시 NULL
Disable_modifications	INT	0보다 클 경우 수정을 금지
Abort_reason	VARCHAR(40)	트랜잭션 중지 사유, 다음 중 하나 'NORMAL', 'ABORT_DUE_TO_DEADLOCK', 'ABORT_DUE_ROLLBACK_ON_ESCALATION'

다음은 이 구문을 수행한 예이다.

```
SHOW TRAN TABLES WHERE CLIENT_TYPE = 'CSQL';
```

=== <Result of SELECT Command in Line 1> ===

```

<00001> Tran_index           : 1
        Tran_id             : 58
        Is_loose_end         : 0
        State                : 'ACTIVE'
        Isolation            : 'COMMITTED READ'
        Wait_msecs           : -1
        Head_lsa             : '(-1|-1)'
        Tail_lsa             : '(-1|-1)'
        Undo_next_lsa        : '(-1|-1)'
        Postpone_next_lsa    : '(-1|-1)'
        Savepoint_lsa        : '(-1|-1)'
        Topop_lsa            : '(-1|-1)'
        Tail_top_result_lsa   : '(-1|-1)'
        Client_id            : 108
        Client_type          : 'CSQL'
        Client_info          : ''
        Client_db_user       : 'PUBLIC'
        Client_program        : 'csql'
        Client_login_user    : 'cubrid'
        Client_host           : 'cubrid001'
        Client_pid           : 13190
        Topop_depth          : 0
        Num_unique_btrees     : 0
        Max_unique_btrees     : 0
        Interrupt            : 0
        Num_transient_classnames : 0
        Repl_max_records      : 0
        Repl_records         : NULL
        Repl_current_index    : 0
        Repl_append_index     : -1
        Repl_flush_marked_index : -1
        Repl_insert_lsa       : '(-1|-1)'
        Repl_update_lsa       : '(-1|-1)'
        First_save_entry      : NULL
        Tran_unique_stats     : NULL
        Modified_class_list   : NULL
        Num_temp_files        : 0
        Waiting_for_res       : NULL
        Has_deadlock_priority : 0
        Suppress_replication  : 0
        Query_timeout         : NULL
        Query_start_time      : 03:10:11.425 PM 02/04/2016
        Tran_start_time      : 03:10:11.425 PM 02/04/2016
        Xasl_id               : 'vpid: (32766|50), vfid: (32766|
43)'
        Disable_modifications : 0
        Abort_reason          : 'NORMAL'

```

SHOW THREADS

각 스레드의 내부 정보를 출력한다. 반환 결과는 “Index” 칼럼의 오름차순으로 정렬되며, 출력되는 스레드 엔트리의 스냅샷이 일관되지 않을 수도 있다. SA MODE일 경우 이 구문은 아무런 결과도 출력하지 않는다.

```
SHOW THREADS [ WHERE expr ];
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼명	타입	설명
Index	INT	쓰레드 시작 인덱스
Jobq_index	INT	워커 쓰레드의 작업 큐 인덱스. 워커 쓰레드가 아닌 경우 NULL
Thread_id	BIGINT	쓰레드 식별자
Tran_index	INT	쓰레드가 속한 트랜잭션 인덱스. 관련 쓰레드가 없을 경우 NULL
Type	VARCHAR(8)	쓰레드 종류. 다음 중 하나 'MASTER', 'SERVER', 'WORKER', 'DAEMON', 'VACUUM_MASTER', 'VACUUM_WORKER', 'NONE', 'UNKNOWN'.
Status	VARCHAR(8)	쓰레드 상태. 다음 중 하나 'FREE', 'RUN', 'WAIT', 'CHECK'.
Resume_status	VARCHAR(32)	재시작 상태. 다음 중 하나 'RESUME_NONE', 'RESUME_DUE_TO_INTERRUPT',

칼럼명	타입	설명
		'RESUME_DUE_TO_SHUTDOWN', 'PGBUF_SUSPENDED', 'PGBUF_RESUMED', 'JOB_QUEUE_SUSPENDED', 'JOB_QUEUE_RESUMED', 'CSECT_READER_SUSPENDED', 'CSECT_READER_RESUMED', 'CSECT_WRITER_SUSPENDED', 'CSECT_WRITER_RESUMED', 'CSECT_PROMOTER_SUSPENDED', 'CSECT_PROMOTER_RESUMED', 'CSS_QUEUE_SUSPENDED', 'CSS_QUEUE_RESUMED', 'QMGR_ACTIVE_QRY_SUSPENDED', 'QMGR_ACTIVE_QRY_RESUMED', 'QMGR_MEMBUF_PAGE_SUSPENDED', 'QMGR_MEMBUF_PAGE_RESUMED', 'HEAP_CLSREPR_SUSPENDED', 'HEAP_CLSREPR_RESUMED', 'LOCK_SUSPENDED', 'LOCK_RESUMED', 'LOGWR_SUSPENDED', 'LOGWR_RESUMED'
Net_request	VARCHAR(64)	net_requests 배열의 요청 이름, 예: 'LC_ASSIGN_OID'. 요청 이름이 없을 경우 NULL
Conn_client_id	INT	쓰레드에 응답하는 클라이언트의 식별자, 클라이언트의 식별자가 없을 경우 NULL

칼럼명	타입	설명
Conn_request_id	INT	쓰레드가 처리하고 있는 요청의 식별자, 요청 식별자가 없을 경우 NULL
Conn_index	INT	연결 인덱스, 없을 경우 NULL
Last_error_code	INT	마지막 에러 코드
Last_error_msg	VARCHAR(256)	마지막 에러 메시지, 메시지가 256 자 보다 클 경우 일부만 보인다. 에러 메시지가 없을 경우 NULL
Private_heap_id	VARCHAR(20)	쓰레드 내부 메모리 할당자의 주소, 예: 0x12345678. 관련 힙 id 가 없을 경우 NULL
Query_entry	VARCHAR(20)	QMGR_QUERY_ENTRY의 주소 , 예: 0x12345678, 연관된 QMGR_QUERY_ENTRY 가 없을 경우 NULL.
Interrupted	INT	요청/트랜잭션의 인터럽트 유/무 0 또는 1
Shutdown	INT	서버의 중지 진행 여/부, 0 또는 1
Check_interrupt	INT	0 또는 1
Wait_for_latch_promote	INT	0 또는 1, 쓰레드가 래치 프로모션(latch promotion)을 대기하는 여/부

칼럼명	타입	설명
Lockwait_blocked_mode	VARCHAR(24)	잠금대기 블록 모드, 다음 중 하나. 'NULL_LOCK', 'IS_LOCK', 'S_LOCK', 'IS_LOCK', 'IX_LOCK', 'SIX_LOCK', 'X_LOCK', 'SCH_M_LOCK', 'UNKNOWN'
Lockwait_start_time	DATETIME	차단이 시작된 시간, 차단 상태 아닌 경우 NULL
Lockwait_msecs	INT	차단되었던 시간 (milliseconds), 차단된 상태가 아닌 경우 NULL
Lockwait_state	VARCHAR(24)	잠금 대기 상태 예: 'SUSPENDED', 'RESUMED', 'RESUMED_ABORTED_FIRST', 'RESUMED_ABORTED_OTHER', 'RESUMED_DEADLOCK_TIMEOUT', 'RESUMED_TIMEOUT', 'RESUMED_INTERRUPT'. 블록된 상태가 없을 경우 NULL
Next_wait_thread_index	INT	다음 대기 스레드 인덱스, 없을 경우 NULL
Next_tran_wait_thread_index	INT	잠금 매니저의 다음 대기 스레드 인덱스, 없을 경우 NULL
Next_worker_thread_index	INT	css_job_queue.worker_thrd_list 의 다음 워커 스레드 인덱스, 없을 경우 NULL

다음은 이 구문을 수행한 예이다.

```
SHOW THREADS WHERE RESUME_STATUS != 'RESUME_NONE' AND STATUS !=
'FREE';
```

```
=== <Result of SELECT Command in Line 1> ===
```

```
<00001> Index : 183
        Jobq_index : 3
        Thread_id : 140077788813056
        Tran_index : 3
        Type : 'WORKER'
        Status : 'RUN'
        Resume_status : 'JOB_QUEUE_RESUMED'
        Net_request : 'QM_QUERY_EXECUTE'
        Conn_client_id : 108
        Conn_request_id : 196635
        Conn_index : 3
        Last_error_code : 0
        Last_error_msg : NULL
        Private_heap_id : '0x02b3de80'
        Query_entry : '0x7f6638004cb0'
        Interrupted : 0
        Shutdown : 0
        Check_interrupt : 1
        Wait_for_latch_promote : 0
        Lockwait_blocked_mode : NULL
        Lockwait_start_time : NULL
        Lockwait_msecs : NULL
        Lockwait_state : NULL
        Next_wait_thread_index : NULL
        Next_tran_wait_thread_index : NULL
        Next_worker_thread_index : NULL
<00002> Index : 192
        Jobq_index : 2
        Thread_id : 140077779339008
        Tran_index : 2
        Type : 'WORKER'
        Status : 'WAIT'
        Resume_status : 'LOCK_SUSPENDED'
        Net_request : 'LC_FIND_LOCKHINT_CLASSOIDS'
        Conn_client_id : 107
        Conn_request_id : 131097
        Conn_index : 2
        Last_error_code : 0
        Last_error_msg : NULL
        Private_heap_id : '0x02bcdcf10'
        Query_entry : NULL
        Interrupted : 0
        Shutdown : 0
        Check_interrupt : 1
        Wait_for_latch_promote : 0
        Lockwait_blocked_mode : 'SCH_S_LOCK'
```



```

Lockwait_start_time      : 10:47:45.000 AM 02/03/2016
Lockwait_msecs           : -1
Lockwait_state           : 'SUSPENDED'
Next_wait_thread_index    : NULL
Next_tran_wait_thread_index: NULL
Next_worker_thread_index  : NULL

```

SHOW JOB QUEUES

작업 큐의 상태를 보여준다. SA MODE일 때에 이 문은 아무 결과도 보여주지 않는다.

```
SHOW JOB QUEUES;
```

이 질의는 다음의 칼럼들을 출력한다:

칼럼명	타입	설명
Jobq_index	INT	작업 큐의 인덱스
Num_total_workers	INT	큐의 워커 쓰레드 총 개수
Num_busy_workers	INT	큐의 활성 워커 쓰레드의 개수
Num_connection_workers	INT	큐의 연결(connection) 워커 쓰레드의 수

SHOW PAGE BUFFER STATUS

데이터 페이지 버퍼의 상태를 출력한다.

```
SHOW PAGE BUFFER STATUS;
```

해당 구문은 다음의 칼럼을 출력한다.

칼럼 이름	타입	설명
Hit_rate	NUMERIC(13,10)	

칼럼 이름	타입	설명
		데이터 버퍼의 페이지 적중률 (사용 중단(deprecated), 항상 NULL)
Num_hit	BIGINT	데이터 버퍼의 페이지 적중 수 (사용 중단(deprecated), 항상 NULL)
Num_page_request	BIGINT	데이터 버퍼에 페이지 요청 수 (사용 중단(deprecated), 항상 NULL)
Pool_size	INT	데이터 버퍼의 전체 페이지 수
Page_size	INT	데이터 버퍼의 단일 페이지 크기
Free_pages	INT	데이터 버퍼의 여유 페이지 수
Victim_candidate_pages	INT	데이터 버퍼의 희생자 (victim) 후보 페이지 수
Clean_pages	INT	데이터 버퍼의 갱신되지 않은 페이지 수
Dirty_pages	INT	데이터 버퍼의 갱신된 페이지 수
Num_index_pages	INT	데이터 버퍼의 인덱스 페이지 수

칼럼 이름	타입	설명
Num_data_pages	INT	데이터 버퍼의 데이터 페이지 수
Num_system_pages	INT	데이터 버퍼의 시스템 페이지 수
Num_temp_pages	INT	데이터 버퍼의 임시 페이지 수
Num_pages_created	BIGINT	데이터 버퍼에서 새롭게 생성된 페이지 수 (사용 중단 (deprecated), 항상 NULL)
Num_pages_written	BIGINT	데이터 버퍼에서 디스크로 쓰여진 페이지 수 (사용 중단 (deprecated), 항상 NULL)
Pages_written_rate	NUMERIC(20,10)	데이터 버퍼에서 디스크로 초당 쓰여진 페이지 수 (사용 중단 (deprecated), 항상 NULL)
Num_pages_read	BIGINT	데이터 버퍼로 디스크에서 읽은 페이지 수 (사용 중단 (deprecated), 항상 NULL)
Pages_read_rate	NUMERIC(20,10)	데이터 버퍼로 디스크에서 초당 읽은 페이지 수 (사용 중단 (deprecated), 항상 NULL)
Num_flusher_waiting_threads	INT	데이터 버퍼의 페이지 할당을 대기하는 쓰레드 수 (사용 중단 (deprecated), 항상 NULL)

참고

Hit_rate, **Num_hit**, **Num_page_request**, **Num_pages_created**, **Num_pages_written**, **Pages_written_rate**, **Num_pages_read**, **Pages_read_rate**, **Num_flusher_waiting_threads** 칼럼은 11.5 버전부터 사용이 중단(deprecated)되어 항상 NULL을 반환한다. 칼럼 자체는 제거되지 않으므로 결과셋의 칼럼 개수, 이름, 타입, 순서는 이전 버전과 동일하다.

이 중 다음 4개 칼럼은 **cubrid statdump**의 통계 항목으로 대체할 수 있다. 각 항목에 대한 자세한 설명은 [statdump](#)을 참고한다.

- **Hit_rate**: **Data_page_buffer_hit_ratio** (소수점 둘째 자리까지 출력된다)
- **Num_page_request**: **Num_data_page_fetches**
- **Num_pages_read**: **Num_data_page_ireads**
- **Num_pages_written**: **Num_data_page_iowrites**

단, 사용이 중단되기 전의 이 칼럼들은 직전 **SHOW PAGE BUFFER STATUS** 실행 이후의 증분값이었던 반면 **cubrid statdump**의 통계 항목은 누적값이므로, 같은 구간의 값을 얻으려면 **cubrid statdump -i SECOND**로 주기별 값을 출력하거나 두 시점의 누적값 차이를 사용한다.

나머지 5개 칼럼은 **cubrid statdump**에 대응하는 항목이 없다. 이 중 **Num_hit**는 **Num_data_page_fetches**에서 **Num_data_page_ireads**를 뺀 값으로, **Pages_written_rate**와 **Pages_read_rate**는 위의 주기별 출력값을 해당 주기로 나누어 간접적으로 구할 수 있다. **Num_pages_created**와 **Num_flusher_waiting_threads**는 대체할 수 있는 통계 항목이 없다.

사용이 중단되지 않은 칼럼(**Pool_size**, **Page_size**, **Free_pages**, **Victim_candidate_pages**, **Clean_pages**, **Dirty_pages**, **Num_index_pages**, **Num_data_pages**, **Num_system_pages**, **Num_temp_pages**)은 데이터 버퍼의 페이지 구성 정보로, 계속 정상 값을 출력한다.

다음은 이 구문을 수행한 예이다.

```
-- csql> ;line on
SHOW PAGE BUFFER STATUS;
```

```
<00001> Hit_rate           : NULL
        Num_hit           : NULL
        Num_page_request  : NULL
        Pool_size         : 32768
        Page_size         : 16392
        Free_pages        : 32739
```

```

Victim_candidate_pages      : 0
Clean_pages                 : 32767
Dirty_pages                 : 1
Num_index_pages             : 2
Num_data_pages              : 15
Num_system_pages            : 12
Num_temp_pages              : 0
Num_pages_created           : NULL
Num_pages_written           : NULL
Pages_written_rate          : NULL
Num_pages_read              : NULL
Pages_read_rate             : NULL
Num_flusher_waiting_threads: NULL

```

사용자 스키마

스키마는 데이터베이스 객체의 논리적인 모음이다. 스키마 이름을 통해 객체를 식별할 수 있다는 것이며, 스키마 객체를 물리적으로 분리된 저장공간에 저장하지 않는다. 객체는 하나의 스키마에만 존재할 수 있으며 동시에 다른 스키마에 존재할 수 없다. 스키마는 사용자가 생성할 수 없으며, 데이터베이스 사용자 생성 시 해당 사용자는 사용자 이름과 동일한 이름의 단일 스키마를 소유하게 된다. 스키마에는 테이블, 시리얼, 트리거, 동의어 및 저장 프로시저/함수 가 포함된다. 스키마의 객체는 동일한 스키마 내에서 고유한 이름을 가져야 하지만 다른 스키마의 객체와 중복되는 이름을 가질 수 있다.

객체에 접근하려면 사용자는 “스키마 이름.테이블 이름”의 경로 표현식을 사용해야 한다. 사용자의 스키마가 객체의 스키마와 (소유자) 동일한 경우 경로 표현식에서 스키마 이름을 생략할 수 있다. 스키마 이름을 생략하면 해당 사용자의 스키마 이름이 사용된다. “스키마 이름.테이블 이름”의 경로 표현식을 사용하더라도 사용자는 해당 객체를 사용할 수 있는 권한이 있어야 한다. .. note:

11.2 버전 이후부터 "스키마 ≡ 데이터베이스" 개념이 "스키마 ≡ 사용자"로 변경되었다. 11.2 버전 이전에는 데이터베이스에 단일 스키마만 가능했지만 11.2 버전부터는 사용자 별로 스키마를 사용할 수 있다. 따라서 테이블 이름만으로는 다른 사용자가 소유한 테이블에 접근할 수 없으며, 스키마 (소유자) 이름을 접두사로 붙여서 사용해야 해당 테이블에 접근할 수 있다.

기본 예제 데이터베이스인 demodb에서는 다음과 같은 차이점이 있다.

```
.. code-block:: shell
```

```
    csql -u dba demodb
```

```
.. code-block:: sql
```

```
    SELECT name FROM athlete LIMIT 1;
```

```
    ERROR: before ' ; '
```

```
Unknown class "dba.athlete".

SELECT name FROM public.athlete LIMIT 1;

      name
=====
'Fernandez Jesus'
```

테이블, 시리얼, 트리거, 동의어 및 저장 프로시저/함수를 관리하는 시스템 테이블에 (_db_class, _db_serial, _db_trigger, _db_synonym, _db_stored_procedure) unique_name 칼럼이 추가되었다. unique_name 칼럼은 스키마 이름이 접두사로 붙은 이름을 저장한다. _db_class의 unique_name 칼럼에서 시스템 테이블 이름은 스키마 이름이 접두사로 붙지 않는다.

```
csql -u public demodb
```

```
CREATE TABLE table_1 (column_1 INTEGER);
CREATE TABLE table_2 (column_1 INTEGER);
CREATE TRIGGER trigger_1 AFTER INSERT ON table_1 EXECUTE INSERT INTO
table_2 VALUES (obj.column_1);
```

```
csql -u dba demodb
```

```
SELECT unique_name, class_name, owner.name FROM _db_class ORDER BY
unique_name;
```

unique_name	class_name	owner.name
=====	=====	=====
'_db_attribute'	'_db_attribute'	'DBA'
'_db_auth'	'_db_auth'	'DBA'
'_db_charset'	'_db_charset'	'DBA'
'_db_class'	'_db_class'	'DBA'
'_db_collation'	'_db_collation'	'DBA'
...		
'db_attr_setdomain_elm'	'db_attr_setdomain_elm'	'DBA'
'db_attribute'	'db_attribute'	'DBA'
'db_auth'	'db_auth'	'DBA'
'db_authorization'	'db_authorization'	'DBA'
...		
'public.athlete'	'athlete'	'PUBLIC'
'public.code'	'code'	'PUBLIC'
'public.event'	'event'	'PUBLIC'
'public.game'	'game'	'PUBLIC'
'public.history'	'history'	'PUBLIC'

```
SELECT unique_name, name, owner.name FROM _db_serial ORDER BY
unique_name;
```

```

        unique_name          name          owner.name
=====
        'public.athlete_ai_code'  'athlete_ai_code'  'PUBLIC'
        'public.event_no'        'event_no'        'PUBLIC'
        'public.stadium_no'      'stadium_no'      'PUBLIC'

SELECT unique_name, name, owner.name FROM _db_trigger ORDER BY
unique_name;

        unique_name          name          owner.name
=====
        'public.trigger_1'  'trigger_1'  'PUBLIC'

SELECT unique_name, name, owner.name, target_unique_name,
target_name, target_owner.name FROM _db_synonym ORDER BY unique_name;

        unique_name          name          owner.name
target_unique_name          target_name          target_owner.name
=====
=====
        'public.synonym_1'  'synonym_1'  'PUBLIC'
        'public.synonym_target_1'  'synonym_target_1'  'PUBLIC'

SELECT unique_name, sp_name, owner.name FROM _db_stored_procedure
WHERE is_system_generated = 0 ORDER BY unique_name;

        unique_name          sp_name          owner.name
=====
        'public.stored_procedure_1'  'stored_procedure_1'  'PUBLIC'

```

시스템 카탈로그

시스템 카탈로그 가상 클래스(virtual class)를 사용하면 다양한 스키마 정보를 SQL 문을 이용하여 쉽게 얻어낼 수 있다. 예를 들어 다음과 같은 스키마 정보들을 카탈로그 가상 클래스를 이용하여 얻을 수 있다.

```

-- 클래스 '_db_user'를 참조하는 클래스들
SELECT class_name
FROM db_attribute
WHERE domain_class_name = '_db_user';

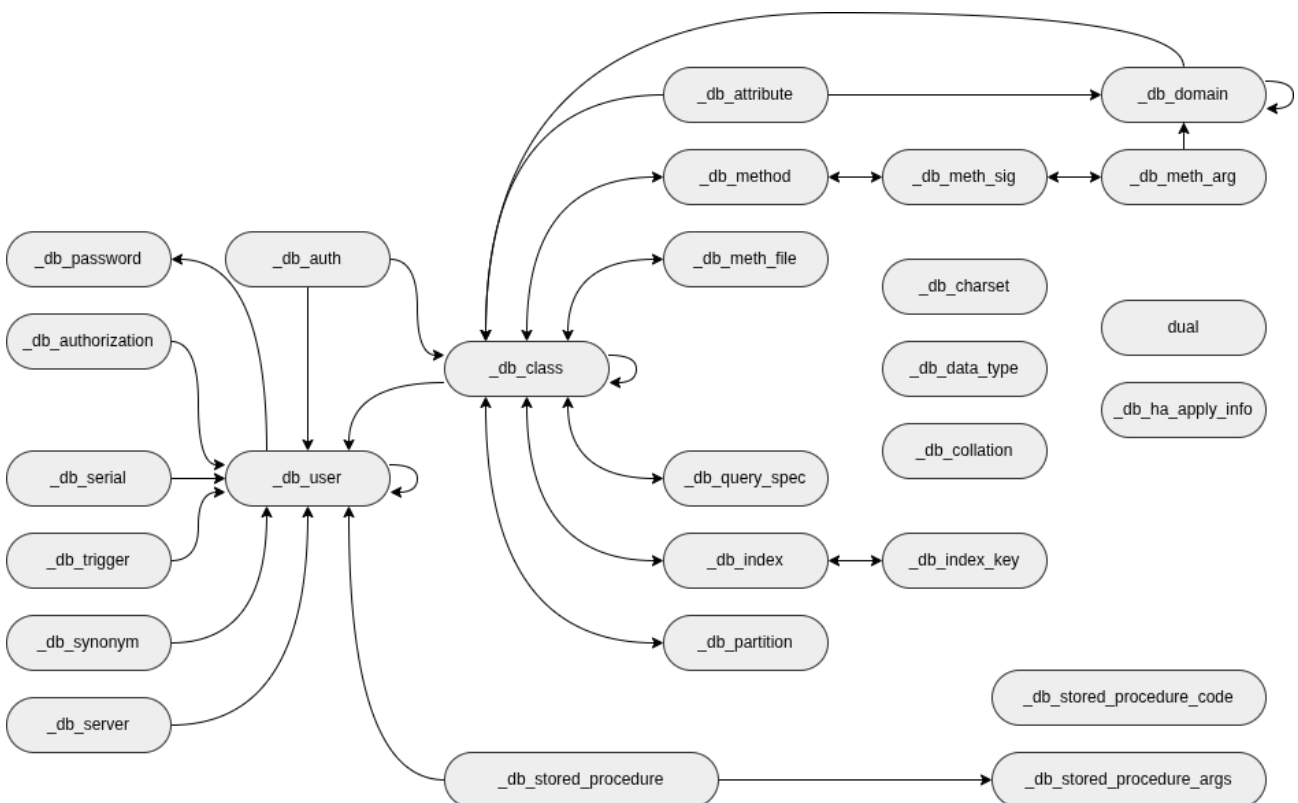
-- 현재 사용자가 접근 가능한 클래스 개수
SELECT COUNT(*)
FROM db_class;

```

```
-- 클래스 '_db_user'의 속성
SELECT attr_name, data_type
FROM db_attribute
WHERE class_name = '_db_user';
```

시스템 카탈로그 클래스

카탈로그 가상 클래스를 정의하기 위해 우선 카탈로그 클래스를 정의한다. 아래 그림은 추가되는 카탈로그 클래스들과 그 관계를 나타낸다. 화살표는 참조 관계이며, '_'로 시작하는 클래스는 카탈로그 클래스이다.



추가한 카탈로그 클래스들은 데이터베이스 안의 모든 클래스, 속성(attribute), 메서드 등에 대한 정보를 표현한다. 카탈로그 클래스들은 클래스 합성 계층(class composition hierarchy)으로 구성되어 있는데, 상호 참조가 가능하도록 카탈로그 클래스 인스턴스들의 OID를 가지도록 구성되어 있다.

클래스 이름	설명
_db_class	클래스에 대한 정보
_db_attribute	클래스의 속성에 대한 정보
_db_domain	도메인에 대한 정보

클래스 이름	설명
<code>_db_charset</code>	문자셋에 대한 정보
<code>_db_collation</code>	콜레이션에 대한 정보
<code>_db_method</code>	메서드에 대한 정보
<code>_db_meth_sig</code>	메서드에 대한 C 함수의 구성 정보
<code>_db_meth_arg</code>	메서드의 인자에 대한 정보
<code>_db_meth_file</code>	메서드가 정의된 파일 정보
<code>_db_query_spec</code>	가상 클래스의 SQL 정의문
<code>_db_index</code>	인덱스에 대한 정보
<code>_db_index_key</code>	인덱스에 대한 키 정보
<code>_db_auth</code>	클래스에 대한 사용자 권한 정보
<code>_db_data_type</code>	데이터 타입에 대한 정보
<code>_db_partition</code>	분할에 대한 정보
<code>_db_stored_procedure</code>	저장 프로시저에 대한 정보
<code>_db_stored_procedure_args</code>	저장 프로시저 인자에 대한 정보
<code>_db_stored_procedure_code</code>	저장 프로시저 코드에 대한 정보

클래스 이름	설명
<code>_db_server</code>	DBLink에 대한 서버 정보
<code>_db_synonym</code>	동의어에 대한 대상 객체 정보
<code>_db_user</code>	사용자에 대한 정보
<code>_db_authorization</code>	사용자별 권한 요약
<code>_db_serial</code>	시리얼에 대한 정보
<code>_db_trigger</code>	트리거에 대한 정보
<code>_db_ha_apply_info</code>	applylogdb 유틸리티가 복제 로그를 적용하는 진행 상태
<code>dual</code>	더미 테이블

`_db_class`

클래스에 대한 정보를 표현하며 `unique_name`에 대한 인덱스와 `class_name`, `owner`에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
<code>class_of</code>	<code>object</code>	클래스 객체로서 시스템 내부에 저장된 클래스에 대한 메타 정보 객체를 의미한다.
<code>unique_name</code>	<code>VARCHAR(255)</code>	스키마명을 포함하는 클래스명

속성명	데이터 타입	설명
class_name	VARCHAR(255)	클래스명
class_type	INTEGER	클래스이면 0, 가상 클래스이면 1
is_system_class	INTEGER	사용자가 정의한 클래스이면 0, 시스템 클래스이면 1
owner	_db_user	클래스 소유자
inst_attr_count	INTEGER	인스턴스 속성의 개수
class_attr_count	INTEGER	클래스 속성의 개수
shared_attr_count	INTEGER	공유 속성의 개수
inst_meth_count	INTEGER	인스턴스 메서드의 개수
class_meth_count	INTEGER	클래스 메서드의 개수
collation_id	INTEGER	콜레이션 식별자
tde_algorithm	INTEGER	TDE 암호화 알고리즘 0: NONE, 1: AES, 2: ARIA
statistics_strategy	INTEGER	통계 정보 수집 전략 (0: SAMPLING, 1: FULLSCAN)
flags	INTEGER	클래스 플래그 • Bit 1: WITH CHECK OPTION 속성이 있으면 1

속성명	데이터 타입	설명
		• Bit 2: LOCAL CHECK OPTION 속성이 있으면 1 • Bit 3: REUSE_OID 클래스인 경우 1 • 나머지 비트는 현재 사용하지 않는다.
sub_classes	SEQUENCE OF _db_class	한 단계 하위 클래스
super_classes	SEQUENCE OF _db_class	한 단계 상위 클래스
inst_attrs	SEQUENCE OF _db_attribute	인스턴스 속성
class_attrs	SEQUENCE OF _db_attribute	클래스 속성
shared_attrs	SEQUENCE OF _db_attribute	공유 속성
inst_meths	SEQUENCE OF _db_method	인스턴스 메서드
class_meths	SEQUENCE OF _db_method	클래스 메서드
meth_files	SEQUENCE OF _db_meth_file	메서드에 대한 함수가 위치한 파일 경로
query_specs	SEQUENCE OF _db_query_spec	가상 클래스인 경우 그 SQL 정의문
indexes	SEQUENCE OF _db_index	클래스에 생성된 인덱스
comment	VARCHAR(2048)	클래스 설명

속성명	데이터 타입	설명
partition	SEQUENCE of _db_partition	파티션 정보
created_time	DATETIME	클래스 생성 시각
updated_time	DATETIME	클래스 수정 시각
checked_time	DATETIME	클래스의 통계 정보 갱신 시각

다음 예제에서는 사용자 **PUBLIC** 이 소유하고 있는 클래스의 모든 하위 클래스를 검색한다(결과로 보이는 하위 클래스 *female_event* 는 **ADD SUPERCLASS 절** 의 예제를 참조한다).

```
SELECT class_name, SEQUENCE(SELECT class_name FROM _db_class s WHERE
s IN c.sub_classes)
FROM _db_class c
WHERE c.owner.name = 'PUBLIC' AND c.sub_classes IS NOT NULL;
```

```
class_name          sequence((select class_name from _db_class s
where s in c.sub_classes))
=====
'event'             {'female_event'}
```

참고

시스템 카탈로그 클래스에 대한 모든 예제는 csql 프로그램에서 작성되었는데, 여기에서는 **--no-auto-commit** (auto-commit 모드 비활성화), **-u** (사용자 dba를 명시) 옵션을 사용하였다.

```
% csql --no-auto-commit -u dba demodb
```

_db_attribute

속성에 대한 정보를 표현하며 class_of, attr_name 및 attr_type 에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
class_of	_db_class	속성이 속한 클래스.
attr_name	VARCHAR(255)	속성명.
attr_type	INTEGER	속성이 정의된 타입. 인스턴스 속성이면 0, 클래스 속성이면 1, 공유 속성이면 2이다.
from_class_of	_db_class	상속받은 속성이면 그 속성이 정의되어 있는 상위 클래스가 설정되며, 상속받지 않은 것이면 NULL 이다
from_attr_name	VARCHAR(255)	상속받은 속성이며 이름 충돌 (name conflict)이 발생하여 이를 해결하기 위해 그 속성명이 바뀐 경우, 상위 클래스에 정의된 원래 이름이 설정된다. 그 이외에는 모두 NULL 이 설정된다.
def_order	INTEGER	속성이 클래스에 정의된 순서로 0부터 시작한다. 상속받은 속성이면 그 상위 클래스에서 정의된 순서를 따른다. 예를 들어, 클래스 y가 클래스 x로부터 속성 a를 상속받고 a는 x에서 첫 번째로 정의되었을 때 def_order는 0이 된다.
data_type	INTEGER	속성의 데이터 타입. 아래의 'CUBRID가 지원하는 데이터 타입' 표에서 명시하는 value 중 하나이다.

속성명	데이터 타입	설명
default_value	VARCHAR(255)	기본값. 데이터 타입에 관계없이 모두 문자열로 저장된다. 기본값이 없으면 NULL , 기본값이 NULL 이면 'NULL'로 표현된다. 데이터 타입이 객체 타입이면 'volume id page id slot id', 집합 타입이면 '{element 1, element 2, ...}'로 표현된다.
domains	SEQUENCE OF _db_domain	데이터 타입에 대한 도메인 정보.
is_nullable	INTEGER	not null 제약이 설정되어 있으면 0, 그렇지 않으면 1이 설정된다.
comment	VARCHAR(2048)	속성에 대한 설명.

CUBRID가 지원하는 데이터 타입

값	의미	값	의미
1	INTEGER	23	BIT
2	FLOAT	24	VARBIT
3	DOUBLE	25	CHAR
4	STRING	27	VARCHAR

값	의미	값	의미
5	OBJECT	31	BIGINT
6	SET	32	DATETIME
7	MULTISET	33	BLOB
8	SEQUENCE	34	CLOB
9	ELO	35	ENUM
10	TIME	36	TIMESTAMP TZ
11	TIMESTAMP	37	TIMESTAMP LTZ
12	DATE	38	DATETIME TZ
18	SHORT	39	DATETIME LTZ
22	NUMERIC	40	JSON

CUBRID가 지원하는 문자셋

값	의미
0	US English - ASCII encoding
2	Binary
3	Latin 1 - ISO 8859 encoding

값	의미
4	KSC 5601 1990 - EUC encoding
5	UTF8 - UTF8 encoding

다음 예제에서는 사용자 **PUBLIC** 이 소유하고 있는 클래스 중에서 사용자 클래스 (from_class_of.is_system_class = 0)인 것을 검색한다(결과로 보이는 *female_event* 클래스는 **ADD SUPERCLASS 절** 의 예제를 참조한다).

```
SELECT class_of.class_name, attr_name
FROM _db_attribute
WHERE class_of.owner.name = 'PUBLIC' AND
from_class_of.is_system_class = 0
ORDER BY 1, def_order;
```

```
class_of.class_name  attr_name
=====
'female_event'      'code'
'female_event'      'sports'
'female_event'      'name'
'female_event'      'gender'
'female_event'      'players'
```

_db_domain

도메인에 대한 정보이며, object_of와 data_type 컬럼에 각각(개별적인) 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
object_of	object	도메인을 참조하는 속성, 메서드 인자 또는 도메인
data_type	INTEGER	도메인의 데이터 타입(_db_attribute 의 'CUBRID가 지원하는 데이터 타입' 표의 '값' 중 하나)

속성명	데이터 타입	설명
prec	INTEGER	데이터 타입에 대한 전체 자릿수(precision). 전체 자릿수가 명시되지 않은 경우 0이 설정됨
scale	INTEGER	데이터 타입에 대한 소수점 이하의 자릿수(scale). 소수점 이하의 자릿수가 명시되지 않은 경우 0이 설정됨
class_of	_db_class	데이터 타입이 객체 타입인 경우 그 도메인 클래스. 객체 타입이 아닌 경우 NULL 이 설정됨.
code_set	INTEGER	문자열 타입인 경우, 문자셋(_db_attribute 의 'CUBRID가 지원하는 문자셋' 표의 '값' 중 하나). 문자 스트링 타입이 아닌 경우 0.
collation_id	INTEGER	콜레이션 ID
enumeration	SEQUENCE OF CHARACTER VARYING(1073741823)	열거형 타입 정의 문자열
set_domains	SEQUENCE OF _db_domain	컬렉션 타입인 경우, 그 집합을 구성하는 원소의 데이터 타입에 대한 도메인 정보. 컬렉션 타입이 아닌 경우 NULL 이 설정됨
json_schema	CHARACTER VARYING(1073741823)	JSON 타입인 경우, JSON 데이터 구조를 정의하는 스키마 정

속성명	데이터 타입	설명
		보. JSON 타입이 아닌 경우 NULL 이 설정됨

_db_charset
문자셋에 대한 정보이다.

속성명	데이터 타입	설명
charset_id	INTEGER	문자셋 식별자
charset_name	CHARACTER VARYING(32)	문자셋 이름
default_collation	INTEGER	기본 콜레이션 식별자
char_size	INTEGER	한 문자의 바이트 크기

_db_collation
콜레이션(collation)에 대한 정보.

속성명	데이터 타입	설명
coll_id	INTEGER	콜레이션 식별자
coll_name	VARCHAR(32)	콜레이션 이름
charset_id	INTEGER	문자셋 식별자
built_in	INTEGER	

속성명	데이터 타입	설명
		제품 설치 시 콜레이션 포함 여부
expansions	INTEGER	확장 지원 여부 (0: 지원 안 함, 1: 지원)
contractions	INTEGER	축약 지원 여부 (0: 지원 안 함, 1: 지원)
uca_strength	INTEGER	가중치 세기(weight strength)
checksum	VARCHAR(32)	콜레이션 파일의 체크섬

_db_method

메서드에 대한 정보이며 class_of, meth_name에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
class_of	_db_class	메서드가 속한 클래스
meth_name	VARCHAR(255)	메서드 이름
meth_type	INTEGER	메서드가 클래스에 정의된 타입. 인스턴스 메서드이면 0, 클래스 메서드이면 1
from_class_of	_db_class	메서드가 상속된 것이면 그 메서드가 정의되어 있는 상위 클래스가 설정되며 그렇지 않으면 NULL

속성명	데이터 타입	설명
from_meth_name	VARCHAR(255)	상속받은 메서드이며 이름 충돌이 발생하여 이를 해결하기 위해 그 메서드명이 바뀐 경우, 상위 클래스에 정의된 원래 이름이 설정됨. 그 이외에는 모두 NULL
signatures	SEQUENCE OF _db_meth_sig	메서드 호출시 수행하는 C 함수에 대한 구성 정보

다음 예제에서는 사용자 **DBA** 가 소유하고 있는 클래스 중에서 클래스 메서드가 있는 것 (`c.class_meth_count > 0`)의 클래스 메서드를 검색한다.

```
SELECT class_name, SEQUENCE(SELECT meth_name
                             FROM _db_method m
                             WHERE m in c.class_meths)
FROM _db_class c
WHERE c.owner.name = 'DBA' AND c.class_meth_count > 0
ORDER BY 1;
```

```
class_name      sequence((select meth_name from _db_method m
where m in c.class_meths))
=====
'_db_serial'    {'change_serial_owner'}
'_db_user'      {'set_password', 'set_password_encoded',
'set_password_encoded_sha1', 'add_member',
'drop_member', 'print_authorizations',
'add_user', 'drop_user', 'find_user', 'login'}
'db_root'       {'add_user', 'drop_user', 'find_user',
'print_authorizations', 'info', 'change_owner',
'change_trigger_owner', 'get_owner',
'change_sp_owner'}
'db_user'       {'find_user', 'login'}
```

_db_meth_sig

메서드에 대한 C 함수의 구성 정보이며 meth_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
meth_of	_db_method	함수 정보에 대한 메서드
func_name	VARCHAR(255)	함수명
arg_count	INTEGER	함수의 입력인자 개수
return_value	SEQUENCE OF _db_meth_arg	함수의 리턴 값
arguments	SEQUENCE OF _db_meth_arg	함수의 입력인자

_db_meth_arg

메서드 인자에 대한 정보이며 meth_sig_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
meth_sig_of	_db_meth_sig	인자가 속한 함수 정보
data_type	INTEGER	인자의 데이터 타입(_db_attribute 의 “CUBRID가 지원하는 데이터 타입” 표의 “값” 중 하나)
index_of	INTEGER	함수정의에 인자가 나열된 순 서. 리턴 값이면 0, 입력인자이 면 1부터 시작함.
domains	SEQUENCE OF _db_domain	인자의 도메인

_db_meth_file

메서드에 대한 함수가 정의된 파일 정보이며 class_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
class_of	_db_class	메서드 파일 정보가 속한 클래스
from_class_of	_db_class	파일 정보가 상속된 것이면 그 파일 정보가 정의되어 있는 상위 클래스가 설정되며, 그렇지 않으면 NULL
path_name	VARCHAR(255)	메서드가 위치한 파일의 경로

_db_query_spec

가상 클래스의 SQL 정의문이며 class_of에 대한 인덱스가 생성되어 있다. 'spec'속성의 데이터 타입은 10.1 Patch 3 이전버전의 경우 VARCHAR(4096) 이다.

속성명	데이터 타입	설명	버전별특성(10.1버전만)
class_of	_db_class	가상 클래스에 대한 클래스 정보	
spec	VARCHAR(1073741823)	가상 클래스에 대한 SQL 정의문	10.1 Patch 4 이후
	VARCHAR(4096)		10.1 Patch 3 이전
invalidated_time	DATETIME	뷰 쿼리가 무효화된 시각	

_db_index

인덱스에 대한 정보이며 class_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
class_of	_db_class	인덱스가 속한 클래스
index_name	VARCHAR(255)	인덱스명
is_unique	INTEGER	고유 인덱스(unique index)이면 1, 그렇지 않으면 0
key_count	INTEGER	키를 구성하는 속성의 개수
key_attrs	SEQUENCE _db_index_key	OF 키를 구성하는 속성들
is_reverse	INTEGER	역 인덱스(reverse index)이면 1, 그렇지 않으면 0
is_primary_key	INTEGER	기본 키이면 1, 그렇지 않으면 0
is_foreign_key	INTEGER	외래 키이면 1, 그렇지 않으면 0
filter_expression	VARCHAR(1073741823)	필터링된 인덱스의 조건
have_function	INTEGER	함수 기반 인덱스이면 1, 그렇지 않으면 0
status	INTEGER	인덱스 상태
referential_index	_db_index	

속성명	데이터 타입	설명
		외래 키가 참조하는 인덱스 (외래 키가 아닌 경우 NULL)
delete_rule	INTEGER	외래 키 제약 조건의 삭제 규칙 (0: CASCADE, 1: RESTRICT, 2: NO ACTION, 3: SET NULL)
update_rule	INTEGER	외래 키 제약 조건의 업데이트 규칙 (1: RESTRICT, 2: NO ACTION, 3: SET NULL)
referential_match_option	INTEGER	외래키 제약 조건의 일치 옵션 (0: NONE) • NONE: NULL이 하나라도 존재 시 검사하지 않음
index_type	INTEGER	인덱스 타입 (0: BTREE)
options	INTEGER	인덱스 옵션 • Bit 0 ~ Bit 3: 인덱스의 deduplicate level 을 나타낸다. 자세한 내용은 DEDUPLICATE 을 참고한다. • 나머지 비트는 현재 사용하지 않는다.
comment	VARCHAR(1024)	인덱스 설명
created_time	DATETIME	인덱스 생성 시각
updated_time	DATETIME	인덱스 수정 시각

다음 예제에서는 클래스에 속하는 인덱스명을 검색한다.

```
SELECT class_of.class_name, index_name
FROM _db_index
ORDER BY 1;
```

```
class_of.class_name    index_name
=====
'_db_attribute'
'i__db_attribute_class_of_attr_name_attr_type'
'_db_auth'             'i__db_auth_grantee'
'_db_auth'             'i__db_auth_object_of'
'_db_class'            'i__db_class_unique_name'
'_db_class'            'i__db_class_class_name_owner'
'_db_class'            'i__db_class_class_of'
'_db_collation'        'pk__db_collation_coll_id'
'_db_data_type'        'pk__db_data_type_type_id_type_name'
'_db_domain'           'i__db_domain_object_of'
'_db_domain'           'i__db_domain_data_type'
'_db_ha_apply_info'    'u__db_ha_apply_info_db_name_copied_log_path'
'_db_index'            'i__db_index_class_of'
'_db_index'            'i__db_index_index_name'
'_db_index_key'        'i__db_index_key_index_of'
'_db_meth_arg'         'i__db_meth_arg_meth_sig_of'
'_db_meth_file'        'i__db_meth_file_class_of'
'_db_meth_sig'         'i__db_meth_sig_meth_of'
'_db_method'           'i__db_method_class_of_meth_name'
'_db_partition'        'i__db_partition_class_of_pname'
'_db_query_spec'       'i__db_query_spec_class_of'
'_db_serial'           'pk_db_serial_unique_name'
'_db_serial'           'u__db_serial_name_owner'
'_db_server'           'pk__db_server_link_name_owner'
'_db_stored_procedure' 'pk_db_stored_procedure_unique_name'
'_db_stored_procedure_args' 'i__db_stored_procedure_args_sp_of'
'_db_stored_procedure_code' 'pk__db_stored_procedure_code_name'
'_db_synonym'          'pk__db_synonym_unique_name'
'_db_synonym'          'i__db_synonym_name_owner_is_public'
'athlete'              'pk_athlete_code'
'code'                 'pk_code'
'_db_user'             'u__db_user_name'
'event'                'pk_event_code'
'game'                 'pk_game_host_year_event_code_athlete_code'
'game'                 'fk_game_event_code'
'game'                 'fk_game_athlete_code'
'history'              'pk_history_event_code_athlete'
'nation'               'pk_nation_code'
'olympic'              'pk_olympic_host_year'
'participant'          'pk_participant_host_year_nation_code'
'participant'          'fk_participant_host_year'
'participant'          'fk_participant_nation_code'
```

```
'record'
'pk_record_host_year_event_code_athlete_code_medal'
'stadium'          'pk_stadium_code'
```

_db_index_key

인덱스에 대한 키 정보이며 index_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
index_of	_db_index	키 속성이 속하는 인덱스
key_attr_name	VARCHAR(255)	키를 구성하는 속성명
key_order	INTEGER	키에서 속성이 위치한 순서로 0부터 시작함
asc_desc	INTEGER	속성 값의 순서가 내림차순이면 1, 그렇지 않으면 0
key_prefix_length	INTEGER	키로 사용할 prefix의 길이
func	VARCHAR(1023)	함수 기반 인덱스의 함수 표현식

다음 예제에서는 클래스에 속하는 인덱스명을 검색한다.

```
SELECT class_of.class_name, SEQUENCE(SELECT key_attr_name
                                     FROM _db_index_key k
                                     WHERE k in i.key_attrs)
FROM _db_index i
WHERE key_count >= 2;
```

```
class_of.class_name  sequence((select key_attr_name from
_db_index_key k where k in
i.key_attrs))
=====
```

```

'_db_server'      {'link_name', 'owner'}
'_db_synonym'     {'name', 'owner', 'is_public'}
'_db_ha_apply_info' {'db_name', 'copied_log_path'}
'_db_serial'      {'name', 'owner'}
'_db_data_type'   {'type_id', 'type_name'}
'_db_partition'   {'class_of', 'pname'}
'_db_method'      {'class_of', 'meth_name'}
'_db_attribute'   {'class_of', 'attr_name', 'attr_type'}
'_db_class'       {'class_name', 'owner'}
'participant'     {'host_year', 'nation_code'}
'game'            {'host_year', 'event_code', 'athlete_code'}
'record'          {'host_year', 'event_code', 'athlete_code',
'medal'}
'history'         {'event_code', 'athlete'}

```

_db_auth

객체에 대한 사용자 권한 정보를 표현하며, grantee에 대한 인덱스와 object_of에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
grantor	_db_user	권한 부여자
grantee	_db_user	권한 피부여자
object_type	INTEGER	권한부여 대상인 객체 타입 (0: 클래스, 5: 저장 프로시저)
object_of	object	권한부여 대상인 객체
auth_type	VARCHAR(7)	부여된 권한 타입 이름
is_grantable	INTEGER	권한 받은 객체에 대해 다른 사용자에게 권한을 부여할 수 있으면 1, 아니면 0
created_time	DATETIME	권한 부여 시각

속성명	데이터 타입	설명
updated_time	DATETIME	권한 변경 시각

CUBRID가 지원하는 권한 타입은 다음과 같다.

- **SELECT**
- **INSERT**
- **UPDATE**
- **DELETE**
- **ALTER**
- **INDEX**
- **EXECUTE**

다음 예제에서는 뷰 *db_trigger*에 정의되어 있는 권한 정보를 검색한다.

```
SELECT a.grantor.name, a.grantee.name, a.auth_type
FROM _db_auth a JOIN _db_class c ON a.object_of = c.class_of
WHERE c.class_name = 'db_trigger';
```

```
grantor.name      grantee.name      auth_type
=====
'DBA'             'PUBLIC'          'SELECT'
```

_db_data_type

CUBRID가 지원하는 데이터 타입(*_db_attribute*의 ‘CUBRID가 지원하는 데이터 타입’ 표 참조)을 나타낸다.

속성명	데이터 타입	설명
type_id	INTEGER	데이터 타입 식별자. ‘CUBRID가 지원하는 데이터 타입’ 표의 ‘값’에 해당함

속성명	데이터 타입	설명
type_name	VARCHAR(16)	데이터 타입 이름. 'CUBRID가 지원하는 데이터 타입' 표의 '의미'에 해당함

다음 예제에서는 클래스 *event*의 속성과 각 타입명을 검색한다.

```
SELECT a.attr_name, t.type_name
FROM _db_attribute a join _db_data_type t ON a.data_type = t.type_id
WHERE class_of.class_name = 'event'
ORDER BY a.def_order;
```

```
attr_name      type_name
=====
'code'         'INTEGER'
'sports'       'STRING'
'name'         'STRING'
'gender'       'CHAR'
'players'      'INTEGER'
```

_db_partition

분할에 대한 정보이며 class_of, pname에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
class_of	_db_class	Parent class의 OID
pname	VARCHAR(255)	Parent - NULL
ptype	INTEGER	• 0 - HASH • 1 - RANGE • 2 - LIST

속성명	데이터 타입	설명
pexpr	VARCHAR(2048)	Parent 에만 해당
pvalues	SEQUENCE OF	• Hash - 칼럼명, Hash size • RANGE - MIN/MAX value - 무한의 MIN/MAX는 NULL로 저장 • LIST - value list
class_partition_type	INTEGER	파티션 클래스의 파티션 타입 (1: PARTITIONED CLASS, 2: PARTITION CLASS) • PARTITIONED CLASS: 파티션을 가지는 부모 테이블 • PARTITION CLASS: 실제 데이터를 저장하는 개별 파티션 테이블
comment	VARCHAR(1024)	분할 설명

_db_stored_procedure

저장 프로시저에 대한 정보이며 unique_name에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
unique_name	VARCHAR(255)	스키마 이름이 접두어로 붙은 저장 함수 이름
sp_name	VARCHAR(255)	저장 프로시저 이름

속성명	데이터 타입	설명
sp_type	INTEGER	저장 프로시저 종류 (프로시저 또는 함수)
return_type	INTEGER	리턴 값 타입
arg_count	INTEGER	매개변수 개수
args	SEQUENCE OF _db_stored_procedure_args	매개변수 리스트
lang	INTEGER	구현 언어
pkg_name	VARCHAR(255)	저장 프로시저가 속한 패키지 이름
is_system_generated	INTEGER	시스템이 생성한 저장 프로시저 여부
directive	INTEGER	저장 프로시저의 실행 동작 속성
sql_data_access	INTEGER	SQL 데이터 접근 속성 (0: NO SQL, 1: CONTAINS SQL, 2: READS SQL DATA, 3: MODIFIES SQL DATA)
target_class	VARCHAR(1024)	실행할 저장 프로시저의 클래스 이름
target_method	VARCHAR(4096)	실행할 저장 프로시저의 메서드 이름

속성명	데이터 타입	설명
owner	_db_user	저장 프로시저의 소유자
comment	VARCHAR(1024)	저장 프로시저 설명
created_time	DATETIME	저장 프로시저 생성 시각
updated_time	DATETIME	저장 프로시저 수정 시각

참고

- **unique_name**: 예) **dba.sp_test**
- **sp_name**: 예) **sp_test**
- **sp_type**: 값이 **1** 이면 저장 프로시저, **2** 이면 저장 함수이다.
- **return_type**: 반환 타입은 아래 값 중 하나이다

```
NULL = 0, INTEGER = 1, FLOAT = 2, DOUBLE = 3,
STRING = 4,
SET = 6, MULTiset = 7, SEQUENCE = 8, TIME = 10,
TIMESTAMP = 11, DATE = 12, SHORT = 18, NUMERIC
= 22, CHAR = 25,
CURSOR = 28, BIGINT = 31, DATETIME = 32
```

- Java SP는 위 명시된 데이터 타입을 모두 지원한다
- PL/CSQL은 위 명시된 데이터 타입 중 SET, MULTiset, SEQUENCE, CURSOR를 제외한 타입을 지원한다.
- 나머지 명시되지 않은 데이터 타입은 지원하지 않는다.
- **lang**: 값이 **0** 이면 PL/CSQL, **1** 이면 Java이다.
- **pkg_name**: 패키지 이름. 현재 패키지 이름은 **DBMS_OUTPUT** 시스템 패키지에서 만 사용한다.

- **is_system_generated**: 값이 **1** 이면 시스템이 생성한 저장 프로시저이며, **0** 이면 사용자가 생성한 저장 프로시저이다.
- **directive**: 값의 제일 오른쪽 값부터 **Bit 0** 부터 **Bit 31** 까지 사용한다.
 - **Bit 0**: 저장 프로시저의 **execution rights (AUTHID)** 를 나타낸다.
 - 값이 **0** 이면 소유자 권한 (또는 정의자 권한), **1** 이면 호출자 권한이다.
 - **Bit 1**: 저장 프로시저의 **deterministic** 속성을 나타낸다.
 - 값이 **1** 이면 deterministic, **0** 이면 non-deterministic이다.
 - 나머지 비트 (**Bit 2 ~ Bit 31**) 은 현재 사용하지 않는다.

_db_stored_procedure_args

저장 함수 인자에 대한 정보이며 sp_name 칼럼에 대한 인덱스가 생성되어 있다.

속성명	데이터 타입	설명
sp_of	_db_stored_procedure	저장 프로시저의 객체
index_of	INTEGER	매개변수의 순서
is_system_generated	INTEGER	시스템이 생성한 저장 프로시저 인수 여부
arg_name	CHARACTER VARYING(255)	매개변수의 이름
data_type	INTEGER	매개변수의 데이터 타입
mode	INTEGER	매개변수 모드 (IN, OUT, INOUT)

속성명	데이터 타입	설명
default_value	CHARACTER VARYING(255)	매개변수의 기본값
is_optional	INTEGER	선택적 매개변수 여부
comment	CHARACTER VARYING(1024)	저장 프로시저 인자 설명

참고

- **index_of**: 매개변수의 순서는 0부터 시작한다.
- **is_system_generated**: 값이 **1** 이면 시스템이 생성한 저장 프로시저 인수이며, **0** 이면 사용자가 생성한 저장 프로시저 인수이다.
- **data_type**: 매개변수의 데이터 타입은 아래 값 중 하나이다

```
NULL = 0, INTEGER = 1, FLOAT = 2, DOUBLE = 3,
STRING = 4,
SET = 6, MULTiset = 7, SEQUENCE = 8, TIME = 10,
TIMESTAMP = 11, DATE = 12, SHORT = 18, NUMERIC
= 22, CHAR = 25,
CURSOR = 28, BIGINT = 31, DATETIME = 32
```

- Java SP는 위 명시된 데이터 타입을 모두 지원한다
 - PL/CSQL은 위 명시된 데이터 타입 중 SET, MULTiset, SEQUENCE, CURSOR를 제외한 타입을 지원한다.
 - 나머지 명시되지 않은 데이터 타입은 지원하지 않는다.
 - **mode**: 매개변수 모드는 아래 값 중 하나이다
- ```
IN = 0, OUT = 1, INOUT = 2
```
- **default\_value**: 매개변수의 기본값이다. 자세한 내용은 **기본값 인자 지정** 을 참고한다.

· **is\_optional**: 매개변수가 선택 가능하면 **1**, 필수이면 **0** 이다.

\_db\_stored\_procedure\_code

저장 함수 코드에 대한 정보이며 name 칼럼에 대한 인덱스가 생성되어 있다.

| 속성명                 | 데이터 타입                        | 설명                     |
|---------------------|-------------------------------|------------------------|
| name                | CHARACTER VARYING(1024)       | 저장 프로시저 코드의 이름         |
| created_time        | CHARACTER VARYING(16)         | 저장 프로시저 코드의 생성 시각      |
| owner               | _db_user                      | 저장 프로시저 코드의 소유자        |
| is_static           | INTEGER                       | 저장 프로시저 코드의 정적 로딩 여부   |
| is_system_generated | INTEGER                       | 시스템이 생성한 저장 프로시저 코드 여부 |
| stype               | INTEGER                       | 저장 프로시저의 소스 코드 유형      |
| scode               | CHARACTER VARYING(1073741823) | 저장 프로시저의 소스 코드         |
| otype               | INTEGER                       | 저장 프로시저의 목적 코드 유형      |
| ocode               | CHARACTER VARYING(1073741823) | 저장 프로시저의 목적 코드         |

## 참고

### · is\_static

- 값이 **1** 이면 저장 프로시저 코드를 한번만 로드하며, 프로시저 코드의 변경 시에도 적용되지 않고 서버 재시작 시에만 적용된다.
- 값이 **0** 이면 저장 프로시저 코드를 매번 로드하며, DDL 또는 프로시저 로드 유틸리티 사용으로 프로시저 코드 변경 시 즉시 적용된다.
- Java Native Interface (JNI) 지원 에서 사용된다.

· **is\_system\_generated**: 값이 **1** 이면 시스템이 생성한 코드이며, **0** 이면 사용자가 작성한 코드이다.

· **stype**: 소스 코드의 유형을 나타낸다. 현재 PL/CSQL 코드만 지원하며 값은 **0** 이다.

· **scode**: 사용자가 입력한 저장 프로시저 코드의 소스 코드가 저장되어 있다. 현재 PL/CSQL 코드만 저장된다.

· **otype**: 저장 프로시저 소스 코드로부터 컴파일한 목적 코드의 유형을 나타낸다. 현재 PL/CSQL만 지원한다.

- 값이 **0** 이면 자바 클래스 파일을 나타낸다
- 값이 **1** 이면 자바 아카이브 파일을 나타낸다
- 그 외의 값은 현재 지원하지 않는다.

· **ocode**: 저장 프로시저 소스 코드로부터 컴파일하여 PL 실행 서버에서 실행할 수 있는 목적 코드가 저장되어 있다. 현재 PL/CSQL을 위한 목적 코드만 저장한다.

\_db\_server

| Attribute Name | Data Type    | Description |
|----------------|--------------|-------------|
| link_name      | VARCHAR(255) | 연결 이름       |

| Attribute Name   | Data Type           | Description         |
|------------------|---------------------|---------------------|
| host             | VARCHAR(255)        | 서버의 호스트 이름          |
| port             | INTEGER             | 서버의 연결 포트           |
| db_name          | VARCHAR(255)        | 서버의 데이터베이스 이름       |
| user_name        | VARCHAR(255)        | 서버의 데이터베이스 사용자 이름   |
| password         | VARCHAR(1073741823) | 서버의 데이터베이스 사용자 비밀번호 |
| properties       | VARCHAR(2048)       | 연결 시 사용되는 프로퍼티 정보   |
| owner            | _db_user            | 연결 정보의 소유자          |
| comment          | VARCHAR(1024)       | 서버 설명               |
| created_time     | DATETIME            | 서버 생성 시각            |
| updated_time     | DATETIME            | 서버 수정 시각            |
| invalidated_time | DATETIME            | 외부 서버 연결이 무효화된 시각   |

\_db\_synonym

동의어에 대한 대상 객체 정보이며 unique\_name에 대한 인덱스와 name, owner 및 is\_public에 대한 인덱스가 생성되어 있다.

| Attribute Name     | Data Type     | Description                               |
|--------------------|---------------|-------------------------------------------|
| unique_name        | VARCHAR(255)  | 스키마 이름이 접두어로 붙은 동의어 이름                    |
| name               | VARCHAR(255)  | 동의어 이름                                    |
| owner              | _db_user      | 동의어의 소유자                                  |
| is_public          | INTEGER       | 공용(Public) 동의어이면 1이고, 전용(Private) 동의어이면 0 |
| target_unique_name | VARCHAR(255)  | 스키마 이름이 접두어로 붙은 대상 객체 이름                  |
| target_name        | VARCHAR(255)  | 대상 객체 이름                                  |
| target_owner       | _db_user      | 대상 객체 소유자                                 |
| comment            | VARCHAR(2048) | 동의어 설명                                    |
| created_time       | DATETIME      | 동의어 생성 시각                                 |
| updated_time       | DATETIME      | 동의어 수정 시각                                 |

### ⚠ 경고

아직은 공용(Public) 동의어를 지원하지 않는다.

## \_db\_user

사용자에 대한 정보를 표현하며 name에 대한 인덱스가 생성되어 있다.

| 속성명               | 데이터 타입             | 설명                                                       |
|-------------------|--------------------|----------------------------------------------------------|
| name              | VARCHAR(32)        | 사용자명                                                     |
| id                | INTEGER            | 사용자 식별자                                                  |
| password          | _db_password       | 사용자 패스워드. 사용자에게는 표시되지 않는다.                               |
| direct_groups     | SET OF _db_user    | 사용자가 직접적으로 속한 그룹                                         |
| groups            | SET OF _db_user    | 사용자가 직·간접적으로 속한 그룹                                       |
| authorization     | _db_authorization  | 사용자가 소유한 권한 정보                                           |
| triggers          | SEQUENCE OF object | 사용자의 action에 의해 발생하는 트리거들                                |
| is_loginable      | INTEGER            | 로그인 허용이면 1, 비허용이면 0                                      |
| is_system_created | INTEGER            | 시스템이 생성한 사용자이면 1 (예: <b>DBA</b> , <b>PUBLIC</b> ), 아니면 0 |
| comment           | VARCHAR(1024)      | 사용자 설명                                                   |
| created_time      | DATETIME           | 사용자 생성 시각                                                |



| 속성명          | 데이터 타입   | 설명        |
|--------------|----------|-----------|
| updated_time | DATETIME | 사용자 수정 시각 |

### 메서드 이름

- **set\_password ()**
- **set\_password\_encoded ()**
- **set\_password\_encoded\_sha1 ()**
- **add\_member ()**
- **drop\_member ()**
- **print\_authorizations ()**
- **add\_user ()**
- **drop\_user ()**
- **find\_user ()**
- **login ()**

### \_db\_authorization

사용자별 권한 요약을 표현한다.

| 속성명    | 데이터 타입   | 설명                                        |
|--------|----------|-------------------------------------------|
| owner  | _db_user | 이 권한 객체의 소유자                              |
| grants | SEQUENCE | {권한이 부여된 객체, 객체의 권한 부여자, 권한 종류}의 sequence |

## \_db\_serial

시리얼에 대한 정보를 표현하며 unique\_name에 대한 인덱스와 name, owner에 대한 인덱스가 생성되어 있다.

[illegible]

| 속성명          | 데이터 타입              | 설명                                             |
|--------------|---------------------|------------------------------------------------|
|              |                     | AUTO_INCREMENT 시리얼이면 테이블명을 저장. 아니면 <b>NULL</b> |
| attr_name    | VARCHAR(1073741823) | AUTO_INCREMENT 시리얼이면 컬럼명을 저장. 아니면 <b>NULL</b>  |
| cached_num   | INTEGER             | 성능 향상을 위해 메모리에 미리 생성할 시리얼 값의 개수. 기본값은 0        |
| comment      | VARCHAR(1024)       | 시리얼 설명                                         |
| created_time | DATETIME            | 시리얼 생성 시각                                      |
| updated_time | DATETIME            | 시리얼 수정 시각                                      |

### 메서드 이름

· **change\_serial\_owner()**

### \_db\_trigger

트리거에 대한 정보를 표현한다.

| 속성명         | 데이터 타입              | 설명              |
|-------------|---------------------|-----------------|
| unique_name | VARCHAR(1073741823) | 스키마명을 포함하는 트리거명 |
| owner       | _db_user            | 트리거 소유자         |

| 속성명                    | 데이터 타입              | 설명                                                                                                                         |
|------------------------|---------------------|----------------------------------------------------------------------------------------------------------------------------|
| name                   | VARCHAR(1073741823) | 트리거명                                                                                                                       |
| status                 | INTEGER             | INACTIVE이면 1, ACTIVE이면 2. 기본값은 2                                                                                           |
| priority               | DOUBLE              | 트리거 간의 수행 순서에 대한 우선순위. 기본값은 0                                                                                              |
| event                  | INTEGER             | UPDATE는 0, UPDATE STATEMENT는 1, DELETE는 2, DELETE STATEMENT는 3, INSERT는 4, INSERT STATEMENT는 5, COMMIT는 8, ROLLBACK은 9로 설정 |
| target_class           | object              | 트리거 대상(target)인 클래스에 대한 클래스 객체                                                                                             |
| target_attribute       | VARCHAR(1073741823) | 트리거 대상 속성명. 대상 속성이 명시되지 않으면 <b>NULL</b> 을 설정                                                                               |
| target_class_attribute | INTEGER             | 대상 속성에 대해, 인스턴스 속성이면 0, 클래스 속성이면 1. 기본값은 0                                                                                 |
| condition_type         | INTEGER             | 조건이 있으면 1, 조건이 없으면 <b>NULL</b>                                                                                             |
| condition              | VARCHAR(1073741823) | IF문에 명시된 action 발생 조건                                                                                                      |
| condition_time         | INTEGER             |                                                                                                                            |

| 속성명               | 데이터 타입              | 설명                                                                                       |
|-------------------|---------------------|------------------------------------------------------------------------------------------|
|                   |                     | 조건이 있으면 BEFORE는 1, AFTER는 2, DEFERRED는 3으로 설정. 조건이 없으면 <b>NULL</b>                       |
| action_type       | INTEGER             | INSERT, UPDATE, DELETE, CALL 중 하나이면 1, REJECT이면 2, INVALIDATE_TRANSACTION이면 3, PRINT이면 4 |
| action_definition | VARCHAR(1073741823) | triggering되는 수행문                                                                         |
| action_time       | INTEGER             | BEFORE는 1, AFTER는 2, DEFERRED는 3으로 설정                                                    |
| comment           | VARCHAR(1024)       | 트리거 설명                                                                                   |
| created_time      | DATETIME            | 트리거 생성 시각                                                                                |
| updated_time      | DATETIME            | 트리거 수정 시각                                                                                |

### \_db\_ha\_apply\_info

**applylogdb** 유틸리티가 복제 로그를 반영할 때마다 그 진행 상태를 저장하기 위한 테이블이다. 이 테이블은 **applylogdb** 유틸리티가 커밋하는 시점마다 갱신되며, *\_counter* 칼럼에는 수행 연산의 누적 카운트 값이 저장된다. 각 칼럼의 의미는 다음과 같다.

| 칼럼명              | 칼럼 타입        | 의미            |
|------------------|--------------|---------------|
| db_name          | VARCHAR(255) | 로그에 저장된 DB 이름 |
| db_creation_time | DATETIME     |               |

| 칼럼명                  | 칼럼 타입         | 의미                                                                                                             |
|----------------------|---------------|----------------------------------------------------------------------------------------------------------------|
|                      |               | 반영하는 로그에 대한 원본 DB의 생성 시각                                                                                       |
| copied_log_path      | VARCHAR(4096) | 반영하는 로그 파일의 경로                                                                                                 |
| committed_lsa_pageid | BIGINT        | 마지막에 반영한 commit log lsa의 page id, applylogdb가 재시작해도 last_committed_lsa 보다 앞선 로그는 재반영하지 않음                      |
| committed_lsa_offset | INTEGER       | 마지막에 반영한 commit log lsa의 offset, applylogdb가 재시작해도 last_committed_lsa 보다 앞선 로그는 재반영하지 않음                       |
| committed_rep_pageid | BIGINT        | 마지막 반영한 복제 로그 lsa의 pageid, 복제 반영 지연 여부 확인                                                                      |
| committed_rep_offset | INTEGER       | 마지막 반영한 복제 로그 lsa의 offset, 복제 반영 지연 여부 확인                                                                      |
| append_lsa_page_id   | BIGINT        | 마지막 반영 당시 복제 로그 마지막 lsa의 page id, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 append_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인 |
| append_lsa_offset    | INTEGER       | 마지막 반영 당시 복제 로그 마지막 lsa의 offset, 복제 반영 당시, applylogdb에서 처                                                      |

| 칼럼명                  | 칼럼 타입   | 의미                                                                                                          |
|----------------------|---------|-------------------------------------------------------------------------------------------------------------|
|                      |         | 리 중인 복제 로그 헤더의 append_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인                                                        |
| eof_lsa_page_id      | BIGINT  | 마지막 반영 당시 복제 로그 EOF lsa의 page id, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 eof_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인 |
| eof_lsa_offset       | INTEGER | 마지막 반영 당시 복제 로그 EOF lsa의 offset, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 eof_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인  |
| final_lsa_pageid     | BIGINT  | applylogdb에서 마지막으로 처리한 로그 lsa의 pageid, 복제 반영 지연 여부 확인                                                       |
| final_lsa_offset     | INTEGER | applylogdb에서 마지막으로 처리한 로그 lsa의 offset, 복제 반영 지연 여부 확인                                                       |
| required_page_id     | BIGINT  | log_max_archives 파라미터에 의해 삭제되지 않아야 할 가장 작은 log page id, 복제 반영 시작할 로그 페이지 번호                                 |
| required_page_offset | INTEGER | 복제 반영 시작할 로그 페이지 offset                                                                                     |

| 칼럼명              | 칼럼 타입    | 의미                                                        |
|------------------|----------|-----------------------------------------------------------|
| log_record_time  | DATETIME | 슬레이브 DB에 커밋된 복제 로그에 포함된 timestamp, 즉 해당 로그 레코드 생성 시각      |
| log_commit_time  | DATETIME | 마지막 commit log의 반영 시각                                     |
| last_access_time | DATETIME | _db_ha_apply_info 카탈로그의 최종 갱신 시각                          |
| status           | INTEGER  | 반영 진행 상태(0: IDLE, 1: BUSY)                                |
| insert_counter   | BIGINT   | applylogdb가 insert한 횟수                                    |
| update_counter   | BIGINT   | applylogdb가 update한 횟수                                    |
| delete_counter   | BIGINT   | applylogdb가 delete한 횟수                                    |
| schema_counter   | BIGINT   | applylogdb가 schema를 변경한 횟수                                |
| commit_counter   | BIGINT   | applylogdb가 commit한 횟수                                    |
| fail_counter     | BIGINT   | applylogdb가 insert/update/delete/commit/schema 변경 중 실패 횟수 |
| start_time       | DATETIME | applylogdb 프로세스가 슬레이브 DB에 접속한 시각                          |



## dual

dual 테이블은 오직 하나의 열과 행을 가지며 더미 테이블로 사용된다. dual 테이블은 상수, 계산식 또는 SYS\_DATE나 USER와 같은 의사 칼럼을 조회할 때 사용된다. 큐브리드에서 의사 칼럼은 함수로써 제공되며 이에 대한 자세한 내용과 예제들은 [연산자와 함수](#)를 참고한다. 상수, 계산식 또는 의사 칼럼을 조회할 때는 FROM절을 생략하여도 dual 테이블이 자동적으로 참조된다. dual 테이블은 다른 시스템 카탈로그처럼 dba 소유로 생성되지만 dba는 dual 테이블에 대해 SELECT 연산만 수행가능하다. 하지만 다른 시스템 카탈로그와 다르게 PUBLIC 사용자들 또한 dual 테이블에 대해 SELECT 연산을 수행할 수 있다.

| 속성명   | 데이터 타입     | 설명              |
|-------|------------|-----------------|
| dummy | VARCHAR(1) | 더미 목적으로만 사용되는 값 |

다음은 CSQL에서 “;plan detail” 명령 입력 또는 “SET OPTIMIZATION LEVEL 513;”을 입력 후 의사 칼럼을 조회하는 질의를 수행한 결과이다([질의 실행 계획 보기](#)). FROM절을 생략하여도 자동적으로 dual 테이블이 참조되는 것을 볼 수 있다.

```
SET OPTIMIZATION LEVEL 513;
SELECT SYS_DATE;
```

```
Join graph segments (f indicates final):
seg[0]: [0]
Join graph nodes:
node[0]: dual dual(1/1) (loc -1)
```

Query plan:

```
sscan
 class: dual node[0]
 cost: 1 card 1
```

Query stmt:

```
select SYS_DATE from dual dual
```

```
=== <Result of SELECT Command in Line 1> ===
```

```
 SYS_DATE
=====
 11/26/2020
```

## 시스템 카탈로그 가상 클래스

일반 사용자는 자신이 권한을 가진 클래스에 대해서만 그 클래스와 관련된 정보들을 시스템 카탈로그 가상 클래스들을 통해 볼 수 있다. 이 절에서는 각 시스템 카탈로그 가상 클래스들이 어떤 정보를 표현하는지와 가상 클래스 정의문에 대해 설명한다.

| 가상 클래스 이름                 | 설명                                                 |
|---------------------------|----------------------------------------------------|
| DB_CLASS                  | 클래스에 대한 정보                                         |
| DB_DIRECT_SUPER_CLASS     | 상위 클래스에 대한 정보                                      |
| DB_VCLASS                 | 가상 클래스의 SQL 정의문                                    |
| DB_ATTRIBUTE              | 클래스의 속성에 대한 정보                                     |
| DB_ATTR_SETDOMAIN_ELM     | 컬렉션 타입(SET, MULTISSET, SEQUENCE) 속성의 원소에 대한 데이터 타입 |
| DB_CHARSET                | 문자셋에 대한 정보                                         |
| DB_COLLATION              | 콜레이션에 대한 정보                                        |
| DB_METHOD                 | 메서드에 대한 정보                                         |
| DB_METH_ARG               | 메서드의 인자에 대한 정보                                     |
| DB_METH_ARG_SETDOMAIN_ELM | 컬렉션 타입(SET, MULTISSET, SEQUENCE) 인자의 원소에 대한 데이터 타입 |
| DB_METH_FILE              | 메소드가 정의된 파일 정보                                     |
| DB_INDEX                  | 인덱스에 대한 정보                                         |

| 가상 클래스 이름                | 설명                                        |
|--------------------------|-------------------------------------------|
| DB_INDEX_KEY             | 인덱스에 대한 키 정보                              |
| DB_AUTH                  | 클래스에 대한 사용자 권한 정보                         |
| DB_SERIAL                | 시리얼에 대한 정보                                |
| DB_TRIGGER               | 트리거에 대한 정보                                |
| DB_HA_APPLY_INFO         | <b>applylogdb</b> 유틸리티가 복제 로그를 적용하는 진행 상태 |
| DB_PARTITION             | 분할에 대한 정보                                 |
| DB_STORED_PROCEDURE      | Java 저장 함수에 대한 정보                         |
| DB_STORED_PROCEDURE_ARGS | Java 저장 함수 인자에 대한 정보                      |
| DB_SERVER                | DBLink에 대한 서버 정보                          |
| DB_SYNONYM               | 동의어에 대한 대상 객체 정보                          |

## DB\_CLASS

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대한 정보를 보여준다.

| 속성명        | 데이터 타입       | 설명   |
|------------|--------------|------|
| class_name | VARCHAR(255) | 클래스명 |

| 속성명                 | 데이터 타입        | 설명                                   |
|---------------------|---------------|--------------------------------------|
| owner_name          | VARCHAR(32)   | 클래스 소유자명                             |
| class_type          | VARCHAR(6)    | 클래스이면 'CLASS', 가상 클래스이면 'VCLASS'     |
| is_system_class     | VARCHAR(3)    | 시스템 클래스이면 'YES', 아니면 'NO'            |
| tde_algorithm       | VARCHAR(32)   | TDE 암호화 알고리즘                         |
| statistics_strategy | VARCHAR(8)    | 통계 정보 수집 전략 ('SAMPLING', 'FULLSCAN') |
| partitioned         | VARCHAR(3)    | 분할 그룹 클래스이면 'YES', 아니면 'NO'          |
| is_reuse_oid_class  | VARCHAR(3)    | REUSE_OID 클래스이면 'YES', 아니면 'NO'      |
| collation           | VARCHAR(32)   | 콜레이션 이름                              |
| comment             | VARCHAR(2048) | 클래스 설명                               |
| created_time        | DATETIME      | 클래스 생성 시각                            |
| updated_time        | DATETIME      | 클래스 수정 시각                            |
| checked_time        | DATETIME      | 클래스의 통계 정보 갱신 시각                     |

다음 예제에서는 현재 사용자가 소유하고 있는 클래스를 검색한다.

```

/* CURRENT_USER: PUBLIC */
SELECT class_name, owner_name
FROM db_class
WHERE owner_name = CURRENT_USER;

```

| class_name    | owner_name |
|---------------|------------|
| =====         | =====      |
| 'stadium'     | 'PUBLIC'   |
| 'code'        | 'PUBLIC'   |
| 'nation'      | 'PUBLIC'   |
| 'event'       | 'PUBLIC'   |
| 'athlete'     | 'PUBLIC'   |
| 'participant' | 'PUBLIC'   |
| 'olympic'     | 'PUBLIC'   |
| 'game'        | 'PUBLIC'   |
| 'record'      | 'PUBLIC'   |
| 'history'     | 'PUBLIC'   |

다음 예제에서는 현재 사용자가 접근할 수 있는 가상 클래스를 검색한다.

```

SELECT class_name
FROM db_class
WHERE class_type = 'VCLASS';

```

| class_name                  |
|-----------------------------|
| =====                       |
| 'db_synonym'                |
| 'db_server'                 |
| 'db_charset'                |
| 'db_collation'              |
| 'db_stored_procedure_args'  |
| 'db_stored_procedure'       |
| 'db_partition'              |
| 'db_trigger'                |
| 'db_auth'                   |
| 'db_index_key'              |
| 'db_index'                  |
| 'db_meth_file'              |
| 'db_meth_arg_setdomain_elm' |
| 'db_meth_arg'               |
| 'db_method'                 |
| 'db_attr_setdomain_elm'     |
| 'db_attribute'              |
| 'db_vclass'                 |
| 'db_direct_super_class'     |
| 'db_class'                  |

다음 예제에서는 현재 사용자가 접근할 수 있는 시스템 클래스를 검색한다.

```
SELECT class_name
FROM db_class
WHERE is_system_class = 'YES' AND class_type = 'CLASS'
ORDER BY class_name;
```

```
class_name
=====
'db_root'
'dual'
```

#### DB\_DIRECT\_SUPER\_CLASS

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 상위 클래스가 존재하면 그 클래스명을 보여준다.

| 속성명              | 데이터 타입       | 설명               |
|------------------|--------------|------------------|
| class_name       | VARCHAR(255) | 클래스명             |
| owner_name       | VARCHAR(32)  | 클래스 소유자명         |
| super_class_name | VARCHAR(255) | 한 단계 상위 클래스명     |
| super_owner_name | VARCHAR(32)  | 한 단계 상위 클래스 소유자명 |

다음 예제에서는 클래스 *female\_event* 의 상위 클래스를 검색한다. (ADD SUPERCLASS 절 참조)

```
SELECT super_class_name
FROM db_direct_super_class
WHERE class_name = 'female_event';
```

```
super_class_name
=====
'event'
```

다음 예제에서는 현재 사용자가 소유하고 있는 클래스의 상위 클래스를 검색한다. ( [ADD SUPERCLASS 절 참조](#))

```
/* CURRENT_USER: PUBLIC */
SELECT class_name, super_class_name
FROM db_direct_super_class
WHERE owner_name = CURRENT_USER
ORDER BY class_name;
```

```
class_name super_class_name
=====
'female_event' 'event'
```

## DB\_VCLASS

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 가상 클래스들에 대해 그 SQL 정의문을 보여 준다.

‘vclass\_def’ 속성의 데이터 타입은 10.1 Patch 3 이전버전의 경우 VARCHAR(4096) 이다.

| 속성명          | 데이터 타입              | 설명              | 버전별특성(10.1버전만)  |
|--------------|---------------------|-----------------|-----------------|
| vclass_name  | VARCHAR(255)        | 가상 클래스명         |                 |
| owner_name   | VARCHAR(32)         | 가상 클래스 소유자명     |                 |
| vclass_def   | VARCHAR(1073741823) | 가상 클래스의 SQL 정의문 | 10.1 Patch 4 이후 |
|              | VARCHAR(4096)       |                 | 10.1 Patch 3 이전 |
| comment      | VARCHAR(2048)       | 가상 클래스 설명       |                 |
| created_time | DATETIME            | 가상 클래스 생성 시각    |                 |

| 속성명              | 데이터 타입   | 설명            | 버전별특성(10.1버전만) |
|------------------|----------|---------------|----------------|
| updated_time     | DATETIME | 가상 클래스 수정 시각  |                |
| invalidated_time | DATETIME | 뷰 쿼리가 무효화된 시각 |                |

다음 예제에서는 가상 클래스 `db_class` 의 SQL 정의문을 검색한다.

```
SELECT vclass_def
FROM db_vclass
WHERE vclass_name = 'db_class';
```

```
vclass_def
=====
'SELECT [c].[class_name] AS [class_name], [c].[owner].[name] AS
[owner_name], CASE [c].[class_type] WHEN 0 THEN 'CLASS' WHEN 1 THEN
'VCLASS' ELSE 'UNKNOWN' END AS [class_type], CASE WHEN [c].
[is_system_class] = 1 THEN 'YES' ELSE 'NO' END AS [is_system_class],
CASE [c].[tde_algorithm] WHEN 0 THEN 'NONE' WHEN 1 THEN 'AES' WHEN 2
THEN 'ARIA' END AS [tde_algorithm], CASE [c].[statistics_strategy]
WHEN 0 THEN 'SAMPLING' WHEN 1 THEN 'FULLSCAN' ELSE NULL END AS
[statistics_strategy], CASE WHEN [c].[sub_classes] IS NULL THEN 'NO'
ELSE NVL ((SELECT 'YES' FROM [_db_partition] AS [p] WHERE [p].
[class_of] = [c] AND [p].[pname] IS NULL), 'NO') END AS
[partitioned], CASE WHEN ([c].[flags] & 8) <> 0 THEN 'YES' ELSE 'NO'
END AS [is_reuse_oid_class], [coll].[coll_name] AS [collation], [c].
[comment] AS [comment], [c].[created_time] AS [created_time], [c].
[updated_time] AS [updated_time], [c].[checked_time] AS
[checked_time] FROM [_db_class] AS [c], [_db_collation] AS [coll]
WHERE [c].[collation_id] = [coll].[coll_id] AND ({'DBA'} SUBSETEQ
(SELECT SET {CURRENT_USER} + COALESCE (SUM (SET {[t].[g].[name]}),
SET {})) FROM [_db_user] AS [u], TABLE ([u].[groups]) AS [t] ([g])
WHERE [u].[name] = CURRENT_USER) OR {[c].[owner].[name]} SUBSETEQ
(SELECT SET {CURRENT_USER} + COALESCE (SUM (SET {[t].[g].[name]}),
SET {})) FROM [_db_user] AS [u], TABLE ([u].[groups]) AS [t] ([g])
WHERE [u].[name] = CURRENT_USER) OR {[c].[class_of]} SUBSETEQ
(SELECT SUM (SET {[au].[object_of]}) FROM [_db_auth] AS [au] WHERE
{[au].[grantee].[name]} SUBSETEQ (SELECT SET {CURRENT_USER} +
COALESCE (SUM (SET {[t].[g].[name]}), SET {})) FROM [_db_user] AS
[u], TABLE ([u].[groups]) AS [t] ([g]) WHERE [u].[name] =
CURRENT_USER) AND [au].[auth_type] = 'SELECT'))'
```



## DB\_ATTRIBUTE

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 그 속성 정보를 보여준다.

| 속성명             | 데이터 타입       | 설명                                                                       |
|-----------------|--------------|--------------------------------------------------------------------------|
| attr_name       | VARCHAR(255) | 속성명                                                                      |
| class_name      | VARCHAR(255) | 속성이 속한 클래스명                                                              |
| owner_name      | VARCHAR(32)  | 속성이 속한 클래스 소유자명                                                          |
| attr_type       | VARCHAR(8)   | 인스턴스 속성이면<br>'INSTANCE', 클래스 속성이면<br>'CLASS', 공유 속성이면<br>'SHARED'        |
| def_order       | INTEGER      | 클래스에서 속성이 정의된 순<br>서로 0부터 시작함. 상속받은<br>속성이면 그 상위 클래스에서<br>정의된 순서임.       |
| from_class_name | VARCHAR(255) | 상속받은 속성이면 그 속성이<br>정의되어 있는 상위 클래스명<br>이 설정되며, 그렇지 않으면<br><b>NULL</b>     |
| from_owner_name | VARCHAR(32)  | 상속받은 속성이면 그 속성이<br>정의되어 있는 상위 클래스의<br>소유자명이 설정되며, 그렇지<br>않으면 <b>NULL</b> |
| from_attr_name  | VARCHAR(255) | 상속받은 속성이며, 이름 충돌<br>이 발생하여 이를 해결하기 위<br>해 그 속성명이 바뀐 경우, 상                |

| 속성명               | 데이터 타입       | 설명                                                                                                                                     |
|-------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------|
|                   |              | 위 클래스에 정의된 원래 이름임. 그 이외에는 모두 <b>NULL</b>                                                                                               |
| data_type         | VARCHAR(9)   | 속성의 데이터 타입(<br><b>_db_attribute</b> 의 'CUBRID가<br>지원하는 데이터 타입' 표의<br>'의미' 중 하나)                                                        |
| prec              | INTEGER      | 데이터 타입의 전체 자릿수.<br>전체 자릿수가 명시되지 않은<br>경우 0임                                                                                            |
| scale             | INTEGER      | 데이터 타입의 소수점 이하의<br>자릿수. 소수점 이하의 자릿수<br>가 명시되지 않은 경우 0임                                                                                 |
| charset           | VARCHAR(32)  | 문자셋 이름                                                                                                                                 |
| collation         | VARCHAR(32)  | 콜레이션 이름                                                                                                                                |
| domain_class_name | VARCHAR(255) | 데이터 타입이 객체 타입인 경<br>우 그 도메인 클래스명. 객체<br>타입이 아닌 경우 <b>NULL</b>                                                                          |
| domain_owner_name | VARCHAR(32)  | 데이터 타입이 객체 타입인 경<br>우 그 도메인 클래스 소유자<br>명. 객체 타입이 아닌 경우<br><b>NULL</b>                                                                  |
| default_value     | VARCHAR(255) | 기본값으로서 그 데이터 타입<br>에 관계없이 모두 문자열로 저<br>장. 기본값이 없으면 <b>NULL</b> ,<br>기본값이 <b>NULL</b> 이면<br>'NULL'로 표현됨. 데이터 타입<br>이 객체 타입이면 'volume id |

| 속성명         | 데이터 타입        | 설명                                                                |
|-------------|---------------|-------------------------------------------------------------------|
|             |               | page id   slot id ; 컬렉션 타입이면 ' {element 1, element 2, ...}'로 표현됨. |
| is_nullable | VARCHAR(3)    | not null 제약이 설정되어 있으면 'NO', 그렇지 않으면 'YES'                         |
| comment     | VARCHAR(1024) | 속성 설명.                                                            |

다음 예제에서는 클래스 *event* 의 속성과 각 데이터 타입을 검색한다.

```
SELECT attr_name, data_type, domain_class_name
FROM db_attribute
WHERE class_name = 'event'
ORDER BY def_order;
```

| attr_name | data_type | domain_class_name |
|-----------|-----------|-------------------|
| =====     | =====     | =====             |
| 'code'    | 'INTEGER' | NULL              |
| 'sports'  | 'STRING'  | NULL              |
| 'name'    | 'STRING'  | NULL              |
| 'gender'  | 'CHAR'    | NULL              |
| 'players' | 'INTEGER' | NULL              |

다음 예제에서는 클래스 *female\_event* 와 그 상위 클래스의 속성을 검색한다. (ADD SUPERCLASS 절 참조)

```
SELECT attr_name, from_class_name
FROM db_attribute
WHERE class_name = 'female_event'
ORDER BY def_order;
```

| attr_name | from_class_name |
|-----------|-----------------|
| =====     | =====           |
| 'code'    | 'event'         |
| 'sports'  | 'event'         |
| 'name'    | 'event'         |

```
'gender' 'event'
'players' 'event'
```

다음 예제에서는 현재 사용자가 소유하고 있는 클래스 중에서 속성명이 *name* 과 유사한 클래스를 검색한다.

```
/* CURRENT_USER: PUBLIC */
SELECT class_name, attr_name
FROM db_attribute
WHERE owner_name = CURRENT_USER AND attr_name like '%name%'
ORDER BY class_name;
```

```
class_name attr_name
=====
'athlete' 'name'
'code' 'f_name'
'code' 's_name'
'event' 'name'
'nation' 'name'
'stadium' 'name'
```

## DB\_ATTR\_SETDOMAIN\_ELM

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스의 속성 중에서 그 데이터 타입이 컬렉션 타입(SET, MULTISSET, SEQUENCE)인 경우, 그 컬렉션의 원소에 대한 데이터 타입을 보여준다.

| 속성명        | 데이터 타입       | 설명                                                                |
|------------|--------------|-------------------------------------------------------------------|
| attr_name  | VARCHAR(255) | 속성명                                                               |
| class_name | VARCHAR(255) | 속성이 속한 클래스명                                                       |
| owner_name | VARCHAR(32)  | 속성이 속한 클래스 소유자명                                                   |
| attr_type  | VARCHAR(8)   | 인스턴스 속성이면<br>'INSTANCE', 클래스 속성이면<br>'CLASS', 공유 속성이면<br>'SHARED' |

| 속성명               | 데이터 타입       | 설명                                   |
|-------------------|--------------|--------------------------------------|
| data_type         | VARCHAR(9)   | 원소의 데이터 타입                           |
| prec              | INTEGER      | 원소의 데이터 타입에 대한 전체 자릿수                |
| scale             | INTEGER      | 원소의 데이터 타입에 대한 소수점 이하의 자릿수           |
| code_set          | INTEGER      | 원소의 데이터 타입이 문자 타입인 경우 그 문자집합         |
| domain_class_name | VARCHAR(255) | 원소의 데이터 타입이 객체 타입인 경우 그 도메인 클래스명     |
| domain_owner_name | VARCHAR(32)  | 원소의 데이터 타입이 객체 타입인 경우 그 도메인 클래스 소유자명 |

예를 들어 클래스 D의 속성 set\_attr 이 SET(A, B, C) 타입이면 다음 세 개의 레코드들이 존재하게 된다.

| Attr_name  | Class_name | Attr_type  | Data_type | Prec | Scale | Code_set | Domain_class_name |
|------------|------------|------------|-----------|------|-------|----------|-------------------|
| 'set_attr' | 'D'        | 'INSTANCE' | 'SET'     | 0    | 0     | 0        | 'A'               |
| 'set_attr' | 'D'        | 'INSTANCE' | 'SET'     | 0    | 0     | 0        | 'B'               |

| Attr_name  | Class_name | Attr_type  | Data_type | Prec | Scale | Code_set | Domain_class_name |
|------------|------------|------------|-----------|------|-------|----------|-------------------|
| 'set_attr' | 'D'        | 'INSTANCE' | 'SET'     | 0    | 0     | 0        | 'C'               |

다음 예제에서는 클래스 *city* 의 컬렉션 타입의 각 원소의 속성과 데이터 타입을 검색한다. ( 포함 연산자 에 정의한 *city* 테이블을 생성)

```
SELECT attr_name, attr_type, data_type, domain_class_name
FROM db_attr_setdomain_elm
WHERE class_name = 'city';
```

```
attr_name attr_type data_type
domain_class_name
=====
=====
'sports' 'INSTANCE' 'STRING'
NULL
```

DB\_CHARSET  
문자셋에 대한 정보이다.

| 속성명               | 데이터 타입                | 설명           |
|-------------------|-----------------------|--------------|
| charset_id        | INTEGER               | 문자셋 식별자      |
| charset_name      | CHARACTER VARYING(32) | 문자셋 이름       |
| default_collation | CHARACTER VARYING(32) | 기본 콜레이션 이름   |
| char_size         | INTEGER               | 한 문자의 바이트 크기 |

## DB\_COLLATION

콜레이션에 대한 정보이다.

| 속성명            | 데이터 타입       | 설명                                                                                                       |
|----------------|--------------|----------------------------------------------------------------------------------------------------------|
| coll_id        | INTEGER      | 콜레이션 식별자                                                                                                 |
| coll_name      | VARCHAR(32)  | 콜레이션 이름                                                                                                  |
| charset_name   | VARCHAR(32)  | 문자셋 이름                                                                                                   |
| is_builtin     | VARCHAR(3)   | 설치 시 제품 내 포함 여부<br>(Yes, No)                                                                             |
| has_expansions | VARCHAR(3)   | 확장 포함 여부(Yes, No)                                                                                        |
| contractions   | INTEGER      | 축약 포함 여부                                                                                                 |
| uca_strength   | VARCHAR(255) | 가중치 세기(weight strength)<br>(Not applicable, Primary, Secondary, Tertiary, Quaternary, Identity, Unknown) |

## DB\_METHOD

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 그 메서드 정보를 보여준다.

| 속성명        | 데이터 타입       | 설명           |
|------------|--------------|--------------|
| meth_name  | VARCHAR(255) | 메서드명         |
| class_name | VARCHAR(255) | 메서드가 속한 클래스명 |

| 속성명             | 데이터 타입       | 설명                                                                                              |
|-----------------|--------------|-------------------------------------------------------------------------------------------------|
| owner_name      | VARCHAR(32)  | 메서드가 속한 클래스 소유자명                                                                                |
| meth_type       | VARCHAR(8)   | 인스턴스 메서드이면 'INSTANCE', 클래스 메서드이면 'CLASS'                                                        |
| from_class_name | VARCHAR(255) | 상속받은 메서드이면 그 메서드가 정의되어 있는 상위 클래스명이 설정되며 그렇지 않으면 <b>NULL</b>                                     |
| from_owner_name | VARCHAR(32)  | 상속받은 메서드이면 그 메서드가 정의되어 있는 상위 클래스 소유자명이 설정되며 그렇지 않으면 <b>NULL</b>                                 |
| from_meth_name  | VARCHAR(255) | 상속받은 메서드이며, 이름 충돌이 발생하여 이를 해결하기 위해 그 메서드명이 바뀐 경우, 상위 클래스에 정의된 원래 이름이 설정됨. 그 이외에는 모두 <b>NULL</b> |
| func_name       | VARCHAR(255) | 메서드에 대한 C 함수명                                                                                   |

다음 예제에서는 클래스 `_db_user` 의 메서드를 검색한다.

```
SELECT meth_name, meth_type, func_name
FROM db_method
WHERE class_name = '_db_user'
ORDER BY meth_type, meth_name;
```

```
meth_name meth_type func_name
=====
```



```

=====
'add_user' 'CLASS' 'au_add_user_method'
'drop_user' 'CLASS' 'au_drop_user_method'
'find_user' 'CLASS' 'au_find_user_method'
'login' 'CLASS' 'au_login_method'
'add_member' 'INSTANCE' 'au_add_member_method'
'drop_member' 'INSTANCE' 'au_drop_member_method'
'print_authorizations' 'INSTANCE' 'au_describe_user_method'
'set_password' 'INSTANCE' 'au_set_password_method'
'set_password_encoded' 'INSTANCE'
'au_set_password_encoded_method'
'set_password_encoded_sha1' 'INSTANCE'
'au_set_password_encoded_sha1_method'

```

## DB\_METH\_ARG

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스의 메서드에 대해 그 입출력 인자 정보를 보여준다.

| 속성명        | 데이터 타입       | 설명                                          |
|------------|--------------|---------------------------------------------|
| meth_name  | VARCHAR(255) | 메서드명                                        |
| class_name | VARCHAR(255) | 메서드가 속한 클래스명                                |
| owner_name | VARCHAR(32)  | 메서드가 속한 클래스 소유자명                            |
| meth_type  | VARCHAR(8)   | 인스턴스 메서드이면 'INSTANCE', 클래스 메서드이면 'CLASS'    |
| index_of   | INTEGER      | 인자가 함수 정의에 나열된 순서. 리턴 값이면 0, 입력인자이면 1부터 시작함 |
| data_type  | VARCHAR(9)   | 인자의 데이터 타입                                  |

| 속성명               | 데이터 타입       | 설명                                 |
|-------------------|--------------|------------------------------------|
| prec              | INTEGER      | 인자의 전체 자릿수                         |
| scale             | INTEGER      | 인자의 소수점 이하의 자릿수                    |
| code_set          | INTEGER      | 인자의 데이터 타입이 문자 타입인 경우 그 문자집합       |
| domain_class_name | VARCHAR(255) | 인자의 데이터 타입이 객체 타입인 경우 도메인 클래스명     |
| domain_owner_name | VARCHAR(32)  | 인자의 데이터 타입이 객체 타입인 경우 도메인 클래스 소유자명 |

다음 예제에서는 클래스 `_db_user` 의 메서드 입력 인자를 검색한다.

```
SELECT meth_name, data_type, prec
FROM db_meth_arg
WHERE class_name = '_db_user';
```

```
meth_name data_type prec
=====
'append_data' 'STRING' 1073741823
```

#### DB\_METH\_ARG\_SETDOMAIN\_ELM

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스의 메서드에 대해 그 입/출력 인자의 데이터 타입이 집합 타입이면 그 집합의 원소에 대한 데이터 타입을 보여준다.

| 속성명       | 데이터 타입       | 설명   |
|-----------|--------------|------|
| meth_name | VARCHAR(255) | 메서드명 |

| 속성명               | 데이터 타입       | 설명                                          |
|-------------------|--------------|---------------------------------------------|
| class_name        | VARCHAR(255) | 메서드가 속한 클래스명                                |
| owner_name        | VARCHAR(32)  | 메서드가 속한 클래스 소유자명                            |
| meth_type         | VARCHAR(8)   | 인스턴스 메서드이면 'INSTANCE', 클래스 메서드이면 'CLASS'    |
| index_of          | INTEGER      | 인자가 함수 정의에 나열된 순서. 리턴 값이면 0, 입력인자이면 1부터 시작함 |
| data_type         | VARCHAR(9)   | 원소의 데이터 타입                                  |
| prec              | INTEGER      | 원소의 전체 자릿수                                  |
| scale             | INTEGER      | 원소의 소수점 이하의 자릿수                             |
| code_set          | INTEGER      | 원소의 데이터 타입이 문자 타입인 경우 그 문자집합                |
| domain_class_name | VARCHAR(255) | 원소의 데이터 타입이 객체 타입인 경우 도메인 클래스명              |
| domain_owner_name | VARCHAR(32)  | 원소의 데이터 타입이 객체 타입인 경우 도메인 클래스 소유자명          |

## DB\_METH\_FILE

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 그 메서드가 정의된 파일 정보를 보여준다.

| 속성명             | 데이터 타입       | 설명                                                                |
|-----------------|--------------|-------------------------------------------------------------------|
| class_name      | VARCHAR(255) | 메서드 파일이 속한 클래스명                                                   |
| owner_name      | VARCHAR(32)  | 메서드 파일이 속한 클래스 소유자명                                               |
| path_name       | VARCHAR(255) | C 함수가 정의된 파일의 경로                                                  |
| from_class_name | VARCHAR(255) | 상속받은 메서드이면 그 메서드 파일이 정의되어 있는 상위 클래스명이 설정. 그렇지 않으면 <b>NULL</b>     |
| from_owner_name | VARCHAR(32)  | 상속받은 메서드이면 그 메서드 파일이 정의되어 있는 상위 클래스 소유자명이 설정. 그렇지 않으면 <b>NULL</b> |

## DB\_INDEX

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 생성된 인덱스에 대한 정보를 보여준다.

| 속성명        | 데이터 타입       | 설명                           |
|------------|--------------|------------------------------|
| index_name | VARCHAR(255) | 인덱스명                         |
| is_unique  | VARCHAR(3)   | 고유 인덱스이면 'YES', 그렇지 않으면 'NO' |

| 속성명                                | 데이터 타입              | 설명                                          |
|------------------------------------|---------------------|---------------------------------------------|
| is_reverse                         | VARCHAR(3)          | 역 인덱스(reverse indexd)이면 'YES', 그렇지 않으면 'NO' |
| class_name                         | VARCHAR(255)        | 인덱스가 속한 클래스명                                |
| owner_name                         | VARCHAR(32)         | 인덱스가 속한 클래스 소유자명                            |
| key_count                          | INTEGER             | 키를 구성하는 속성의 개수                              |
| is_primary_key                     | VARCHAR(3)          | 기본 키이면 'YES', 그렇지 않으면 'NO'                  |
| is_foreign_key                     | VARCHAR(3)          | 외래 키이면 'YES', 그렇지 않으면 'NO'                  |
| filter_expression                  | VARCHAR(1073741823) | 필터링된 인덱스의 조건                                |
| have_function                      | VARCHAR(3)          | 함수 기반 인덱스이면 'YES', 그렇지 않으면 'NO'             |
| status                             | VARCHAR(255)        | 인덱스 상태                                      |
| referential_index_class_owner_name | VARCHAR(32)         | 외래 키가 참조하는 인덱스의 클래스 소유자명                    |
| referential_index_class_name       | VARCHAR(255)        | 외래 키가 참조하는 인덱스의 클래스명                        |
| referential_index_name             | VARCHAR(255)        | 외래 키가 참조하는 인덱스명                             |

| 속성명                      | 데이터 타입        | 설명                                                             |
|--------------------------|---------------|----------------------------------------------------------------|
| delete_rule              | VARCHAR(32)   | 외래 키 제약 조건의 삭제 규칙 (CASCADE, RESTRICT, NO ACTION, SET NULL)     |
| update_rule              | VARCHAR(32)   | 외래 키 제약 조건의 업데이트 규칙 (RESTRICT, NO ACTION, SET NULL)            |
| referential_match_option | VARCHAR(7)    | 외래키 제약 조건의 일치 옵션 (NONE)<br><br>• NONE: NULL이 하나라도 존재 시 검사하지 않음 |
| index_type               | VARCHAR(32)   | 인덱스 타입 (BTREE)                                                 |
| deduplicate_key_level    | INTEGER       | 인덱스의 중복 제거 레벨                                                  |
| comment                  | VARCHAR(1024) | 인덱스 설명                                                         |
| created_time             | DATETIME      | 인덱스 생성 시각                                                      |
| updated_time             | DATETIME      | 인덱스 수정 시각                                                      |

다음 예제에서는 클래스의 인덱스 정보를 검색한다.

```
SELECT class_name, index_name, is_unique
FROM db_index
ORDER BY owner_name, class_name;
```

[illegible]

```

'u__db_user_name' 'YES'
'athlete'
'pk_athlete_code' 'YES'
'event'
'pk_event_code' 'YES'
'game'
'pk_game_host_year_event_code_athlete_code' 'YES'
'game'
'fk_game_event_code' 'NO'
'game'
'fk_game_athlete_code' 'NO'
'history'
'pk_history_event_code_athlete' 'YES'
'nation'
'pk_nation_code' 'YES'
'olympic'
'pk_olympic_host_year' 'YES'
'participant'
'pk_participant_host_year_nation_code' 'YES'
'participant'
'fk_participant_host_year' 'NO'
'participant'
'fk_participant_nation_code' 'NO'
'record'
'pk_record_host_year_event_code_athlete_code_medal' 'YES'
'stadium'
'pk_stadium_code' 'YES'

```

## DB\_INDEX\_KEY

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대해 생성된 인덱스에 대한 키 정보를 보여준다.

| 속성명        | 데이터 타입       | 설명               |
|------------|--------------|------------------|
| index_name | VARCHAR(255) | 인덱스명             |
| class_name | VARCHAR(255) | 인덱스가 속한 클래스명     |
| owner_name | VARCHAR(32)  | 인덱스가 속한 클래스 소유자명 |

| 속성명               | 데이터 타입        | 설명                                     |
|-------------------|---------------|----------------------------------------|
| key_attr_name     | VARCHAR(255)  | 키를 구성하는 속성의 이름                         |
| key_order         | INTEGER       | 키에서 속성이 위치한 순서. 0부터 시작함                |
| asc_desc          | VARCHAR(4)    | 속성 값의 순서가 내림차순이면 'DESC', 그렇지 않으면 'ASC' |
| key_prefix_length | INTEGER       | 키로 사용할 prefix의 길이                      |
| func              | VARCHAR(1023) | 함수 기반 인덱스의 함수 표현식                      |

다음 예제에서는 클래스의 인덱스 키 정보를 검색한다.

```
SELECT class_name, key_attr_name, index_name
FROM db_index_key
ORDER BY owner_name, class_name, key_order
LIMIT 20;
```

```
class_name key_attr_name index_name
=====
'_db_user' 'name' 'u__db_user_name'
'athlete' 'code' 'pk_athlete_code'
'code' 's_name' 'pk_code'
'event' 'code' 'pk_event_code'
'game' 'athlete_code' 'fk_game_athlete_code'
'game' 'host_year'
'pk_game_host_year_event_code_athlete_code'
'game' 'event_code' 'fk_game_event_code'
'game' 'event_code'
'pk_game_host_year_event_code_athlete_code'
'game' 'athlete_code'
'pk_game_host_year_event_code_athlete_code'
'history' 'event_code'
'pk_history_event_code_athlete'
'history' 'athlete'
'pk_history_event_code_athlete'
'nation' 'code' 'pk_nation_code'
```



```

'olympic' 'host_year' 'pk_olympic_host_year'
'participant' 'host_year'
'pk_participant_host_year_nation_code'
'participant' 'nation_code'
'fk_participant_nation_code'
'participant' 'host_year'
'fk_participant_host_year'
'participant' 'nation_code'
'pk_participant_host_year_nation_code'
'record' 'host_year'
'pk_record_host_year_event_code_athlete_code_medal'
'record' 'event_code'
'pk_record_host_year_event_code_athlete_code_medal'

```

## DB\_AUTH

데이터베이스 내에서 현재 사용자가 권한을 가지는 객체에 대한 권한 정보를 보여준다.

| 속성명          | 데이터 타입       | 설명                                                         |
|--------------|--------------|------------------------------------------------------------|
| grantor_name | VARCHAR(32)  | 권한을 부여한 사용자명                                               |
| grantee_name | VARCHAR(32)  | 권한을 받은 사용자명                                                |
| object_type  | VARCHAR(16)  | 권한부여 대상인 객체 타입명<br>(CLASS, VCLASS,<br>PROCEDURE, FUNCTION) |
| object_name  | VARCHAR(255) | 권한부여 대상인 객체명                                               |
| owner_name   | VARCHAR(32)  | 권한부여 대상인 객체 소유자<br>명                                       |
| auth_type    | VARCHAR(7)   | 부여된 권한 타입명                                                 |
| is_grantable | VARCHAR(3)   | 권한 받은 객체에 대해 다른<br>사용자에게 권한을 부여할 수<br>있으면 'YES', 아니면 'NO'  |

| 속성명          | 데이터 타입   | 설명       |
|--------------|----------|----------|
| created_time | DATETIME | 권한 부여 시각 |
| updated_time | DATETIME | 권한 변경 시각 |

다음 예제에서는 이름이 *db\_a* 로 시작되는 객체의 권한 정보를 검색한다.

```
SELECT object_name, auth_type, grantor_name
FROM db_auth
WHERE object_name like 'db_a%'
ORDER BY owner_name, object_name;
```

```
object_name auth_type grantor_name
=====
'db_attr_setdomain_elm' 'SELECT' 'DBA'
'db_attribute' 'SELECT' 'DBA'
'db_auth' 'SELECT' 'DBA'
'db_authorization' 'SELECT' 'DBA'
```

## DB\_USER

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 사용자에 대한 정보를 보여준다.

| 속성명           | 데이터 타입             | 설명                          |
|---------------|--------------------|-----------------------------|
| name          | VARCHAR(32)        | 사용자명                        |
| id            | INTEGER            | 사용자 식별자                     |
| password      | _db_password       | 항상 <b>NULL</b> (호환성을 위해 유지) |
| direct_groups | SET OF VARCHAR(32) | 사용자가 직접적으로 속한 그룹의 이름 집합     |

| 속성명               | 데이터 타입             | 설명                                                            |
|-------------------|--------------------|---------------------------------------------------------------|
| groups            | SET OF VARCHAR(32) | 사용자가 직·간접적으로 속한 그룹의 이름 집합                                     |
| authorization     | _db_authorization  | 항상 <b>NULL</b> (호환성을 위해 유지)                                   |
| triggers          | SEQUENCE OF object | 사용자의 action에 의해 발생하는 트리거들                                     |
| is_loginable      | VARCHAR(3)         | 로그인 허용이면 'YES', 비허용이면 'NO'                                    |
| is_system_created | VARCHAR(3)         | 시스템 생성 사용자이면 'YES' (예: <b>DBA</b> , <b>PUBLIC</b> ), 아니면 'NO' |
| comment           | VARCHAR(1024)      | 사용자 설명                                                        |
| created_time      | DATETIME           | 사용자 생성 시각                                                     |
| updated_time      | DATETIME           | 사용자 수정 시각                                                     |

### 메서드 이름

- **find\_user** ()
- **login** ()

### DB\_AUTHORIZATION

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스에 대한 사용자 권한 정보를 보여준다.

| 속성명    | 데이터 타입      | 설명                                        |
|--------|-------------|-------------------------------------------|
| owner  | VARCHAR(32) | 이 권한 객체의 소유자명                             |
| grants | SEQUENCE    | {권한이 부여된 객체, 객체의 권한 부여자, 권한 종류}의 sequence |

## DB\_SERIAL

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 시리얼에 대한 정보를 보여준다.

| 속성명           | 데이터 타입        | 설명                               |
|---------------|---------------|----------------------------------|
| unique_name   | VARCHAR(255)  | 스키마명을 포함하는 시리얼명                  |
| name          | VARCHAR(255)  | 시리얼명                             |
| owner         | VARCHAR(32)   | 시리얼 소유자                          |
| current_val   | NUMERIC(38,0) | 현재 시리얼 값                         |
| increment_val | NUMERIC(38,0) | 시리얼 값의 간격                        |
| max_val       | NUMERIC(38,0) | 시리얼의 최대값                         |
| min_val       | NUMERIC(38,0) | 시리얼의 최소값                         |
| start_val     | NUMERIC(38,0) | 시리얼의 시작값                         |
| cyclic        | INTEGER       | 시리얼의 최대 또는 최소값에 도달한 후 순환해서 값을 생성 |

| 속성명          | 데이터 타입        | 설명                                             |
|--------------|---------------|------------------------------------------------|
|              |               | 할 수 있으면 1 (CYCLE), 없으면 0 (NOCYCLE)             |
| started      | INTEGER       | 생성 후 한 번이라도 값을 생성했으면 1, 아니면 0                  |
| class_name   | VARCHAR(255)  | AUTO_INCREMENT 시리얼이면 테이블명을 저장. 아니면 <b>NULL</b> |
| attr_name    | VARCHAR(255)  | AUTO_INCREMENT 시리얼이면 칼럼명을 저장. 아니면 <b>NULL</b>  |
| cached_num   | INTEGER       | 성능 향상을 위해 메모리에 미리 생성할 시리얼 값의 개수.               |
| comment      | VARCHAR(1024) | 시리얼 설명                                         |
| created_time | DATETIME      | 시리얼 생성 시각                                      |
| updated_time | DATETIME      | 시리얼 수정 시각                                      |

## DB\_TRIGGER

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 클래스나 그 소속 속성을 대상(target)으로 하는 트리거 정보를 보여준다.

| 속성명          | 데이터 타입       | 설명   |
|--------------|--------------|------|
| trigger_name | VARCHAR(255) | 트리거명 |

| 속성명               | 데이터 타입        | 설명                                                                                       |
|-------------------|---------------|------------------------------------------------------------------------------------------|
| owner_name        | VARCHAR(32)   | 트리거 소유자명                                                                                 |
| target_class_name | VARCHAR(255)  | 대상이 되는 클래스명                                                                              |
| target_owner_name | VARCHAR(32)   | 대상이 되는 클래스 소유자명                                                                          |
| target_attr_name  | VARCHAR(255)  | 대상이 되는 속성으로서 트리거에 명시되지 않으면 <b>NULL</b>                                                   |
| target_attr_type  | VARCHAR(8)    | 대상이 속성으로 명시될 경우, 인스턴스 속성이면 'INSTANCE', 클래스 속성이면 'CLASS'.                                 |
| action_type       | INTEGER       | INSERT, UPDATE, DELETE, CALL 중 하나이면 1, REJECT이면 2, INVALIDATE_TRANSACTION이면 3, PRINT이면 4 |
| action_time       | INTEGER       | BEFORE는 1, AFTER는 2, DEFERRED는 3으로 설정                                                    |
| comment           | VARCHAR(1024) | 트리거 설명                                                                                   |
| created_time      | DATETIME      | 트리거 생성 시각                                                                                |
| updated_time      | DATETIME      | 트리거 수정 시각                                                                                |

#### DB\_HA\_APPLY\_INFO

데이터베이스 내에서 복제 로그 반영 과정과 그 진행 상태를 나타내는 정보를 보여준다.

| 칼럼명                  | 칼럼 타입         | 의미                                                                                                             |
|----------------------|---------------|----------------------------------------------------------------------------------------------------------------|
| db_name              | VARCHAR(255)  | 로그에 저장된 DB 이름                                                                                                  |
| db_creation_time     | DATETIME      | 반영하는 로그에 대한 원본 DB의 생성 시각                                                                                       |
| copied_log_path      | VARCHAR(4096) | 반영하는 로그 파일의 경로                                                                                                 |
| committed_lsa_pageid | BIGINT        | 마지막에 반영한 commit log lsa의 page id, applylogdb가 재시작해도 last_committed_lsa 보다 앞선 로그는 재반영하지 않음                      |
| committed_lsa_offset | INTEGER       | 마지막에 반영한 commit log lsa의 offset, applylogdb가 재시작해도 last_committed_lsa 보다 앞선 로그는 재반영하지 않음                       |
| committed_rep_pageid | BIGINT        | 마지막 반영한 복제 로그 lsa의 pageid, 복제 반영 지연 여부 확인                                                                      |
| committed_rep_offset | INTEGER       | 마지막 반영한 복제 로그 lsa의 offset, 복제 반영 지연 여부 확인                                                                      |
| append_lsa_page_id   | BIGINT        | 마지막 반영 당시 복제 로그 마지막 lsa의 page id, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 append_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인 |

| 칼럼명               | 칼럼 타입   | 의미                                                                                                            |
|-------------------|---------|---------------------------------------------------------------------------------------------------------------|
| append_lsa_offset | INTEGER | 마지막 반영 당시 복제 로그 마지막 lsa의 offset, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 append_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인 |
| eof_lsa_page_id   | BIGINT  | 마지막 반영 당시 복제 로그 EOF lsa의 page id, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 eof_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인   |
| eof_lsa_offset    | INTEGER | 마지막 반영 당시 복제 로그 EOF lsa의 offset, 복제 반영 당시, applylogdb에서 처리 중인 복제 로그 헤더의 eof_lsa를 저장 복제 로그 반영 당시의 지연 여부를 확인    |
| final_lsa_pageid  | BIGINT  | applylogdb에서 마지막으로 처리한 로그 lsa의 pageid, 복제 반영 지연 여부 확인                                                         |
| final_lsa_offset  | INTEGER | applylogdb에서 마지막으로 처리한 로그 lsa의 offset, 복제 반영 지연 여부 확인                                                         |
| required_page_id  | BIGINT  | log_max_archives 파라미터에 의해 삭제되지 않아야 할 가장 작은 log page id, 복제 반영 시작할 로그 페이지 번호                                   |



| 칼럼명                  | 칼럼 타입    | 의미                                                        |
|----------------------|----------|-----------------------------------------------------------|
| required_page_offset | INTEGER  | 복제 반영 시작할 로그 페이지 offset                                   |
| log_record_time      | DATETIME | 슬레이브 DB에 커밋된 복제 로그에 포함된 timestamp, 즉 해당 로그 레코드 생성 시각      |
| log_commit_time      | DATETIME | 마지막 commit log의 반영 시각                                     |
| last_access_time     | DATETIME | _db_ha_apply_info 카탈로그의 최종 갱신 시각                          |
| status               | INTEGER  | 반영 진행 상태(0: IDLE, 1: BUSY)                                |
| insert_counter       | BIGINT   | applylogdb가 insert한 횟수                                    |
| update_counter       | BIGINT   | applylogdb가 update한 횟수                                    |
| delete_counter       | BIGINT   | applylogdb가 delete한 횟수                                    |
| schema_counter       | BIGINT   | applylogdb가 schema를 변경한 횟수                                |
| commit_counter       | BIGINT   | applylogdb가 commit한 횟수                                    |
| fail_counter         | BIGINT   | applylogdb가 insert/update/delete/commit/schema 변경 중 실패 횟수 |
| start_time           | DATETIME |                                                           |

| 칼럼명 | 칼럼 타입 | 의미                               |
|-----|-------|----------------------------------|
|     |       | applylogdb 프로세스가 슬레이브 DB에 접속한 시각 |

## DB\_PARTITION

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 분할 클래스에 대한 정보를 보여준다.

| 속성명                  | 데이터 타입        | 설명                                                                                   |
|----------------------|---------------|--------------------------------------------------------------------------------------|
| class_name           | VARCHAR(255)  | 클래스명                                                                                 |
| owner_name           | VARCHAR(32)   | 클래스 소유자명                                                                             |
| partition_name       | VARCHAR(255)  | 파티션명                                                                                 |
| partition_class_name | VARCHAR(255)  | 파티션 클래스 명                                                                            |
| partition_type       | VARCHAR(32)   | 파티션 타입 (HASH, RANGE, LIST)                                                           |
| partition_expr       | VARCHAR(2048) | 파티션 표현식                                                                              |
| partition_values     | SEQUENCE OF   | 파티션 유형이 RANGE인 경우 MIN 및 MAX 값이고, MIN 및 MAX가 무한인 경우 <b>NULL</b> 파티션 유형이 LIST인 경우 값 목록 |
| class_partition_type | VARCHAR(32)   |                                                                                      |

| 속성명          | 데이터 타입        | 설명                                                                                  |
|--------------|---------------|-------------------------------------------------------------------------------------|
|              |               | 파티션 클래스의 파티션 타입 (PARTITION CLASS)<br><br>• PARTITION CLASS: 실제 데이터를 저장하는 개별 파티션 테이블 |
| comment      | VARCHAR(1024) | 파티션 설명                                                                              |
| created_time | DATETIME      | 클래스 생성 시각 (부모 클래스로부터 상속)                                                            |
| updated_time | DATETIME      | 클래스 수정 시각 (부모 클래스로부터 상속)                                                            |

다음 예제에서는 `participant2` 클래스의 현재 구성된 분할 정보를 조회한다.

```
SELECT *
FROM db_partition
WHERE class_name = 'participant2';
```

```
class_name owner_name partition_name
partition_class_name partition_type partition_expr
partition_values comment
=====
=====
'participant2' 'PUBLIC' 'before_2000'
'participant2__p__before_2000' 'RANGE' '[host_year]'
{NULL, 2000} NULL
'participant2' 'PUBLIC' 'before_2008'
'participant2__p__before_2008' 'RANGE' '[host_year]'
{2000, 2008} NULL
```

## DB\_STORED\_PROCEDURE

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 저장 프로시저에 대한 정보를 보여준다.

| 속성명              | 데이터 타입              | 설명                                                                      |
|------------------|---------------------|-------------------------------------------------------------------------|
| sp_name          | VARCHAR(255)        | 저장 프로시저 이름                                                              |
| pkg_name         | VARCHAR(255)        | 저장 프로시저가 속한 패키지 이름                                                      |
| sp_type          | VARCHAR(16)         | 저장 프로시저의 종류 (프로시저 또는 함수)                                                |
| return_type      | VARCHAR(16)         | 저장 프로시저의 리턴 타입 이름                                                       |
| arg_count        | INTEGER             | 매개변수 개수                                                                 |
| lang             | VARCHAR(16)         | 저장 프로시저의 구현 언어 이름                                                       |
| authid           | VARCHAR(16)         | 저장 프로시저의 실행 권한                                                          |
| is_deterministic | VARCHAR(3)          | 결정적 함수 여부                                                               |
| target           | VARCHAR(4096)       | 실행할 저장 프로시저의 대상 이름                                                      |
| owner            | VARCHAR(32)         | 소유자                                                                     |
| code             | VARCHAR(1073741823) | 저장 프로시저의 소스 코드                                                          |
| sql_data_access  | VARCHAR(17)         | SQL 데이터 접근 모드 (NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA) |

| 속성명          | 데이터 타입        | 설명            |
|--------------|---------------|---------------|
| comment      | VARCHAR(1024) | 저장 프로시저 설명    |
| created_time | DATETIME      | 저장 프로시저 생성 시각 |
| updated_time | DATETIME      | 저장 프로시저 수정 시각 |

### 참고

#### · **sp\_type**

- **PROCEDURE** 또는 **FUNCTION** 값을 가진다.

#### · **pkg\_name**: 패키지 이름. 현재 패키지 이름은 **DBMS\_OUTPUT** 시스템 패키지에서만 사용한다.

- **dbms\_output** 또는 **NULL** 값을 가진다

#### · **return\_type**: 반환 타입은 아래 값 중 하나이다

```
NULL, INTEGER, FLOAT, DOUBLE, STRING,
SET, MULTISSET, SEQUENCE, TIME,
TIMESTAMP, DATE, SHORT, NUMERIC, CHAR,
CURSOR, BIGINT, DATETIME
```

- 나머지 값은 현재 지원하지 않는다.

#### · **lang**: 저장 프로시저의 구현 언어 이름이며 아래의 값 중 하나이다.

- **PLCSQL**: PL/CSQL
- **JAVA**: Java 저장 프로시저

#### · **authid**: 저장 프로시저의 **execution rights (AUTHID)** 를 나타낸다.

- **DEFINER**: 소유자 권한 (또는 정의자 권한)
- **CURRENT\_USER**: 호출자 권한

#### · **is\_deterministic**: 결정적 함수 여부를 나타낸다.

- **YES**: 결정적 함수

◦ **NO**: 비결정적 함수

다음 예제에서는 현재 사용자가 소유하고 있는 저장 프로시저를 조회한다.

```
CREATE OR REPLACE FUNCTION hello RETURN VARCHAR AS BEGIN RETURN
'Hello'; END;

CREATE OR REPLACE FUNCTION sp_int(p_int INTEGER) RETURN INTEGER AS
BEGIN RETURN p_int; END;

-- csql
;line on

/* CURRENT_USER: PUBLIC */
SELECT * from db_stored_procedure
WHERE sp_type = 'FUNCTION' AND owner = CURRENT_USER;

::

<00001> sp_name : 'hello'
 pkg_name : NULL
 sp_type : 'FUNCTION'
 return_type : 'STRING'
 arg_count : 0
 lang : 'PLCSQL'
 authid : 'DEFINER'
 is_deterministic: 'NO'
 target : 'Func_HELLO_9.HELLO() return
java.lang.String'
 owner : 'DBA'
 code : 'CREATE OR REPLACE FUNCTION
hello RETURN VARCHAR AS BEGIN RETURN 'Hello'; END'
 comment : NULL
<00002> sp_name : 'sp_int'
 pkg_name : NULL
 sp_type : 'FUNCTION'
 return_type : 'INTEGER'
 arg_count : 1
 lang : 'PLCSQL'
 authid : 'DEFINER'
 is_deterministic: 'NO'
 target :
'Func_SP_INT_10.SP_INT(java.lang.Integer) return java.lang.Integer'
 owner : 'DBA'
 code : 'CREATE OR REPLACE FUNCTION
sp_int(p_int INTEGER) RETURN INTEGER AS BEGIN RETURN p_int; END'
 comment : NULL
```

## DB\_STORED\_PROCEDURE\_ARGS

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 저장 프로시저 인자에 대한 정보를 보여준다.

| 속성명           | 데이터 타입        | 설명                       |
|---------------|---------------|--------------------------|
| sp_name       | VARCHAR(255)  | 저장 프로시저 이름               |
| owner_name    | VARCHAR(32)   | 저장 프로시저 소유자의 이름          |
| pkg_name      | VARCHAR(255)  | 저장 프로시저가 속한 패키지 이름       |
| index_of      | INTEGER       | 매개변수의 순서                 |
| arg_name      | VARCHAR(255)  | 매개변수의 이름                 |
| data_type     | VARCHAR(16)   | 매개변수의 데이터 타입             |
| mode          | VARCHAR(6)    | 매개변수 모드 (IN, OUT, INOUT) |
| is_optional   | VARCHAR(3)    | 선택적 매개변수 여부              |
| default_value | VARCHAR(255)  | 매개변수의 기본값                |
| comment       | VARCHAR(1024) | 매개변수에 대한 설명              |

### 참고

- **index\_of**: 매개변수의 순서는 0부터 시작하고 63을 최대값으로 가질 수 있다.
- **data\_type**: 매개변수의 데이터 타입은 아래 값 중 하나이다

NULL, INTEGER, FLOAT, DOUBLE, STRING,  
SET, MULTiset, SEQUENCE, TIME,

TIMESTAMP, DATE, SHORT, NUMERIC, CHAR,  
CURSOR, BIGINT, DATETIME

◦ 나머지 값은 현재 지원하지 않는다.

· **mode**: 매개변수 모드는 아래 값 중 하나이다

IN, OUT, INOUT

· **is\_optional**: 매개변수가 선택 가능하면 **YES**, 필수이면 **NO** 이다.

· **default\_value**: 매개변수의 기본값이다. 자세한 내용은 **기본값 인자 지정** 을 참고한다.

다음 예제에서는 'process\_order' 저장 프로시저의 인수 정보를 순서대로 조회한다.

```
CREATE OR REPLACE PROCEDURE process_order(
 p_order_id IN NUMBER,
 p_status IN VARCHAR2 DEFAULT 'NEW'
)
AS
BEGIN
 DBMS_OUTPUT.PUT_LINE('Processing Order ID: ' || p_order_id);
 DBMS_OUTPUT.PUT_LINE('Order Status: ' || p_status);
END;
```

```
SELECT index_of, arg_name, data_type, mode, is_optional,
 default_value
FROM db_stored_procedure_args
WHERE sp_name = 'process_order'
ORDER BY index_of;
```

::

|       | index_of    | arg_name      | data_type  |
|-------|-------------|---------------|------------|
| mode  | is_optional | default_value |            |
| ===== |             |               |            |
|       |             |               |            |
|       | 0           | 'p_order_id'  | 'NUMBER'   |
| 'IN'  | 'NO'        | NULL          |            |
|       | 1           | 'p_status'    | 'VARCHAR2' |
| 'IN'  | 'YES'       | 'NEW'         |            |



## DB\_SERVER

| Attribute Name   | Data Type     | Description       |
|------------------|---------------|-------------------|
| link_name        | VARCHAR(255)  | 연결 이름             |
| host             | VARCHAR(255)  | 서버의 호스트 이름        |
| port             | INTEGER       | 서버의 연결 포트         |
| db_name          | VARCHAR(255)  | 서버의 데이터베이스 이름     |
| user_name        | VARCHAR(255)  | 서버의 데이터베이스 사용자 이름 |
| properties       | VARCHAR(2048) | 연결 시 사용되는 프로퍼티 정보 |
| owner            | VARCHAR(32)   | 연결 정보의 소유자 이름     |
| comment          | VARCHAR(1024) | 서버 설명             |
| created_time     | DATETIME      | 서버 생성 시각          |
| updated_time     | DATETIME      | 서버 수정 시각          |
| invalidated_time | DATETIME      | 외부 서버 연결이 무효화된 시각 |

## DB\_SYNONYM

데이터베이스 내에서 현재 사용자가 접근 권한을 가진 동의어에 대한 대상 객체 정보를 나타낸다.

| Attribute Name     | Data Type     | Description                                      |
|--------------------|---------------|--------------------------------------------------|
| synonym_name       | VARCHAR(255)  | 동의어 이름                                           |
| synonym_owner_name | VARCHAR(32)   | 동의어의 소유자 이름                                      |
| is_public_synonym  | VARCHAR(3)    | 공용(Public) 동의어이면 “YES”이고, 전용(Private) 동의어이면 “NO” |
| target_name        | VARCHAR(255)  | 대상 객체 이름                                         |
| target_owner_name  | VARCHAR(32)   | 대상 객체의 소유자 이름                                    |
| comment            | VARCHAR(2048) | 동의어 설명                                           |
| created_time       | DATETIME      | 동의어 생성 시각                                        |
| updated_time       | DATETIME      | 동의어 수정 시각                                        |

### ⚠ 경고

아직은 공용(Public) 동의어를 지원하지 않는다.

## 카탈로그 클래스/가상 클래스 사용 권한

카탈로그 클래스들은 **dba** 소유로 생성된다. 그러나, **dba** 가 **SELECT** 연산만 수행할 수 있을 뿐이며, **UPDATE** / **DELETE** 등의 연산을 수행할 경우에는 authorization failure 에러가 발생한다. 일반 사용자는 시스템 카탈로그 클래스에 대해서 질의를 수행할 수 없다.

카탈로그 가상 클래스는 **dba** 소유로 생성되지만 모든 사용자가 카탈로그 가상 클래스에 대해 **SELECT** 문을 수행할 수 있다. 물론 카탈로그 가상 클래스에 대한 **UPDATE** / **DELETE** 연산은 수행이 불가능하다.

카탈로그 클래스/가상 클래스에 대한 갱신은 사용자가 클래스/속성/인덱스/사용자/권한을 생성/변경/삭제하는 DDL 문을 수행할 경우 시스템에 의해 자동으로 수행된다.

## 카탈로그에 대한 질의

클래스, 가상 클래스, 속성, 트리거, 메서드, 인덱스 이름 등과 같은 식별자(identifier)는 모두 소문자로 변경되어 시스템 카탈로그에 저장된다. 따라서 시스템 카탈로그 클래스에 대해 대상 식별자를 검색하려면 소문자를 사용해야 한다. 반면, DB 사용자 이름은 대문자로 변경되어 `_db_user` 시스템 카탈로그 테이블에 저장된다.

```
CREATE TABLE Foo(name varchar(255));
SELECT class_name, partitioned FROM db_class WHERE class_name =
'Foo';
```

There are no results.

```
SELECT class_name, partitioned FROM db_class WHERE class_name =
'foo';
```

| class_name | partitioned |
|------------|-------------|
| 'foo'      | 'NO'        |

```
CREATE USER tester PASSWORD 'testpwd';
SELECT name, password FROM _db_user;
```

| name     | password     |
|----------|--------------|
| 'DBA'    | NULL         |
| 'PUBLIC' | NULL         |
| 'TESTER' | _db_password |

## Information Schema

**Information Schema**는 현재 데이터베이스의 객체(테이블, 칼럼, 제약 조건, 루틴, 권한 등)에 대한 메타데이터를 제공하는 읽기 전용 뷰의 집합이다. Information Schema는 SQL 표준(ISO/IEC 9075-11)으로 정의되어 있어 이식성이 높고 안정적이다. 이는 CUBRID에 특화되어 있고 구현상의 고려 사항을 반영하여 설계된 **시스템 카탈로그**와 대비된다.

```
-- 사용자 'PUBLIC'이 소유한 테이블
SELECT table_name, table_type
FROM information_schema.tables
WHERE table_schema = 'PUBLIC';

-- 'game' 테이블의 칼럼
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'game'
ORDER BY ordinal_position;
```

## Information Schema 뷰

다음 표는 Information Schema 뷰를 요약한 것이다. 각 뷰의 자세한 내용은 아래 설명을 참고한다.

| 뷰 이름                    | 설명             |
|-------------------------|----------------|
| column_privileges       | 칼럼에 정의된 권한     |
| columns                 | 각 테이블의 칼럼      |
| domains                 | 사용자 정의 도메인     |
| foreign_servers         | 외부 서버 정보       |
| key_column_usage        | 제약 조건이 있는 키 칼럼 |
| parameters              | 루틴의 매개변수       |
| partitions              | 테이블 파티션 정보     |
| referential_constraints | 외래 키 정보        |
| routine_privileges      | 루틴에 정의된 권한     |

| 뷰 이름                           | 설명            |
|--------------------------------|---------------|
| <code>routines</code>          | 루틴 정보         |
| <code>schemata</code>          | 스키마 정보        |
| <code>sequences</code>         | 시퀀스 정보        |
| <code>statistics</code>        | 테이블 인덱스 통계    |
| <code>synonyms</code>          | 동의어 정보        |
| <code>table_constraints</code> | 제약 조건이 있는 테이블 |
| <code>table_privileges</code>  | 테이블에 정의된 권한   |
| <code>tables</code>            | 테이블 정보        |
| <code>triggers</code>          | 트리거 정보        |
| <code>views</code>             | 뷰 정보          |

### column\_privileges

`column_privileges` 뷰는 현재 사용자가 부여하거나 부여받은 칼럼 권한을 제공한다. CUBRID는 칼럼 단위 권한을 지원하지 않으므로 이 뷰는 항상 비어 있다.

| 칼럼명                  | 데이터 타입      | 설명 |
|----------------------|-------------|----|
| <code>grantor</code> | VARCHAR(32) |    |

| 칼럼명            | 데이터 타입       | 설명 |
|----------------|--------------|----|
| grantee        | VARCHAR(32)  |    |
| table_catalog  | VARCHAR(255) |    |
| table_schema   | VARCHAR(32)  |    |
| table_name     | VARCHAR(255) |    |
| column_name    | VARCHAR(255) |    |
| privilege_type | VARCHAR(7)   |    |
| is_grantable   | VARCHAR(3)   |    |

## columns

`columns` 뷰는 현재 데이터베이스의 모든 테이블과 뷰의 칼럼 정보를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 테이블의 칼럼만 표시된다.

| 칼럼명           | 데이터 타입       | 설명                              |
|---------------|--------------|---------------------------------|
| table_catalog | VARCHAR(255) | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| table_schema  | VARCHAR(32)  | 테이블이 속한 스키마 이름                  |
| table_name    | VARCHAR(255) | 테이블 이름                          |
| column_name   | VARCHAR(255) | 칼럼 이름                           |

| 칼럼명                      | 데이터 타입        | 설명                                        |
|--------------------------|---------------|-------------------------------------------|
| ordinal_position         | INTEGER       | 테이블 내 칼럼의 순서 위치(1부터 시작)                   |
| column_default           | VARCHAR(2048) | 칼럼의 기본값 표현식                               |
| is_nullable              | VARCHAR(3)    | NOT NULL 제약이 설정되어 있으면 'NO', 그렇지 않으면 'YES' |
| data_type                | VARCHAR(16)   | 칼럼의 데이터 타입                                |
| character_maximum_length | BIGINT        | 문자형 또는 비트열 타입의 최대 문자 길이(그 외 타입은 NULL)     |
| character_octet_length   | BIGINT        | 문자형 타입의 최대 바이트 길이(그 외 타입은 NULL)           |
| numeric_precision        | INTEGER       | 숫자 타입의 정밀도(그 외 타입은 NULL)                  |
| numeric_scale            | INTEGER       | 정확한 숫자 타입의 소수 자릿수(그 외 타입은 NULL)           |
| datetime_precision       | INTEGER       | 날짜/시간 타입의 소수 초 정밀도(그 외 타입은 NULL)          |
| character_set_name       | VARCHAR(32)   | 문자형 타입의 문자 집합 이름(그 외 타입은 NULL)            |
| collation_name           | VARCHAR(32)   | 문자형 타입의 콜레이션 이름(그 외 타입은 NULL)             |

| 칼럼명            | 데이터 타입       | 설명                                                                                                                                                                                                                                 |
|----------------|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| domain_catalog | VARCHAR(255) | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| domain_schema  | VARCHAR(32)  | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| domain_name    | VARCHAR(255) | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| udt_catalog    | VARCHAR(255) | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| udt_schema     | VARCHAR(32)  | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| udt_name       | VARCHAR(255) | 미지원(항상 NULL)                                                                                                                                                                                                                       |
| column_type    | STRING       | 칼럼의 데이터 타입.<br>data_type 은 타입 이름만 나<br>타내지만 column_type 은 정<br>밀도나 길이 등의 정보를 함<br>께 나타낸다                                                                                                                                           |
| column_key     | VARCHAR(3)   | 칼럼이 인덱싱된 방식. 'PRI'<br>는 기본 키 또는 다중 칼럼 기<br>본 키의 구성 칼럼, 'UNI' 는 고<br>유 인덱스의 선두 칼럼, 'MUL'<br>은 중복 값을 허용하는 인덱스<br>의 선두 칼럼을 나타낸다. 인<br>덱싱되지 않았거나 다중 칼럼<br>비고유 인덱스의 보조 칼럼이<br>면 빈 문자열이다. 둘 이상 해<br>당하면 'PRI', 'UNI', 'MUL' 순으<br>로 우선한다 |
| extra          | VARCHAR(255) | 공백으로 구분된 플래그<br>(auto_increment,<br>partition_key)                                                                                                                                                                                 |



| 칼럼명                   | 데이터 타입        | 설명                                 |
|-----------------------|---------------|------------------------------------|
| privileges            | VARCHAR(512)  | 미지원(항상 NULL)                       |
| column_comment        | VARCHAR(2048) | 칼럼 설명                              |
| is_generated          | VARCHAR(6)    | 칼럼이 생성 칼럼인지 여부(항상 'NEVER')         |
| generation_expression | STRING        | 미지원(항상 NULL)                       |
| is_updatable          | VARCHAR(3)    | 칼럼이 갱신 가능한지 여부(항상 'YES')           |
| is_visible            | VARCHAR(3)    | 숨김(invisible) 칼럼이면 'NO', 아니면 'YES' |

## domains

**domains** 뷰는 현재 데이터베이스에 정의된 모든 사용자 정의 도메인을 제공한다. CUBRID는 사용자 정의 도메인을 지원하지 않으므로 이 뷰는 항상 비어 있다.

| 칼럼명            | 데이터 타입       | 설명 |
|----------------|--------------|----|
| domain_catalog | VARCHAR(255) |    |
| domain_schema  | VARCHAR(32)  |    |
| domain_name    | VARCHAR(255) |    |
| data_type      | VARCHAR(16)  |    |

| 칼럼명                      | 데이터 타입        | 설명 |
|--------------------------|---------------|----|
| character_maximum_length | BIGINT        |    |
| character_octet_length   | BIGINT        |    |
| character_set_catalog    | VARCHAR(255)  |    |
| character_set_schema     | VARCHAR(32)   |    |
| character_set_name       | VARCHAR(32)   |    |
| collation_catalog        | VARCHAR(255)  |    |
| collation_schema         | VARCHAR(32)   |    |
| collation_name           | VARCHAR(32)   |    |
| numeric_precision        | INTEGER       |    |
| numeric_precision_radix  | INTEGER       |    |
| numeric_scale            | INTEGER       |    |
| datetime_precision       | INTEGER       |    |
| domain_default           | VARCHAR(2048) |    |
| udt_catalog              | VARCHAR(255)  |    |
| udt_schema               | VARCHAR(32)   |    |

| 칼럼명            | 데이터 타입        | 설명 |
|----------------|---------------|----|
| udt_name       | VARCHAR(255)  |    |
| domain_comment | VARCHAR(2048) |    |
| create_time    | DATETIME      |    |
| update_time    | DATETIME      |    |

### foreign\_servers

`foreign_servers` 뷰는 현재 데이터베이스에 정의된 모든 외부 서버를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 서버만 표시된다.

| 칼럼명                          | 데이터 타입       | 설명                                |
|------------------------------|--------------|-----------------------------------|
| foreign_server_catalog       | VARCHAR(255) | 외부 서버가 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| foreign_server_name          | VARCHAR(255) | 외부 서버의 이름                         |
| foreign_data_wrapper_catalog | VARCHAR(255) | 미지원(항상 NULL)                      |
| foreign_data_wrapper_name    | VARCHAR(255) | 미지원(항상 NULL)                      |
| foreign_server_type          | STRING       | 미지원(항상 NULL)                      |
| foreign_server_version       | STRING       | 미지원(항상 NULL)                      |
| authorization_identifier     | VARCHAR(32)  | 외부 서버 소유자명                        |

| 칼럼명               | 데이터 타입        | 설명           |
|-------------------|---------------|--------------|
| server_host       | VARCHAR(255)  | 대상 호스트       |
| server_port       | INTEGER       | 대상 포트        |
| server_database   | VARCHAR(255)  | 대상 데이터베이스 이름 |
| server_user       | VARCHAR(255)  | 접속 사용자 이름    |
| server_properties | VARCHAR(2048) | 접속 속성        |
| server_comment    | VARCHAR(1024) | 서버 설명        |
| create_time       | DATETIME      | 외부 서버 생성 시각  |
| update_time       | DATETIME      | 외부 서버 수정 시각  |

### key\_column\_usage

`key_column_usage` 뷰는 현재 데이터베이스에서 PRIMARY KEY, UNIQUE, FOREIGN KEY 제약 조건으로 제약된 모든 칼럼을 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 테이블의 칼럼만 표시된다.

| 칼럼명                | 데이터 타입       | 설명                                |
|--------------------|--------------|-----------------------------------|
| constraint_catalog | VARCHAR(255) | 제약 조건이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| constraint_schema  | VARCHAR(32)  | 제약 조건이 속한 스키마 이름                  |

| 칼럼명                           | 데이터 타입       | 설명                                                           |
|-------------------------------|--------------|--------------------------------------------------------------|
| constraint_name               | VARCHAR(255) | 제약 조건 이름                                                     |
| table_catalog                 | VARCHAR(255) | 이 제약 조건으로 제약되는 칼럼이 속한 테이블의 데이터베이스 이름(항상 현재 데이터베이스)           |
| table_schema                  | VARCHAR(32)  | 이 제약 조건으로 제약되는 칼럼이 속한 테이블의 스키마 이름                            |
| table_name                    | VARCHAR(255) | 이 제약 조건으로 제약되는 칼럼이 속한 테이블의 이름                                |
| column_name                   | VARCHAR(255) | 이 제약 조건으로 제약되는 칼럼의 이름                                        |
| ordinal_position              | INTEGER      | 제약 조건 키 내 칼럼의 순서 위치(1부터 시작)                                  |
| position_in_unique_constraint | INTEGER      | 외래 키 제약 조건인 경우 참조되는 고유 제약 조건 내 칼럼의 순서 위치(1부터 시작), 그 외에는 NULL |
| referenced_table_schema       | VARCHAR(32)  | 제약 조건이 참조하는 스키마 이름(외래 키가 아니면 NULL)                           |
| referenced_table_name         | VARCHAR(255) | 제약 조건이 참조하는 테이블 이름(외래 키가 아니면 NULL)                           |
| referenced_column_name        | VARCHAR(255) | 제약 조건이 참조하는 칼럼 이름(외래 키가 아니면 NULL)                            |

## parameters

`parameters` 뷰는 현재 데이터베이스의 모든 프로시저와 함수의 매개변수 정보를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 루틴의 매개변수만 표시된다.

| 칼럼명                      | 데이터 타입       | 설명                                    |
|--------------------------|--------------|---------------------------------------|
| specific_catalog         | VARCHAR(255) | 루틴이 속한 데이터베이스 이름(항상 현재 데이터베이스)        |
| specific_schema          | VARCHAR(32)  | 루틴이 속한 스키마 이름                         |
| specific_name            | VARCHAR(255) | 루틴의 고유 이름                             |
| ordinal_position         | INTEGER      | 루틴의 인자 목록 내 매개변수의 순서 위치(1부터 시작)       |
| parameter_mode           | VARCHAR(5)   | 매개변수 모드 (IN: 입력, OUT: 출력, INOUT: 입출력) |
| is_result                | VARCHAR(3)   | 매개변수가 함수의 반환값인지 여부(항상 'NO')           |
| parameter_name           | VARCHAR(255) | 매개변수 이름                               |
| data_type                | VARCHAR(16)  | 매개변수의 데이터 타입                          |
| character_maximum_length | BIGINT       | 문자열 매개변수의 최대 문자 길이(항상 NULL)           |
| character_octet_length   | BIGINT       | 문자열 매개변수의 최대 바이트 길이(항상 NULL)          |
| character_set_name       | VARCHAR(32)  |                                       |

| 칼럼명                | 데이터 타입        | 설명                                   |
|--------------------|---------------|--------------------------------------|
|                    |               | 문자열 매개변수의 문자 집합<br>(항상 NULL)         |
| collation_name     | VARCHAR(32)   | 문자열 매개변수의 콜레이션<br>(항상 NULL)          |
| numeric_precision  | INTEGER       | 숫자 매개변수의 정밀도(항상<br>NULL)             |
| numeric_scale      | INTEGER       | 숫자 매개변수의 소수 자릿수<br>(항상 NULL)         |
| datetime_precision | INTEGER       | 날짜/시간 타입의 소수 초 정<br>밀도(그 외 타입은 NULL) |
| dtd_identifier     | VARCHAR(1024) | 미지원(항상 NULL)                         |
| routine_type       | VARCHAR(9)    | 루틴의 종류 (PROCEDURE,<br>FUNCTION)      |
| parameter_default  | VARCHAR(2048) | 매개변수의 기본값 표현식(없<br>으면 NULL)          |
| parameter_comment  | VARCHAR(1024) | 매개변수 설명                              |

## partitions

`partitions` 뷰는 현재 데이터베이스의 파티션된 테이블의 모든 파티션을 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 테이블의 파티션만 표시된다.

| 칼럼명                           | 데이터 타입        | 설명                                   |
|-------------------------------|---------------|--------------------------------------|
| table_catalog                 | VARCHAR(255)  | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스)      |
| table_schema                  | VARCHAR(32)   | 테이블이 속한 스키마 이름                       |
| table_name                    | VARCHAR(255)  | 파티션된(상위) 테이블 이름                      |
| partition_name                | VARCHAR(255)  | 파티션 이름                               |
| partition_ordinal_position    | INTEGER       | 테이블 내 파티션의 순서 위치 (항상 NULL)           |
| partition_method              | VARCHAR(5)    | 파티셔닝 방식 (HASH, RANGE, LIST)          |
| partition_expression          | VARCHAR(2048) | 파티션 키 표현식                            |
| partition_description         | STRING        | RANGE/LIST 파티션의 경계 값, HASH 파티션은 NULL |
| subpartition_name             | VARCHAR(255)  | 미지원(항상 NULL)                         |
| subpartition_ordinal_position | INTEGER       | 미지원(항상 NULL)                         |
| subpartition_method           | VARCHAR(5)    | 미지원(항상 NULL)                         |
| subpartition_expression       | VARCHAR(2048) | 미지원(항상 NULL)                         |
| table_rows                    | BIGINT        | 파티션의 추정 행 수                          |



| 칼럼명               | 데이터 타입        | 설명               |
|-------------------|---------------|------------------|
| avg_row_length    | BIGINT        | 추정 평균 행 길이(바이트)  |
| data_length       | BIGINT        | 추정 총 데이터 길이(바이트) |
| data_free         | BIGINT        | 추정 여유 공간(바이트)    |
| tablespace_name   | VARCHAR(255)  | 미지원(항상 NULL)     |
| partition_comment | VARCHAR(1024) | 파티션 설명           |
| create_time       | DATETIME      | 파티션 생성 시각        |
| update_time       | DATETIME      | 파티션 수정 시각        |

### referential\_constraints

`referential_constraints` 뷰는 현재 데이터베이스의 모든 참조(외래 키) 제약 조건을 제공한다. 단, 현재 사용자가 참조하는 테이블을 소유하거나 그에 대해 SELECT 외의 권한을 가진 제약 조건만 표시된다.

| 칼럼명                | 데이터 타입       | 설명                                |
|--------------------|--------------|-----------------------------------|
| constraint_catalog | VARCHAR(255) | 제약 조건이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| constraint_schema  | VARCHAR(32)  | 제약 조건이 속한 스키마 이름                  |
| constraint_name    | VARCHAR(255) | 제약 조건 이름                          |

| 칼럼명                       | 데이터 타입       | 설명                                                      |
|---------------------------|--------------|---------------------------------------------------------|
| unique_constraint_catalog | VARCHAR(255) | 외래 키가 참조하는 고유 또는 기본 키 제약 조건이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| unique_constraint_schema  | VARCHAR(32)  | 외래 키가 참조하는 고유 또는 기본 키 제약 조건이 속한 스키마 이름                  |
| unique_constraint_name    | VARCHAR(255) | 외래 키가 참조하는 고유 또는 기본 키 제약 조건의 이름                         |
| match_option              | VARCHAR(7)   | 외래 키의 매칭 옵션 (NONE, PARTIAL, FULL)                       |
| update_rule               | VARCHAR(9)   | 외래 키의 갱신 규칙 (RESTRICT, NO ACTION, SET NULL)             |
| delete_rule               | VARCHAR(9)   | 외래 키의 삭제 규칙 (CASCADE, RESTRICT, NO ACTION, SET NULL)    |
| table_name                | VARCHAR(255) | 참조하는 테이블                                                |
| referenced_table_name     | VARCHAR(255) | 참조되는 테이블                                                |

## routine\_privileges

`routine_privileges` 뷰는 현재 사용자가 부여하거나 부여받은 프로시저와 함수에 대한 권한을 제공한다. 루틴, 부여자(grantor), 피부여자(grantee)의 조합마다 한 행이 있다.

| 칼럼명              | 데이터 타입       | 설명                                       |
|------------------|--------------|------------------------------------------|
| grantor          | VARCHAR(32)  | 권한을 부여한 사용자                              |
| grantee          | VARCHAR(32)  | 권한을 부여받은 사용자                             |
| specific_catalog | VARCHAR(255) | 루틴이 속한 데이터베이스 이름(항상 현재 데이터베이스)           |
| specific_schema  | VARCHAR(32)  | 루틴이 속한 스키마 이름                            |
| specific_name    | VARCHAR(255) | 루틴의 고유 이름                                |
| routine_catalog  | VARCHAR(255) | 루틴이 속한 데이터베이스 이름(항상 현재 데이터베이스)           |
| routine_schema   | VARCHAR(32)  | 루틴이 속한 스키마 이름                            |
| routine_name     | VARCHAR(255) | 루틴 이름                                    |
| privilege_type   | VARCHAR(7)   | 권한의 종류(항상 EXECUTE)                       |
| is_grantable     | VARCHAR(3)   | 피부여자가 권한을 다른 사용자에게 부여할 수 있는지 여부(항상 'NO') |

## routines

**routines** 뷰는 현재 데이터베이스의 모든 프로시저와 함수를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 루틴만 표시된다.

| 칼럼명                      | 데이터 타입       | 설명                             |
|--------------------------|--------------|--------------------------------|
| specific_catalog         | VARCHAR(255) | 루틴이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| specific_schema          | VARCHAR(32)  | 루틴이 속한 스키마 이름                  |
| specific_name            | VARCHAR(255) | 루틴의 고유 이름                      |
| routine_catalog          | VARCHAR(255) | 루틴이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| routine_schema           | VARCHAR(32)  | 루틴이 속한 스키마 이름                  |
| routine_name             | VARCHAR(255) | 루틴 이름                          |
| routine_type             | VARCHAR(9)   | 루틴의 종류 (PROCEDURE, FUNCTION)   |
| data_type                | VARCHAR(16)  | 함수의 반환 데이터 타입(프로시저는 NULL)      |
| character_maximum_length | BIGINT       | 문자열 반환값의 최대 문자 길이(항상 NULL)     |
| character_octet_length   | BIGINT       | 문자열 반환값의 최대 바이트 길이(항상 NULL)    |
| character_set_name       | VARCHAR(32)  | 문자열 반환 타입의 경우 반환값의 문자 집합       |
| collation_name           | VARCHAR(32)  | 문자열 반환값의 콜레이션(항상 NULL)         |

| 칼럼명                | 데이터 타입       | 설명                                                                      |
|--------------------|--------------|-------------------------------------------------------------------------|
| numeric_precision  | INTEGER      | 숫자 반환값의 정밀도(항상 NULL)                                                    |
| numeric_scale      | INTEGER      | 숫자 반환값의 소수 자릿수(항상 NULL)                                                 |
| datetime_precision | INTEGER      | 날짜/시간 타입의 소수 초 정밀도(그 외 타입은 NULL)                                        |
| dtd_identifier     | VARCHAR(255) | 미지원(항상 NULL)                                                            |
| routine_body       | VARCHAR(8)   | 루틴 본문의 유형(항상 EXTERNAL)                                                  |
| routine_definition | STRING       | PL/CSQL 소스 텍스트                                                          |
| external_name      | STRING       | 루틴의 외부 이름(없으면 NULL)                                                     |
| external_language  | VARCHAR(64)  | 루틴의 구현 언어 (PLCSQL, JAVA)                                                |
| parameter_style    | VARCHAR(3)   | 루틴의 매개변수 전달 방식(항상 SQL)                                                  |
| is_deterministic   | VARCHAR(3)   | DETERMINISTIC으로 선언되었으면 'YES', 아니면 'NO'                                  |
| sql_data_access    | VARCHAR(17)  | SQL 데이터 접근 수준 (NO SQL, CONTAINS SQL, READS SQL DATA, MODIFIES SQL DATA) |

| 칼럼명             | 데이터 타입        | 설명                                  |
|-----------------|---------------|-------------------------------------|
| sql_path        | STRING        | 미지원(항상 NULL)                        |
| security_type   | VARCHAR(7)    | 루틴이 실행되는 보안 컨텍스트 (INVOKER, DEFINER) |
| routine_comment | VARCHAR(1024) | 루틴 설명                               |
| created         | DATETIME      | 루틴 생성 시각                            |
| last_altered    | DATETIME      | 루틴 수정 시각                            |
| definer         | VARCHAR(32)   | 루틴의 소유자                             |

#### 참고

`routine_definition` 은 PL/CSQL 소스 코드를 노출하므로 DBA 또는 루틴 소유자에게만 표시된다. 그 외 사용자에게는 NULL이다.

## schemata

`schemata` 뷰는 현재 데이터베이스의 모든 스키마를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 스키마만 표시된다.

| 칼럼명          | 데이터 타입       | 설명                              |
|--------------|--------------|---------------------------------|
| catalog_name | VARCHAR(255) | 스키마가 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| schema_name  | VARCHAR(32)  | 스키마 이름                          |

| 칼럼명                           | 데이터 타입        | 설명                                   |
|-------------------------------|---------------|--------------------------------------|
| schema_owner                  | VARCHAR(32)   | 스키마 소유자명                             |
| default_character_set_catalog | VARCHAR(255)  | 기본 문자 집합이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| default_character_set_schema  | VARCHAR(255)  | 기본 문자 집합이 속한 스키마 이름(항상 NULL)         |
| default_character_set_name    | VARCHAR(32)   | 데이터베이스의 기본 문자 집합                     |
| sql_path                      | STRING        | 미지원(항상 NULL)                         |
| schema_comment                | VARCHAR(1024) | 스키마 설명                               |
| create_time                   | DATETIME      | 스키마 생성 시각                            |

#### 참고

CUBRID에서 스키마는 데이터베이스 사용자와 동일하다.

## sequences

[sequences](#) 뷰는 현재 데이터베이스에 정의된 모든 시퀀스를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 시퀀스만 표시된다.

| 칼럼명              | 데이터 타입       | 설명 |
|------------------|--------------|----|
| sequence_catalog | VARCHAR(255) |    |

| 칼럼명                     | 데이터 타입        | 설명                              |
|-------------------------|---------------|---------------------------------|
|                         |               | 시퀀스가 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| sequence_schema         | VARCHAR(32)   | 시퀀스가 속한 스키마 이름                  |
| sequence_name           | VARCHAR(255)  | 시퀀스 이름                          |
| data_type               | VARCHAR(16)   | 시퀀스의 데이터 타입(항상 NUMERIC)         |
| numeric_precision       | INTEGER       | 시퀀스 데이터 타입의 정밀도(항상 38)          |
| numeric_precision_radix | INTEGER       | 정밀도가 표현되는 기수(항상 10)             |
| numeric_scale           | INTEGER       | 시퀀스 데이터 타입의 소수 자리 수(항상 0)       |
| start_value             | NUMERIC(38,0) | 시퀀스의 시작값                        |
| minimum_value           | NUMERIC(38,0) | 시퀀스의 최솟값                        |
| maximum_value           | NUMERIC(38,0) | 시퀀스의 최댓값                        |
| increment               | NUMERIC(38,0) | 시퀀스의 증가값                        |
| cycle_option            | VARCHAR(3)    | 시퀀스가 순환하면 'YES', 아니면 'NO'       |
| is_cached               | VARCHAR(3)    |                                 |



| 칼럼명              | 데이터 타입        | 설명                            |
|------------------|---------------|-------------------------------|
|                  |               | 캐시가 활성화되어 있으면 'YES', 아니면 'NO' |
| sequence_comment | VARCHAR(1024) | 시퀀스 설명                        |
| create_time      | DATETIME      | 시퀀스 생성 시각                     |
| update_time      | DATETIME      | 시퀀스 수정 시각                     |

#### 참고

CUBRID의 **시리얼** 객체는 이 뷰에 시퀀스로 표시된다. AUTO\_INCREMENT 칼럼에 연결된 시퀀스는 표시되지 않는다.

## statistics

**statistics** 뷰는 현재 데이터베이스의 테이블 인덱스 정보를 제공한다. 각 인덱스의 칼럼마다 한 행이 있다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 테이블의 인덱스만 표시된다.

| 칼럼명           | 데이터 타입       | 설명                              |
|---------------|--------------|---------------------------------|
| table_catalog | VARCHAR(255) | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| table_schema  | VARCHAR(32)  | 테이블이 속한 스키마 이름                  |
| table_name    | VARCHAR(255) | 테이블 이름                          |

| 칼럼명           | 데이터 타입        | 설명                                                  |
|---------------|---------------|-----------------------------------------------------|
| non_unique    | INTEGER       | 인덱스가 중복을 허용하지 않으면 0, 허용하면 1                         |
| index_schema  | VARCHAR(32)   | 인덱스가 속한 스키마 이름                                      |
| index_name    | VARCHAR(255)  | 인덱스 이름                                              |
| seq_in_index  | INTEGER       | 인덱스 내 칼럼의 순서 위치(1부터 시작)                             |
| column_name   | VARCHAR(255)  | 인덱스에 포함된 칼럼 이름                                      |
| collation     | VARCHAR(1)    | 인덱스 내 칼럼의 정렬 순서<br>(A: 오름차순, D: 내림차순)               |
| cardinality   | INTEGER       | 인덱스의 추정 고유 값 수                                      |
| sub_part      | INTEGER       | 칼럼을 접두(prefix) 인덱싱한 경우 인덱싱된 문자 수, 전체 칼럼을 인덱싱하면 NULL |
| nullable      | VARCHAR(3)    | 인덱스 칼럼이 NULL을 허용하면 'YES', 아니면 'NO'                  |
| index_type    | VARCHAR(32)   | 인덱스 방식(항상 BTREE)                                    |
| comment       | VARCHAR(1024) | 시스템이 생성하는 인덱스 정보(CUBRID에서는 항상 NULL)                 |
| index_comment | VARCHAR(1024) | 인덱스 설명                                              |

| 칼럼명               | 데이터 타입        | 설명                                  |
|-------------------|---------------|-------------------------------------|
| is_visible        | VARCHAR(3)    | 인덱스가 옵티마이저에 보이<br>면 'YES', 아니면 'NO' |
| expression        | VARCHAR(1023) | 함수 기반 인덱스의 표현식(일<br>반 칼럼은 NULL)     |
| filter_condition  | STRING        | 필터 인덱스의 조건식(필터 인<br>덱스가 아니면 NULL)   |
| deduplicate_level | INTEGER       | 인덱스의 중복 제거<br>(deduplicate) 옵션 레벨   |
| create_time       | DATETIME      | 인덱스 생성 시각                           |
| update_time       | DATETIME      | 인덱스 수정 시각                           |
| access_time       | DATETIME      | 미지원(항상 NULL)                        |

## synonyms

**synonyms** 뷰는 현재 데이터베이스에 정의된 모든 동의어를 제공한다. 단, 현재 사용자가 소유하거나 공용(public)인 동의어만 표시된다.

| 칼럼명             | 데이터 타입       | 설명                                     |
|-----------------|--------------|----------------------------------------|
| synonym_catalog | VARCHAR(255) | 동의어가 속한 데이터베이스<br>이름(항상 현재 데이터베이<br>스) |
| synonym_schema  | VARCHAR(32)  | 동의어가 속한 스키마 이름                         |

| 칼럼명               | 데이터 타입        | 설명                                |
|-------------------|---------------|-----------------------------------|
| synonym_name      | VARCHAR(255)  | 동의어 이름                            |
| is_public_synonym | VARCHAR(3)    | 공용 동의어이면 'YES', 아니면 'NO'          |
| target_catalog    | VARCHAR(255)  | 대상 객체가 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| target_schema     | VARCHAR(32)   | 대상 객체가 속한 스키마 이름                  |
| target_name       | VARCHAR(255)  | 대상 객체의 이름                         |
| synonym_comment   | VARCHAR(2048) | 동의어 설명                            |
| create_time       | DATETIME      | 동의어 생성 시각                         |
| update_time       | DATETIME      | 동의어 수정 시각                         |

## table\_constraints

`table_constraints` 뷰는 현재 사용자가 소유하거나 SELECT 외의 권한을 가진 테이블에 속한 모든 PRIMARY KEY, UNIQUE, FOREIGN KEY 제약 조건을 제공한다.

| 칼럼명                | 데이터 타입       | 설명                                |
|--------------------|--------------|-----------------------------------|
| constraint_catalog | VARCHAR(255) | 제약 조건이 속한 데이터베이스 이름(항상 현재 데이터베이스) |

| 칼럼명                | 데이터 타입       | 설명                                           |
|--------------------|--------------|----------------------------------------------|
| constraint_schema  | VARCHAR(32)  | 제약 조건이 속한 스키마 이름                             |
| constraint_name    | VARCHAR(255) | 제약 조건 이름                                     |
| table_catalog      | VARCHAR(255) | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스)              |
| table_schema       | VARCHAR(32)  | 테이블이 속한 스키마 이름                               |
| table_name         | VARCHAR(255) | 테이블 이름                                       |
| constraint_type    | VARCHAR(11)  | 제약 조건의 종류 (PRIMARY KEY, UNIQUE, FOREIGN KEY) |
| is_deferrable      | VARCHAR(3)   | 제약 조건을 지연할 수 있는지 여부(항상 'NO')                 |
| initially_deferred | VARCHAR(3)   | 제약 조건이 초기 지연 상태인지 여부(항상 'NO')                |
| enforced           | VARCHAR(3)   | 제약 조건이 적용되면 'YES', 아니면 'NO'                  |

#### 참고

PRIMARY KEY, UNIQUE, FOREIGN KEY 제약 조건만 표시되며, CHECK 제약 조건은 표시되지 않는다.

## table\_privileges

`table_privileges` 뷰는 현재 사용자가 부여하거나 부여받은 테이블 또는 뷰 권한을 제공한다. 테이블, 부여자(grantor), 피부여자(grantee)의 조합마다 한 행이 있다.

| 칼럼명            | 데이터 타입       | 설명                                                             |
|----------------|--------------|----------------------------------------------------------------|
| grantor        | VARCHAR(32)  | 권한을 부여한 사용자                                                    |
| grantee        | VARCHAR(32)  | 권한을 부여받은 사용자                                                   |
| table_catalog  | VARCHAR(255) | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스)                                |
| table_schema   | VARCHAR(32)  | 테이블이 속한 스키마 이름                                                 |
| table_name     | VARCHAR(255) | 테이블 이름                                                         |
| privilege_type | VARCHAR(7)   | 권한의 종류 (SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, EXECUTE) |
| is_grantable   | VARCHAR(3)   | 피부여자가 다른 사용자에게 권한을 부여할 수 있으면 'YES', 아니면 'NO'                   |

## tables

`tables` 뷰는 현재 데이터베이스에 정의된 모든 테이블과 뷰를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 테이블과 뷰만 표시된다.

| 칼럼명           | 데이터 타입       | 설명 |
|---------------|--------------|----|
| table_catalog | VARCHAR(255) |    |

| 칼럼명             | 데이터 타입        | 설명                                                                                                                    |
|-----------------|---------------|-----------------------------------------------------------------------------------------------------------------------|
|                 |               | 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스)                                                                                       |
| table_schema    | VARCHAR(32)   | 테이블이 속한 스키마 이름                                                                                                        |
| table_name      | VARCHAR(255)  | 테이블 이름                                                                                                                |
| table_type      | VARCHAR(12)   | 테이블의 종류 (BASE TABLE, VIEW, SYSTEM TABLE, SYSTEM VIEW)                                                                 |
| table_rows      | BIGINT        | 추정 행 수                                                                                                                |
| avg_row_length  | BIGINT        | 추정 평균 행 길이(바이트)                                                                                                       |
| data_length     | BIGINT        | 추정 총 데이터 길이(바이트)                                                                                                      |
| data_free       | BIGINT        | 추정 여유 공간(바이트)                                                                                                         |
| auto_increment  | NUMERIC(38,0) | 다음 AUTO_INCREMENT 값(해당하지 않으면 NULL)                                                                                    |
| table_collation | VARCHAR(32)   | 테이블의 기본 콜레이션                                                                                                          |
| create_options  | VARCHAR(255)  | OID 재사용 설정(REUSE_OID 또는 DONT_REUSE_OID)과 TDE 설정(ENCRYPT=NONE, ENCRYPT=AES, ENCRYPT=ARIA)을 포함하는, 공백으로 구분된 테이블 생성 옵션 목록 |

| 칼럼명                 | 데이터 타입        | 설명                            |
|---------------------|---------------|-------------------------------|
| is_temporary        | VARCHAR(3)    | 테이블이 임시 테이블인지 여부(항상 'NO')     |
| table_comment       | VARCHAR(2048) | 테이블 설명                        |
| create_time         | DATETIME      | 테이블 생성 시각                     |
| update_time         | DATETIME      | 테이블 수정 시각                     |
| statistics_strategy | VARCHAR(8)    | 통계 수집 방식 (SAMPLING, FULLSCAN) |
| last_analyzed       | DATETIME      | 테이블의 통계 정보 갱신 시각              |

## triggers

**triggers** 뷰는 현재 사용자가 소유하거나 SELECT 외의 권한을 가진 테이블에 정의된 모든 트리거를 제공한다.

| 칼럼명             | 데이터 타입       | 설명                              |
|-----------------|--------------|---------------------------------|
| trigger_catalog | VARCHAR(255) | 트리거가 속한 데이터베이스 이름(항상 현재 데이터베이스) |
| trigger_schema  | VARCHAR(32)  | 트리거가 속한 스키마 이름                  |
| trigger_name    | VARCHAR(255) | 트리거 이름                          |



| 칼럼명                        | 데이터 타입       | 설명                                         |
|----------------------------|--------------|--------------------------------------------|
| event_manipulation         | VARCHAR(8)   | 트리거를 발생시키는 이벤트 (INSERT, UPDATE, DELETE)    |
| event_object_catalog       | VARCHAR(255) | 트리거가 정의된 테이블이 속한 데이터베이스 이름(항상 현재 데이터베이스)   |
| event_object_schema        | VARCHAR(32)  | 트리거가 정의된 테이블이 속한 스키마 이름                    |
| event_object_table         | VARCHAR(255) | 트리거가 정의된 테이블의 이름                           |
| event_object_column        | VARCHAR(255) | UPDATE OF 절의 대상 칼럼 (그 외에는 NULL)            |
| action_order               | INTEGER      | 미지원(항상 NULL)                               |
| action_condition           | STRING       | 트리거의 WHEN/IF 조건(없으면 NULL)                  |
| action_statement           | STRING       | 트리거가 실행하는 문장                               |
| action_orientation         | VARCHAR(9)   | 트리거가 행마다 발생하는지 문장마다 발생하는지 (ROW, STATEMENT) |
| action_timing              | VARCHAR(8)   | 트리거가 발생하는 시점 (BEFORE, AFTER, DEFERRED)     |
| action_reference_old_table | VARCHAR(3)   | 미지원(항상 NULL)                               |

| 칼럼명                        | 데이터 타입        | 설명                           |
|----------------------------|---------------|------------------------------|
| action_reference_new_table | VARCHAR(3)    | 미지원(항상 NULL)                 |
| action_reference_old_row   | VARCHAR(3)    | 이전 행을 참조하는 데 사용되는 별칭(항상 OLD) |
| action_reference_new_row   | VARCHAR(3)    | 새 행을 참조하는 데 사용되는 별칭(항상 NEW)  |
| definer                    | VARCHAR(32)   | 트리거의 소유자                     |
| trigger_comment            | VARCHAR(1024) | 트리거 설명                       |
| create_time                | DATETIME      | 트리거 생성 시각                    |
| update_time                | DATETIME      | 트리거 수정 시각                    |

#### 참고

이 뷰에는 INSERT, UPDATE, DELETE (DML) **트리거**만 표시된다. CUBRID는 COMMIT, ROLLBACK 트리거도 지원하지만 여기에는 표시되지 않는다.

## views

**views** 뷰는 현재 데이터베이스에 정의된 모든 뷰를 제공한다. 단, 현재 사용자가 소유하거나 접근 권한을 가진 뷰만 표시된다.

| 칼럼명           | 데이터 타입       | 설명                                |
|---------------|--------------|-----------------------------------|
| table_catalog | VARCHAR(255) | 뷰가 속한 데이터베이스 이름<br>(항상 현재 데이터베이스) |

| 칼럼명             | 데이터 타입        | 설명                                                                          |
|-----------------|---------------|-----------------------------------------------------------------------------|
| table_schema    | VARCHAR(32)   | 뷰가 속한 스키마 이름                                                                |
| table_name      | VARCHAR(255)  | 뷰 이름                                                                        |
| view_definition | STRING        | 뷰를 정의하는 질의 표현식                                                              |
| check_option    | VARCHAR(8)    | 뷰의 WITH CHECK OPTION 설정: CASCADED, LOCAL, 또는 CHECK OPTION이 정의되어 있지 않으면 NONE |
| is_updatable    | VARCHAR(3)    | 뷰가 갱신 가능한지 여부(항상 NULL)                                                      |
| definer         | VARCHAR(32)   | 뷰의 소유자                                                                      |
| view_comment    | VARCHAR(2048) | 뷰 설명                                                                        |
| create_time     | DATETIME      | 뷰 생성 시각                                                                     |
| update_time     | DATETIME      | 뷰 수정 시각                                                                     |

#### 참고

CUBRID는 뷰의 갱신 가능 여부를 고정 값으로 저장하지 않으므로 `is_updatable` 은 항상 NULL이다.

## CUBRID 절차적 언어

이 장에서는 저장 프로시저와 저장 함수를 사용할 수 있도록 CUBRID 에서 제공하는 절차적 언어 확장 기능을 소개한다.

CUBRID 는 절차적 언어 확장 기능으로 저장 프로시저와 저장 함수를 지원한다. 저장 프로시저와 저장 함수를 사용하면, SQL 만으로 구현하기 어려운 복잡한 비즈니스 로직을 효율적으로 처리하고 데이터 조작을 간편하게 수행할 수 있다.

다음의 주요한 장점을 가진다.

- 생산성과 사용성: 저장 프로시저와 저장 함수는 한 번 만들어 두면 여러 번 재사용할 수 있다. 사용자가 SQL 문이나 JDBC를 통해 응용 프로그램에서도 쉽게 호출할 수 있다.
- 성능: 저장 프로시저와 저장 함수는 데이터베이스 서버에서 실행되기 때문에 네트워크 트래픽을 줄이고 데이터베이스 서버의 성능을 향상시킨다.
- 보안: 저장 프로시저는 특정 사용자에게 실행 권한을 부여할 수 있으므로, 데이터 접근 및 수정 권한을 세밀하게 제어할 수 있다.
- 상호 운용성과 이식성: 저장 프로시저와 함수는 다양한 언어 및 실행 환경으로 동작할 수 있도록 설계되어 데이터베이스의 활용도를 극대화하였다. CUBRID는 저장 프로시저/함수를 위해 다음 두 가지 절차적 언어를 지원한다.
  - PL/CSQL
  - Java

이 장에서 설명하는 주요 내용은 다음과 같다.

- 저장 프로시저의 생성: 저장 프로시저와 저장 함수의 생성에 대하여 설명한다.
- 저장 프로시저의 호출: 저장 프로시저와 저장 함수를 호출에 대하여 설명한다.
- 저장 프로시저의 권한: 저장 프로시저에 대한 호출 권한 부여와, 소유자 권한과 호출자 권한의 차이에 대하여 설명한다.
- 트랜잭션 커밋과 롤백: 저장 프로시저 내에서 트랜잭션 커밋과 롤백 사용에 대하여 설명한다.
- 성능 최적화: 저장 프로시저를 사용할 때 최적화된 성능을 얻기 위한 방법에 대해 설명한다.
- 시스템 패키지: CUBRID에서 지원하는 시스템 패키지에 대해 설명한다.
- PL/CSQL: CUBRID의 SQL 절차적 언어의 확장인 PL/CSQL의 개요와 문법에 대해 설명한다.
- 자바 저장 프로시저: 자바 언어를 사용하여 저장 프로시저를 개발하는 방법에 대해 설명한다.
- 메서드: CUBRID의 내장 함수인 메서드에 대해 설명한다.

## 저장 프로시저의 생성

저장 프로시저와 저장 함수는 각각 `CREATE PROCEDURE`와 `CREATE FUNCTION`을 사용하여 생성 할 수 있다. CUBRID에서는 저장 프로시저와 저장 함수에 대해서 기능에 특별한 차이점은 없다. 유일한 차이점은 저장 함수의 경우 반환 타입을 정의하지만, 저장 프로시저는 반환 타입은 `void`로 항상 `NULL` 값을 반환한다.

이 장에서는 저장 프로시저와 저장 함수를 생성할 때 고려해야 할 사항에 대하여 자세히 설명한다.

## 스키마와 프로시저 이름 지정

```
CREATE [OR REPLACE] [PROCEDURE] [schema_name.]procedure_name
CREATE [OR REPLACE] [FUNCTION] [schema_name.]function_name
```

저장 프로시저와 저장 함수의 이름은 식별자에 해당하며 **식별자**의 작성 원칙에 따라 작성해야 한다. 이름의 최대 222 바이트 길이로 제한되며, 이 길이를 초과하면 오류가 발생한다.

저장 프로시저와 저장 함수의 이름을 지정할 때 스키마 이름(**schema\_name**)을 지정하지 않아도 현재 로그인 된 **사용자의 스키마**에 저장되어 생성된다. DBA 권한이 있는 사용자는 다른 사용자의 스키마에 저장 프로시저와 저장 함수를 생성할 수 있지만, 일반 사용자의 경우 자신의 스키마에만 저장 프로시저와 저장 함수를 생성할 수 있다.

만약 일반 사용자가 다른 사용자의 스키마에 저장 프로시저와 저장 함수를 생성하려고 시도하면 에러가 발생한다.

```
-- Login as DBA
CREATE USER U1;
CREATE USER U2;

CALL login ('U1', '') ON CLASS db_user;

CREATE PROCEDURE u2.test_proc (arg1 INT, arg2 INT)
AS BEGIN
 DBMS_OUTPUT.put_line (arg1 + arg2);
END;
```

```
ERROR: before ' ; '
DBA, members of DBA group, and owner can perform CREATE PROCEDURE/
FUNCTION.
```

## 저장 프로시저의 인자 지정

저장 프로시저와 저장 함수에 인자를 지정하면 호출 시 인수를 전달할 수 있다. 인자를 지정할 때에는 인자의 이름과 데이터 타입을 지정해야 하며 최대 **64**개의 인자를 지정할 수 있다.

저장 함수는 입력 매개 변수를 받아 결과값을 반환하므로 결과 데이터 타입을 지정해야 하며, 저장 프로시저는 결과 데이터 타입을 지정하지 않고 항상 NULL 을 반환하는 void 타입으로 간주한다. 저장 프로시저와 저장 함수의 결과 타입으로 CUBRID SQL이 지원하는 (**데이터 타입**) 중 일부만을 지원한다. 자세한 내용은 **지원하는 인자와 결과 데이터 타입**을 참고한다.

```
CREATE PROCEDURE test_proc (arg1 INT, arg2 INT)
AS BEGIN
 DBMS_OUTPUT.put_line (arg1 + arg2);
END;

CREATE FUNCTION test_func (arg1 INT, arg2 INT) RETURN INT
AS BEGIN
 RETURN arg1 + arg2;
END;

SELECT test_proc(1, 2);
SELECT test_func(1, 2);
```

```
test_proc(1, 2)
=====
NULL

<DBMS_OUTPUT>
====
3

test_func(1, 2)
=====
3
```

### 참고

저장 프로시저와 저장 함수의 인자 또는 반환 값의 데이터 타입의 이름 만을 명시할 수 있다. 즉, VARCHAR 또는 NUMERIC 와 같은 타입에 정밀도와 스케일을 지정하지 않는다. 실행 시 실제 전달 되는 데이터에 따라 가변적으로 정밀도와 스케일이 결정된다.

```
create or replace funtion t (i_min VARCHAR, i_max VARCHAR) RETURN
VARCHAR as
 var_min VARCHAR(5) := i_min;
```

```

var_max VARCHAR(5) := i_max;
begin
 return var_min || ',' || var_max;
end;

```

## 지원하는 인자와 결과 데이터 타입

저장 프로시저와 저장 함수의 인수와 결과 데이터 타입으로 CUBRID SQL이 지원하는 데이터 타입 중 일부 데이터 타입을 명시할 수 있다. 다음의 표는 언어 확장에서 지원하는 데이터 타입이다.

| 유형    | 타입                                                  | 지원 여부 (O, X) |         |
|-------|-----------------------------------------------------|--------------|---------|
|       |                                                     | PL/CSQL      | Java SP |
| 수치    | SHORT, SMALLINT                                     | O            | O       |
|       | INTEGER, INT                                        | O            | O       |
|       | BIGINT                                              | O            | O       |
|       | NUMERIC, DECIMAL                                    | O            | O       |
|       | FLOAT, REAL                                         | O            | O       |
|       | DOUBLE, DOUBLE PRECISION                            | O            | O       |
| 날짜/시간 | DATE, TIME, TIMESTAMP, DATETIME                     | O            | O       |
|       | TIMESTAMPPLTZ, TIMESTAMPTZ, DATETIMELTZ, DATETIMETZ | X            | X       |

| 유형  | 타입                                   | 지원 여부 (O, X) |         |
|-----|--------------------------------------|--------------|---------|
|     |                                      | PL/CSQL      | Java SP |
| 문자열 | CHAR, VARCHAR, STRING, CHAR, VARYING | O            | O       |
| 컬렉션 | SET, MULTISSET, LIST, SEQUENCE       | X            | O       |
| 기타  | BIT, BIT VARYING                     | X            | X       |
|     | ENUM                                 | X            | X       |
|     | BLOB/CLOB                            | X            | X       |
|     | JSON                                 | X            | X       |
|     | CURSOR                               | X            | O*      |

· Java SP에서는 CURSOR 타입에 대해서 반환 타입만을 지원하며, 인수 타입으로는 지원하지 않는다.

지원하지 않는 데이터 타입을 사용해서 저장 프로시저를 생성하면 다음과 같은 오류가 발생한다.

```
CREATE FUNCTION unsupported_json() RETURN JSON
AS BEGIN RETURN NULL; END;

CREATE PROCEDURE unsupported_args (arg TIMESTAMPTZ)
AS BEGIN NULL; END;
```

```
ERROR: Unsupported return type 'json' of the stored procedure
ERROR: before ')'
```



```
AS BEGIN NULL; END; '
Unsupported argument type 'timestamp_tz' of the stored procedure
```

## 인자의 모드 지정

인자의 모드에는 IN, OUT, IN OUT 모드가 있다. 인자를 지정할 때 모드를 지정하지 않으면 기본적으로 IN 모드로 간주한다.

저장 프로시저와 저장 함수에서 **CALL** 구문과 OUT 또는 IN OUT 인자를 사용하여 저장 프로시저와 저장 함수에서 변경한 인자의 값을 다시 반환할 수 있다.

```
CREATE OR REPLACE PROCEDURE test_proc (arg1 OUT INT)
AS BEGIN
 arg1 := 5;
END;

CREATE OR REPLACE FUNCTION test_func (arg1 IN OUT INT) RETURN INT
AS BEGIN
 arg1 := arg1 + 1;
 RETURN arg1 + 5;
END;
```

```
SELECT 1 into :a;
select :a;
CALL test_proc(:a);
select :a;
```

```
-- select :a;

 :a
=====
 1

-- select :a;

 :a
=====
 5
```

만약 저장 프로시저와 함수의 어떠한 인자에 OUT 또는 IN OUT 모드를 지정하는 경우, **CALL**이 아닌 질의문 (**SELECT**, **UPDATE**, **DELETE** 등) 내에서 호출할 수 없으며 에러를 반환한다.

```
SELECT test_func(1);
```

```
In the command from line 1,
ERROR: Semantic: Stored procedure/function 'u1.test_func' has OUT or
IN OUT arguments
```

## 기본값 인자 지정

저장 프로시저와 저장 함수의 인자에 기본값을 지정할 수 있다.

- `:=` 또는 **DEFAULT** 키워드를 사용하여 기본값을 지정할 수 있다.
- 기본값을 지정하면 저장 프로시저와 저장 함수의 인수를 생략하고 호출할 수 있다. 생략된 인수는 기본값으로 대체된다.
- 기본값은 리터럴 값으로 지정할 수 있으며, 255 바이트 이하의 문자열 값으로 저장한다. 이 때 크기를 초과하면 오류가 발생한다.
- 기본값에 리터럴 값 외에 허용하는 함수는 다음과 같다.

| 기본값               | 데이터 타입    |
|-------------------|-----------|
| SYS_TIMESTAMP     | TIMESTAMP |
| UNIX_TIMESTAMP()  | INTEGER   |
| CURRENT_TIMESTAMP | TIMESTAMP |
| SYS_DATETIME      | DATETIME  |
| CURRENT_DATETIME  | DATETIME  |
| SYS_DATE          | DATE      |
| CURRENT_DATE      | DATE      |

| 기본값                          | 데이터 타입 |
|------------------------------|--------|
| SYS_TIME                     | TIME   |
| CURRENT_TIME                 | TIME   |
| USER, USER()                 | STRING |
| TO_CHAR(date_time[, format]) | STRING |
| TO_CHAR(number[, format])    | STRING |

#### 참고

위의 표에 명시되지 않은 함수를 **DEFAULT** 값으로 사용하는 경우, 프로시저 생성 시점에 평가된 결과를 **DEFAULT** 값으로 저장한다. 따라서 **INSERT** 시 해당 칼럼의 기본 값은 **INSERT** 수행시 함수 수행 값이 아닌, 프로시저 생성 시점에 수행된 결과 값을 사용한다. 저장 함수인 경우 **DEFAULT** 값으로 사용할 수 없으므로 오류를 반환한다.

```
CREATE FUNCTION fn_int (a int) return int as begin return a; end;

-- error
CREATE FUNCTION default_val_func_test (id INT DEFAULT fn_int (1))
return int as begin return id; end;
```

```
ERROR: before '); '
'fn_int(1)' function can not be used in DEFAULT clause.
```

다음은 기본값에 리터럴 값을 지정하는 간단한 예시이다.

```
CREATE FUNCTION default_args (
 a INT := 1,
 b INT DEFAULT 2
) RETURN INT
AS BEGIN RETURN a + b; END;
```

```
SELECT default_args();
SELECT default_args(3);
SELECT default_args(3, 4);
```

```
default_args()
=====
3

default_args(3)
=====
5

default_args(3, 4)
=====
7
```

다음은 기본값에 함수를 지정하는 예시이다.

```
CREATE FUNCTION get_age (
 birth DATE DEFAULT DATE'2000-01-01',
 today DATE DEFAULT SYS_DATE
) RETURN INT
AS
BEGIN
 RETURN YEAR(today) - YEAR(birth)
 - CASE WHEN TO_CHAR(today, 'MMDD') < TO_CHAR(birth, 'MMDD')
THEN 1 ELSE 0 END;
END;

SELECT get_age();
SELECT get_age(DATE'2000-05-10');
SELECT get_age(DATE'2000-05-10', DATE'2025-03-24');
```

```
get_age()
=====
25

get_age(date '2000-05-10')
=====
24

get_age(date '2000-05-10', date '2025-03-24')
=====
24
```

## 함수 오버로딩 미지원

함수 오버로딩이란 동일한 이름이지만 인자의 타입이나 개수가 다른 함수를 여러 개 정의하는 것을 의미한다. CUBRID에서는 함수 오버로딩을 지원하지 않는다. 즉, 인수의 갯수나 데이터 타입이 다르더라도 동일한 이름의 저장 프로시저와 저장 함수를 생성할 수 없다. 따라서 **CREATE OR REPLACE** 구문을 사용했을 때, 새로운 프로시저가 생성되지 않고 기존에 등록한 프로시저를 덮어쓰기 때문에 주의가 필요하다.

```
CREATE FUNCTION test_func (arg1 INT) RETURN INT AS
BEGIN
 RETURN arg1;
END;

SELECT test_func (1);

CREATE OR REPLACE FUNCTION test_func (arg1 INT, arg2 INT) RETURN INT
AS
BEGIN
 RETURN arg1 + arg2;
END;

SELECT test_func (1); -- error
SELECT test_func (1, 2); -- 3
```

```
ERROR: Parameter count is invalid. expected: 2, actual: 1
```

```
test_func(1, 2)
=====
3
```

## 객체 의존성

저장 프로시저와 저장 함수가 내부적으로 참조하는 데이터베이스 객체에 대한 의존성은 PL/CSQL에서 등록 시에만 확인한다. 따라서 PL/CSQL 프로시저에서 참조하는 테이블, 뷰, 시퀀스, 인덱스 등의 객체가 존재하지 않는 경우 프로시저 등록 시 오류가 발생한다.

```
DROP TABLE IF EXISTS test_tbl;

CREATE OR REPLACE FUNCTION test_func () RETURN INT
k INT;
AS
BEGIN
 SELECT id INTO k FROM test_tbl LIMIT 1;
 RETURN k;
END;
```

```
ERROR: Stored procedure compile error: Syntax: before ' LIMIT 1'
Unknown class "dba.test_tbl". select id from [dba.test_tbl] limit 1
```

다음 예제에서 test\_tbl 테이블을 생성하는 경우 올바르게 생성된다.

```
DROP TABLE IF EXISTS test_tbl;
CREATE TABLE test_tbl (id INT);
INSERT INTO test_tbl VALUES (1);

CREATE OR REPLACE FUNCTION test_func () RETURN INT
AS
 k INT;
BEGIN
 SELECT id INTO k FROM test_tbl LIMIT 1;
 RETURN k;
END;

SELECT test_func();
```

```
test_func()
=====
 1
```

단, PL/CSQL의 Dynamic SQL 이나 Java SP의 경우 저장된 문자열을 런타임에 실행하기 때문에 프로시저 등록 시 확인하지 않고, 프로시저 호출 시 확인된다.

```
DROP TABLE IF EXISTS test_tbl;

CREATE OR REPLACE FUNCTION test_func () RETURN INT
AS
 k INT;
BEGIN
 execute immediate 'SELECT id FROM test_tbl LIMIT 1' INTO k;
 RETURN k;
END;

SELECT test_func();
```

```
ERROR: Stored procedure execute error:
(line 5, column 9) Syntax: before ' LIMIT 1'
Unknown class "dba.test_tbl". select id from [dba.test_tbl] limit 1
```

저장 프로시저와 저장 함수가 내부적으로 참조하는 객체가 삭제되는 경우에는 시스템에서 의존성을 확인하지 않으므로 올바르게 등록되더라도 프로시저 호출 시 오류가 발생할 수 있다. 따라서 저장 프로시저와 저장 함수가 참조하는 객체가 삭제되지 않도록 주의해야 한다.

```
DROP TABLE IF EXISTS test_tbl;
CREATE TABLE test_tbl (id INT);
INSERT INTO test_tbl VALUES (1);

CREATE OR REPLACE FUNCTION test_func () RETURN INT
AS
 k INT;
BEGIN
 SELECT id INTO k FROM test_tbl LIMIT 1;
 RETURN k;
END;

DROP TABLE IF EXISTS test_tbl;
SELECT test_func();
```

```
ERROR: Stored procedure execute error:
(line 5, column 9) Syntax: Unknown class "dba.test_tbl". select
[dba.test_tbl].id from [dba.test_tbl] [dba.test_tbl] ...
```

## 문자열 코드셋 인코딩 고려사항

CUBRID의 모든 문자열 데이터는 코드셋을 가지고 있다. CUBRID PL 엔진에서는 저장 프로시저와 저장 함수의 내부에서 문자열 데이터를 단일 코드셋으로만 처리한다. 그리고 문자열 데이터를 입출력할 때는 코드셋 변환을 수행하지 않는다. 다음의 규칙에 따라 문자열 데이터를 처리한다.

- 문자열 타입의 인수값은 데이터베이스의 코드셋과 동일 코드셋을 사용하는 것으로 간주한다.
- 문자열 타입의 결과값은 데이터베이스의 코드셋과 동일 코드셋을 사용하는 것으로 간주한다.
- 문자열 리터럴 값은 데이터베이스의 코드셋과 동일한 코드셋을 사용하는 것으로 간주한다.
- SQL 질의문의 결과가 문자열 타입인 경우, 해당 컬럼의 코드셋에서 데이터베이스의 코드셋으로 변환된 문자열이 전달된다.

### 참고

- 데이터베이스의 코드셋은 **cubrid createdb** 유틸리티에서 **charset** 옵션을 사용하여 설정되며, DB 생성 이후에는 변경할 수 없다.

- 저장 프로시저/함수에서 문자열을 입력하거나 출력하는 모든 작업은 데이터베이스 코드셋과 일치하는 코드셋을 사용해야만 올바르게 동작한다.

### 예제 1

다음의 예시는 문자열 타입의 인수가 '한글' 리터럴 값과 연산하여 결과를 반환하는 예시이다. 문자열 타입의 인수와 결과 그리고 리터럴 값의 코드셋은 데이터베이스의 코드셋과 동일하다고 가정한다. 따라서 다음의 SQL 문을 올바르게 실행하기 위해서는 UTF-8 인코딩으로 저장하고 수행해야 한다.

```
CREATE OR REPLACE FUNCTION test_func (arg VARCHAR) RETURN VARCHAR
AS
BEGIN
 RETURN arg || ',' || '한글';
END;

SELECT test_func('큐브리드');
```

```
test_func('큐브리드')
=====
큐브리드,한글
```

### 예제 2

SQL 질의문의 결과에 사용되는 코드셋은 기본적으로 데이터베이스의 코드셋을 따르지만, **컬럼의 문자셋과 콜레이션**의 설명한 것처럼 테이블의 각 컬럼의 코드셋을 개별적으로 지정할 수 있다. 이 경우, 다음과 같은 상황이 발생할 수 있다.

- 컬럼 코드셋과 데이터베이스 코드셋이 동일: 변환이 필요하지 않는다.
- 컬럼 코드셋과 데이터베이스 코드셋이 다름: CUBRID는 내부적으로 데이터베이스 코드셋으로 변환한다. 변환 과정에서 잘못된 문자 변환이나 데이터 손실이 발생할 수 있으니 주의해야 한다.

다음의 예시는 UTF-8로 설정된 데이터베이스에서 테이블 컬럼의 코드셋을 EUC-KR로 지정된 경우에, 질의문 결과로 반환되는 문자열은 EUC-KR에서 UTF-8로 내부적으로 변환되어 문자열 리터럴과 연산한 후 반환하는 것을 보여준다.

```
-- 이 파일은 euc-kr로 저장되어야 하며, '큐브리드'도 euc-kr 인코딩으로 입력되어야 한다.
CREATE TABLE tblcharcol (col STRING CHARSET euckr) COLLATE euckr_bin;
INSERT INTO tblcharcol VALUES ('큐브리드');
```



```
-- 이 파일은 utf-8로 저장되어야 하며, '한글'도 utf-8 인코딩으로 입력되어야 한다.
CREATE OR REPLACE FUNCTION test_func () RETURN VARCHAR
AS
 v VARCHAR;
BEGIN
 SELECT col INTO v FROM tblcharcol LIMIT 1;
 RETURN v || '한글';
END;

SELECT test_func();
```

```
test_func()
=====
큐브리드한글
```

## 저장 프로시저의 호출

등록된 저장 프로시저와 저장 함수는 **CALL** 구문을 사용하거나 SQL 문에서 호출할 수 있다. 일반적인 경우 사용자의 용도에 따라 저장 프로시저는 **CALL** 구문을 사용하여 호출하고, 저장 함수는 그 외 반환 값이 필요한 SQL 질의문에서 사용한다.

이 장에서는 저장 프로시저와 저장 함수를 호출과 관련하여 고려해야 할 사항에 대하여 자세히 설명한다.

## CALL 문

**CALL** 구문을 사용하여 다음과 같이 CSQL 인터프리터에서 저장 프로시저를 호출할 수 있다.

```
-- csql에서 실행
CREATE PROCEDURE hello ()
AS
BEGIN
 DBMS_OUTPUT.put_line('Hello, CUBRID!');
END;

;server-output on

-- 호출
CALL hello();
```

```

Result
=====
NULL

<DBMS_OUTPUT>
====
Hello, CUBRID!

```

## SQL 문에서 호출

SQL 문에서 호출하는 경우 일반적으로 저장 함수를 사용하여 값을 반환한다. 다음은 저장 함수 사용의 예시로 특정 국가의 메달 수를 계산하는 예시를 보여준다.

```

CREATE FUNCTION count_medals(nation STRING) RETURN INT
AS
cnt INT;
BEGIN
 SELECT COUNT(*) INTO cnt
 FROM game
 WHERE nation_code = nation;
 RETURN cnt;
END;

SELECT count_medals('USA');
SELECT count_medals('KOR');

```

```

count_medals('USA')
=====
1118

count_medals('KOR')
=====
316

```

## 프로시저 호출 시 중첩 호출 제한사항

저장 프로시저와 저장 루틴의 허용된 최대 중첩 호출 수는 **16**으로 제한된다. 중첩 호출이란 프로시저 내부에서 다른 프로시저를 호출하는 것을 의미한다. 이 때 질의문을 통한 호출이 아닌 재귀 호출의 경우에는 제한하지 않는다.

다음은 중첩 호출 제한 사항에 대한 예시이다.

```

CREATE OR REPLACE FUNCTION test_factorial_query(n BIGINT) RETURN
BIGINT
AS
k BIGINT;
BEGIN
 IF n = 0 THEN
 RETURN 1;
 ELSE
 SELECT test_factorial_query(n - 1) INTO k;
 RETURN n * k;
 END IF;
END;

SELECT test_factorial_query(15);
SELECT test_factorial_query(16);

```

```

test_factorial_query(15)
=====
1307674368000

-- SELECT test_factorial_query(16);
ERROR: Stored procedure execute error:
...
(line 8, column 13) Stored procedure execute error:
(line 8, column 13) Stored procedure execute error: Too many
nested stored procedure call.

```

다음은 질의문을 통하지 않고 재귀 호출을 하는 예시이다. 이 경우 중첩 호출 제한 사항이 적용되지 않는다. 재귀 호출을 통하여 다음과 같이 팩토리얼의 값이 BIGINT 값을 넘어 오버플로우가 발생하거나, 시스템 자원을 과도하게 사용하는 경우 질의가 실패할 수 있으므로 주의한다.

```

CREATE OR REPLACE FUNCTION test_factorial_constant(n BIGINT) RETURN
BIGINT
AS
k BIGINT;
BEGIN
 IF n = 0 THEN
 RETURN 1;
 ELSE
 RETURN n * test_factorial_constant(n - 1);
 END IF;
END;

SELECT test_factorial_constant(16);
SELECT test_factorial_constant(25);

```

```
test_factorial_constant(16)
=====
20922789888000

-- SELECT test_factorial_constant(25);
ERROR: Stored procedure execute error:
(line 8, column 20) data overflow in multiplication of BIGINT
values
```

## 저장 프로시저의 권한

이 장에서는 저장 프로시저를 호출하기 위해 필요한 권한과 다른 사용자에게 프로시저에 대한 호출 권한을 부여하는 방법에 대하여 자세히 설명한다. 또한 호출 시 어떤 사용자의 권한으로 호출하는지 명시하는 소유자 권한과 호출자 권한의 차이에 대하여 자세히 설명한다.

### 프로시저 호출 권한 부여

프로시저를 생성하면 프로시저를 생성한 사용자의 스키마에 속하며, 오직 **소유자**와 **DBA** 권한을 가진 사용자만이 프로시저를 호출할 수 있다. 프로시저의 소유자가 다른 사용자에게 프로시저에 대한 호출 권한을 부여하면, 해당 사용자는 권한을 부여받은 다른 사용자의 프로시저를 호출할 수 있다.

이 때 주의해야 할 점은 구현하는 일련의 루틴에서 권한이 없는 객체를 참조하는 경우에는 컴파일 또는 실행 시 오류가 발생할 수 있다. 따라서 저장 프로시저에서 참조하는 객체에 대해서도 권한이 있는지 확인하는 것이 필요하다.

CUBRID에서는 **GRANT** 문을 사용하여 프로시저에 대한 호출 권한을 부여할 수 있다.

```
GRANT EXECUTE ON PROCEDURE procedure_name TO user_name;
```

반대로 **REVOKE** 문을 사용하여 프로시저에 대한 호출 권한을 해지할 수 있다.

```
REVOKE EXECUTE ON PROCEDURE procedure_name FROM user_name;
```

#### 참고

저장 프로시저와 저장 함수에 대한 **EXECUTE ON PROCEDURE** 권한은 **WITH GRANT OPTION** 을 지원하지 않는다. 따라서 프로시저의 소유자가 부여한 권한을 다른 사용자에게 부여할 수 없다.

## 소유자 권한과 호출자 권한의 차이

프로시저의 호출 권한을 가지면, 해당 프로시저를 실행할 때 어떠한 사용자의 권한으로 실행될지 명시할 수 있다. 저장 프로시저와 저장 함수는 **소유자** 권한 또는 **호출자** 권한으로 실행될 수 있다. 실행 권한은 저장 프로시저 또는 저장함수 생성시 둘 중 하나를 선택하여 생성할 수 있으며, 지정한 권한으로 실행된다.

다음은 각 권한에 대한 설명과 차이점을 설명한다.

### · 소유자 권한 (Owner's Rights 또는 Definer's Rights)

- 저장 프로시저를 만든 사용자의 권한으로 실행된다.
- 저장 프로시저를 만든 사용자가 다른 사용자에게 권한을 부여하면 부여된 사용자도 저장 프로시저를 실행할 수 있다.
- 저장 프로시저의 소유자의 모든 권한을 사용 가능하므로 별도의 권한 부여 없이 소유자가 접근할 수 있는 데이터베이스 객체에 접근할 수 있다. **DBA**의 소유자 권한으로 저장 프로시저를 만들면 모든 데이터베이스 객체에 접근할 수 있으므로 특히 동적 SQL(Dynamic SQL)을 사용할 때 SQL 인젝션(SQL Injection)과 같은 보안 문제에 주의해야 한다.
- 공통적인 작업이나 데이터 접근을 필요로 하는데 적합하다.

반면에 호출자 권한은 다음과 같다.

### · 호출자 권한 (Caller's Rights 또는 Invoker's Rights)

- 저장 프로시저를 만든 사용자의 권한이 아닌 호출자의 권한으로 실행된다.
- 호출자 권한으로 저장 프로시저를 만들고 다른 유저에게 권한을 부여하면, 저장 프로시저를 호출하는 사용자의 권한으로 실행된다.
- 호출자 권한으로 저장 프로시저를 만들면 저장 프로시저를 만든 사용자의 권한이 아닌 호출자의 권한 수준에서 실행되므로 호출자의 권한으로 데이터베이스 객체에 접근할 수 있다. 따라서 호출자 권한으로 저장 프로시저를 만들 때는 호출자의 권한 수준을 고려하여 저장 프로시저를 만들어야 한다.
- 호출자별로 권한을 다르게 적용해야할 경우에 적합하다.

저장 프로시저 생성 시 소유자 권한과 호출자 권한의 차이를 이해하고 저장 프로시저를 활용하면 보다 효율적으로 데이터베이스를 관리할 수 있다.

저장 프로시저 또는 저장 함수 생성 시 **AUTHID** 속성을 지정하여 **소유자 권한** 또는 **호출자 권한** 으로 의 동작 여부를 지정할 수 있다. 정의문에 대한 자세한 내용은 **저장 프로시저/함수 정의문**를 참고한다.

**AUTHID** 속성을 다음과 같이 지정할 수 있으며, **DEFINER**와 **OWNER** 그리고 **CURRENT\_USER**와 **CALLER**는 동의어이다.

- **소유자 권한**: AUTHID DEFINER 또는 AUTHID OWNER
- **호출자 권한**: AUTHID CURRENT\_USER 또는 AUTHID CALLER

속성을 지정하지 않으면 기본적으로 **소유자 권한**으로 동작한다.

다음은 DBA 사용자로 로그인하여 **CURRENT\_USER**, **USER** 를 반환하는 DBA 소유자 권한의 저장 함수를 만들고 U1 사용자에서 호출하는 예이다.

```
-- DBA로 로그인
CREATE USER U1;

CREATE OR REPLACE FUNCTION fn_current_user() RETURN STRING AUTHID
DEFINER AS
BEGIN
 RETURN CURRENT_USER;
END;

GRANT EXECUTE ON PROCEDURE fn_current_user TO U1;

CALL login ('U1', '') ON CLASS db_user;

SELECT dba.fn_current_user();
```

-- 위와 같이 U1 사용자로 호출했지만, 결과는 'DBA@' 이므로 소유자 권한으로 실행되었음을 알 수 있다.

```
dba.fn_current_user()
=====
'DBA@<host>'
```

#### ⚠ 경고

- CUBRID의 PL/CSQL에서는 현재 **소유자 권한** 만을 지원하고 있으며, **호출자 권한**은 지원하지 않는다.
- **호출자 권한**을 사용하기 위해서는 Java SP를 사용하여 저장 프로시저를 작성해야 한다.

## 참고

- 소유자 권한은 CUBRID 11.4 버전부터 지원한다.

## 트랜잭션 커밋과 롤백 지원

저장 프로시저와 저장 함수에서는 트랜잭션을 **COMMIT**하거나 **ROLLBACK**하여 작업을 완료하거나 취소할 수 있다. 이는 데이터 무결성을 유지하고, 트랜잭션 처리의 유연성을 제공하는 중요한 기능이다. 저장 프로시저와 함수 내부의 로직에서 조건이나 예외 처리에 따라 트랜잭션을 제어할 수 있어 복잡한 데이터 로직을 안전하고 효율적으로 관리하는데 큰 도움을 줄 수 있다.

## 조건 및 예외 처리와 트랜잭션 제어의 장점

1. **조건 기반 트랜잭션 처리** 저장 프로시저와 함수 내부에서 조건에 따라 트랜잭션을 커밋하거나 롤백할 수 있다. 예를 들어, 특정 조건이 만족되지 않을 경우 롤백하여 데이터의 무결성을 보장할 수 있다.
2. **예외 발생 시 안전한 롤백** 저장 프로시저 내부에서 오류(예외)가 발생하면 해당 트랜잭션을 롤백하고, 작업을 안전한 상태로 되돌릴 수 있다. 이는 예기치 않은 상황에서도 데이터가 손상되지 않도록 하는 데 유용하다.
3. **복잡한 로직의 처리 안정성** 여러 작업이 포함된 복잡한 로직을 처리하면서도, 상황에 따라 트랜잭션의 성공 여부를 명확히 제어할 수 있다. 이를 통해 데이터 처리의 신뢰성을 높일 수 있다.

## 예제: 조건 및 예외 처리와 트랜잭션 제어

다음은 저장 프로시저에서 조건과 예외 처리를 활용하여 트랜잭션을 제어하는 예제이다

```
;server-output on

CREATE TABLE orders (
 id INT NOT NULL PRIMARY KEY,
 product VARCHAR(100) NOT NULL,
 quantity INT NOT NULL
);

CREATE TABLE inventory (
 product VARCHAR(100) NOT NULL PRIMARY KEY,
 stock INT NOT NULL
);
```

```

CREATE OR REPLACE PROCEDURE process_transaction()
AS
current_stock INT;
BEGIN
 INSERT INTO orders (id, product, quantity) VALUES (1,
'Laptop', 5);

 UPDATE inventory SET stock = stock - 5 WHERE product =
'Laptop';

 SELECT stock INTO current_stock FROM inventory WHERE product
= 'Laptop';

 IF current_stock < 0 THEN
 ROLLBACK;
 DBMS_OUTPUT.PUT_LINE('Transaction failed: Insufficient
stock. ');
 ELSE
 COMMIT;
 DBMS_OUTPUT.PUT_LINE('Transaction successful. ');
 END IF;
EXCEPTION
 WHEN OTHERS THEN
 ROLLBACK;
 DBMS_OUTPUT.PUT_LINE('Transaction failed: An error
occurred. ');
END;

```

```

process_transaction ();

=== <Result of CALL Command in Line 2> ===

Result
=====
NULL

<DBMS_OUTPUT>
====
Transaction successful.

```

이 예제는 다음과 같은 시나리오를 다룬다

- \* 재고가 부족한 경우 트랜잭션을 롤백하여 데이터의 무결성을 유지한다.
- \* 작업 중 예외가 발생하면 트랜잭션을 안전하게 롤백한다.
- \* 조건이 만족되면 트랜잭션을 커밋하여 작업을 완료한다.



## 주의사항

- PL/CSQL에서는 기본적으로 **COMMIT**과 **ROLLBACK** 문을 지원하며, 이를 통해 저장 프로시저 내부에서 트랜잭션 처리를 명시적으로 제어할 수 있다
- 그러나 자바 저장 프로시저의 경우, 하위호환성을 유지하기 위해 **pl\_transaction\_control** 설정을 활용해야 하며 **yes**로 설정하는 경우에만 COMMIT과 ROLLBACK 문을 사용할 수 있다.
- **pl\_transaction\_control**의 기본값은 **no**로 설정되어 있으며 COMMIT과 ROLLBACK이 무시된다.

## 성능 최적화

저장 프로시저와 저장 함수는 사용자가 직접 작성한 코드이므로, 성능 최적화가 필요할 수 있다. 이 장에서는 저장 프로시저와 저장 함수의 성능을 최적화하기 위한 다양한 가이드를 제공한다.

목차

### 성능 최적화

7

- SQL 질의 최적화 864
- 질의에서의 저장 함수 호출 최적화 10
- 내장 함수 사용 1430
- 결정적 함수 사용 1431

### SQL 질의 최적화

저장 프로시저와 저장 함수의 가장 흔한 성능 오버헤드는 비효율적인 SQL 구문 수행에서 발생할 수 있다. 내부 루틴에서 사용되는 SQL 문에 대해서 충분히 튜닝하여 다음의 비효율적인 문제를 방지해야 한다.

- **테이블 풀 스캔**: 인덱스를 사용하지 않고 전체 테이블을 스캔하는 경우 성능이 저하될 수 있다.
- **통신 오버헤드**: CUBRID 데이터베이스 서버와 CUBRID PL 서버 간의 통신 오버헤드를 줄이기 위해서는 최소한의 개수의 레코드를 반환하도록 쿼리를 작성해야 한다.
- **비효율적인 반복문 내의 질의 호출**: 저장 프로시저와 저장 함수 내부에서 반복문을 사용할 때, 반복문 내에서 다수의 질의를 호출하지 않고 한번의 질의로 데이터를 가져오는 것을 권장한다.

저장 프로시저와 저장함수에서 사용된 비효율적인 질의는 **조인 방식** 문서를 참조하여 질의 최적화하기를 권장한다.

## 질의에서의 저장 함수 호출 최적화

질의에서 실행하는 저장 함수의 불필요한 반복 호출은 성능을 저하시킬 수 있다. 따라서 저장 함수 호출을 최적화하기 위해 다음과 같은 방법을 고려해야 한다.

- **불필요한 반복 호출 최소화:** 저장 함수의 호출 횟수를 줄이는 것이 성능을 향상시키는 가장 좋은 방법이다.
  - 인덱스를 활용하여 저장 함수를 호출하는 레코드의 수를 줄인다.
  - 동일한 인수에 대해 반복 호출되지 않도록, 중복되는 데이터를 그룹화 하여 한번의 호출로 처리할 수 있도록 한다.
  - 만약 함수의 로직이 결정적이라면 **결정적 함수 사용**을 활용한 상관 부질의 결과 캐시를 사용하여 성능을 향상시킬 수 있다.
- **함수의 인수와 반환 값의 크기를 최소화:** 저장 함수의 인수와 반환 값을 필요한 값만 반환하도록 설계하여 불필요하게 큰 데이터를 반환하지 않게 한다.

## 내장 함수 사용

CUBRID에서 기본적으로 제공하는 내장 함수는 (**연산자와 함수**) CUBRID의 쿼리 실행 동작에 맞춰 최적화되어 효율적으로 동작한다. 반면, 사용자가 작성하는 저장 함수는 그 내부 루틴을 모르는 블랙박스 이므로 내장 함수에 비해 성능이 떨어질 수 있다. 따라서 내장 함수의 단순한 조합으로 구현이 가능한 저장 함수보다 내장 함수를 사용하는 것을 것을 권장한다.

아래는 CONCAT 내장 함수와 동일한 기능을 수행하는 저장 함수의 예시이다. 사용자 저장 함수인 my\_concat()보다 내장 함수인 concat()사용 시 성능 이점이 있다.

```
CREATE OR REPLACE FUNCTION my_concat (a STRING, b STRING) RETURN
STRING AS
BEGIN
 RETURN a || b;
END;

SELECT COUNT(*) FROM (SELECT /*+ NO_MERGE */ concat (name, event)
FROM athlete);

SELECT COUNT(*) FROM (SELECT /*+ NO_MERGE */ my_concat (name, event)
FROM athlete);
```

```
-- concat 함수 사용
count(*)
```

```
=====
6677

1 row selected. (0.019853 sec) Committed. (0.000000 sec)

-- my_concat 함수 사용
count(*)
=====
6677

1 row selected. (0.302333 sec) Committed. (0.000000 sec)
```

## 결정적 함수 사용

결정적 함수는 동일한 인수에 대해 항상 동일한 결과를 반환하는 함수를 의미한다. 저장 함수가 결정적 함수이면, 저장 함수의 결과를 재사용하여 성능을 향상시킬 수 있다.

- 저장 함수를 결정적 함수로 만들기 위해 생성 시 **CREATE FUNCTION** 구문에서 **DETERMINISTIC** 속성을 지정할 수 있다. 자세한 내용은 [CREATE FUNCTION](#)를 참고한다.
- **DETERMINISTIC** 속성을 지정하면 저장 함수는 상관 부질의 결과 캐시의 최적화에 사용될 수 있으며 동일한 인수에 대해 항상 동일한 결과를 반환한다. 상관 부질의 캐시 동작 방식에 대한 자세한 내용은 [서브 쿼리 캐시 \(correlated\)](#)을 참고한다.

다음은 **DETERMINISTIC**으로 결정적 함수로 생성한 저장 함수의 예시이다. 이 예시에서는 상관 부질의 사용할 때 결과를 캐시하여 성능을 최적화하는 과정을 보여준다.

```
CREATE TABLE dummy_tbl (col1 INTEGER);
INSERT INTO dummy_tbl VALUES (1), (2), (1), (2);

CREATE OR REPLACE FUNCTION pl_csql_not_deterministic (n INTEGER)
RETURN INTEGER AS
BEGIN
 return n + 1;
END;

CREATE OR REPLACE FUNCTION pl_csql_deterministic (n INTEGER) RETURN
INTEGER DETERMINISTIC AS
BEGIN
 return n + 1;
END;

SELECT sp_name, owner, sp_type, is_deterministic from
db_stored_procedure;
```

```

 sp_name owner
sp_type is_deterministic
=====
=====
 'pl_csqli_not_deterministic' 'DBA'
'FUNCTION' 'NO'
 'pl_csqli_deterministic' 'DBA'
'FUNCTION' 'YES'

```

위 예시에서 pl\_csqli\_not\_deterministic 함수는 **NOT DETERMINISTIC**이므로 상관 부질의에서 쿼리 캐시를 사용하지 않는다. 반면, pl\_csqli\_deterministic 함수는 **DETERMINISTIC** 키워드가 지정되어 있으므로 상관 부질의에서 쿼리 캐시를 사용하여 성능을 최적화할 수 있다.

```

;trace on
SELECT (SELECT pl_csqli_not_deterministic (t1.col1) FROM dual) AS
results FROM dummy_tbl t1;

```

```

 results
=====
 2
 3
 2
 3

=== Auto Trace ===
...
Trace Statistics:
 SELECT (time: 4, fetch: 11, fetch_time: 0, ioread: 0)
 FUNC (time: 4, fetch: 2, ioread: 0, calls: 4)
 SCAN (table: dba.dummy_tbl), (heap time: 0, fetch: 5, ioread:
0, readrows: 4, rows: 4)
 SUBQUERY (correlated)
 SELECT (time: 4, fetch: 6, fetch_time: 0, ioread: 0)
 FUNC (time: 4, fetch: 2, ioread: 0, calls: 4)
 SCAN (table: dual), (heap time: 0, fetch: 4, ioread: 0,
readrows: 4, rows: 4)

```

pl\_csqli\_not\_deterministic 함수는 **NOT DETERMINISTIC** 이므로 상관 부질의 결과를 캐시하지 않는다.

```

;trace on
SELECT (SELECT pl_csqli_deterministic (t1.col1) FROM dual) AS results
FROM dummy_tbl t1;

```

```

results
=====
 2
 3
 2
 3

=== Auto Trace ===
...
Trace Statistics:
 SELECT (time: 2, fetch: 9, fetch_time: 0, ioread: 0)
 FUNC (time: 2, fetch: 2, ioread: 0, calls: 2)
 SCAN (table: dba.dummy_tbl), (heap time: 0, fetch: 5, ioread:
0, readrows: 4, rows: 4)
 SUBQUERY (correlated)
 SELECT (time: 2, fetch: 4, fetch_time: 0, ioread: 0)
 FUNC (time: 2, fetch: 2, ioread: 0, calls: 2)
 SCAN (table: dual), (heap time: 0, fetch: 2, ioread: 0,
readrows: 2, rows: 2)
 SUBQUERY_CACHE (hit: 2, miss: 2, size: 150808, status:
enabled)

```

pl\_csql\_deterministic 함수의 Trace 결과에서는 **SUBQUERY\_CACHE** 항목이 표시되며(hit: 2, miss: 2, size: 150808, status: enabled), 상단의 **SCAN (table: dual)** 에서 읽은 레코드 수(**readrows**)가 **NOT DETERMINISTIC** 예시와 비교해 감소한 것을 확인할 수 있다.

### ⚠ 경고

- 저장 프로시저에서는 **DETERMINISTIC** 속성을 지원하지 않는다.
- 결정적이지 않은 결과를 반환하는 내부 구현에서 **DETERMINISTIC** 속성을 사용하는 경우에는 기대한 결과를 반환하지 않을 수 있다.

```

CREATE TABLE test_table (
 id INT AUTO_INCREMENT PRIMARY KEY,
 name VARCHAR(100)
);

INSERT INTO test_table (name) VALUES
('Alice'),
('Bob'),
('Charlie'),
('Alice'),
('Bob');

CREATE SERIAL my_serial;

```

```
CREATE OR REPLACE FUNCTION cnt_name (name VARCHAR) RETURN VARCHAR
DETERMINISTIC AS BEGIN RETURN name || my_serial.NEXT_VALUE; END;
```

```
SELECT
id,
name,
(SELECT cnt_name(name) FROM DUAL) AS result
FROM test_table;
```

| id | name      | result     |
|----|-----------|------------|
| 1  | 'Alice'   | 'Alice1'   |
| 2  | 'Bob'     | 'Bob2'     |
| 3  | 'Charlie' | 'Charlie3' |
| 4  | 'Alice'   | 'Alice1'   |
| 5  | 'Bob'     | 'Bob2'     |

위 예시에서 cnt\_name 함수 내부의 my\_serial.NEXT\_VALUE는 결정적이지 않은 결과를 반환하므로 상관 부질의 캐시에 의해 기대하지 않은 결과를 반환한다. 저장 함수의 구현을 고려하여 **DETERMINISTIC** 속성을 지정 할 것을 권장한다.

## 시스템 패키지

CUBRID에서는 사용자 편의를 위해 시스템 패키지를 제공한다. 향후 버전에서 더 많은 패키지가 추가되어 CUBRID의 기능이 확장될 예정이며 현재 버전에서는 DBMS\_OUTPUT 패키지만을 제공하고 있다.

### DBMS\_OUTPUT

DBMS\_OUTPUT 패키지는 문자열 메시지를 DBMS\_OUTPUT 버퍼에 저장하고 읽어오기 위한 기능을 제공한다. 저장 프로시저/함수 개발자는 이 패키지의 PUT\_LINE이나 PUT 함수로 원하는 메시지를 DBMS\_OUTPUT 버퍼에 쌓을 수 있다. CSQ나 DBeaver 같은 클라이언트 도구들은 이 패키지의 ENABLE, DISABLE, GET\_LINE, GET\_LINES 함수들을 사용하여 메시지 저장 기능을 활성화/비활성화하고 버퍼에 쌓인 메시지들을 가져온다. 개발자는 프로그램 진행 상황 확인이나 디버깅을 위해 이 메시지들을 유용하게 사용할 수 있다.

이 섹션에서는 DBMS\_OUTPUT 패키지의 사용법과 활용 예시를 설명한다. DBMS\_OUTPUT 패키지의 함수는 다음과 같다:

- DBMS\_OUTPUT.ENABLE
- DBMS\_OUTPUT.DISABLE

- DBMS\_OUTPUT.PUT
- DBMS\_OUTPUT.PUT\_LINE
- DBMS\_OUTPUT.NEW\_LINE
- DBMS\_OUTPUT.GET\_LINE
- DBMS\_OUTPUT.GET\_LINES

## DBMS\_OUTPUT.ENABLE

### DBMS\_OUTPUT.ENABLE(*size*)

DBMS\_OUTPUT 메시지 버퍼를 활성화하고, 메시지를 저장할 버퍼의 크기를 설정한다.

매개변수:

**size** – 버퍼의 크기를 지정하며, 이 값은 바이트 단위로 지정한다. 최대 크기는 32767 바이트이며 이 값을 초과하면 오류가 발생한다.

#### 참고

CSQL 인터프리터에서 **server-output on** 을 호출하면 내부적으로 기본값인 DBMS\_OUTPUT.ENABLE(20000)을 호출한 것과 동일하다. 자세한 내용은 [CSQL 세션명령어 server-output](#)을 참고한다.

## DBMS\_OUTPUT.DISABLE

### DBMS\_OUTPUT.DISABLE()

현재 버퍼에 저장된 메시지를 제거하고 버퍼를 비활성화한다. 따라서 DBMS\_OUTPUT 패키지 내의 다른 프로시저를 호출하더라도 아무런 출력이 나타나지 않는다.

#### 참고

CSQL 인터프리터에서 **server-output off** 을 호출하면 내부적으로 DBMS\_OUTPUT.DISABLE()을 호출한 것과 같다. 자세한 내용은 [CSQL 세션명령어 server-output](#)을 참고한다.

## DBMS\_OUTPUT.PUT

### DBMS\_OUTPUT.PUT(*str* VARCHAR)

지정된 문자열을 줄바꿈 없이 버퍼에 저장한다.

매개변수:

**str** – 저장할 문자열을 지정한다. 저장할 문자열이 NULL이면 아무런 동작도 하지 않는다.

## DBMS\_OUTPUT.PUT\_LINE

### DBMS\_OUTPUT.PUT\_LINE(*line* VARCHAR)

지정된 문자열을 버퍼에 저장하고 줄바꿈을 추가한다.

매개변수:

**line** – 저장할 문자열을 지정한다. 저장할 문자열이 NULL이면 아무런 동작도 하지 않는다.

## DBMS\_OUTPUT.NEW\_LINE

### DBMS\_OUTPUT.NEW\_LINE()

버퍼에 줄바꿈 문자를 추가한다. PUT 함수로 문자열을 추가한 후 NEW\_LINE 함수를 호출하여 GET\_LINE에서 줄 단위로 읽어올 수 있다.

## DBMS\_OUTPUT.GET\_LINE

### DBMS\_OUTPUT.GET\_LINE(*line* OUT VARCHAR, *status* OUT INTEGER)

버퍼에 저장된 문자열 메시지를 중 첫 번째 줄을 읽어온다. 읽어온 줄은 버퍼에서 삭제된다.



매개변수:

- **line** – 버퍼로 부터 읽어온 문자열을 저장한다.
- **status** – 문자열을 성공적으로 읽어왔을 경우 0을, 그렇지 않을 경우 1을 저장한다.

## DBMS\_OUTPUT.GET\_LINES

DBMS\_OUTPUT.GET\_LINES(*lines OUT VARCHAR, num\_lines IN OUT INTEGER*)

버퍼에 저장된 문자열 메시지를 지정된 줄 수만큼 읽어온다. 읽어온 줄은 버퍼에서 삭제된다.

매개변수:

- **lines** – 버퍼로 부터 읽어온 문자열을 저장한다.
- **num\_lines** – 읽어올 줄의 수를 지정한다.

## 활용 예시

다음은 CSQL 인터프리터로 DBMS\_OUTPUT 패키지를 사용한 단순한 예시이다. PUT\_LINE 함수는 저장 함수 개발자가 사용하고 ENABLE, DISABLE, GET\_LINE 함수들은 CSQL 인터프리터가 기능 구현을 위해 내부적으로 사용한다.

```
;server-output on -- CSQL이 내부적으로 DBMS_OUTPUT.ENABLE 호출

CREATE OR REPLACE FUNCTION test() RETURN VARCHAR
AS
BEGIN
 DBMS_OUTPUT.PUT_LINE('Hello, World!');
 DBMS_OUTPUT.PUT_LINE('Hello, CUBRID!');
 DBMS_OUTPUT.PUT_LINE('Hello, DBMS_OUTPUT!');
 RETURN 'Success';
END;

SELECT test();

;server-output off -- CSQL이 내부적으로 DBMS_OUTPUT.DISABLE 호출
```

```
test ()
=====
'Success'
```

```

<DBMS_OUTPUT> <-- CSQL이 출력할 메시지를 가져오기 위해 내부적으로
DBMS_OUTPUT.GET_LINE 여러 번 호출
====
Hello, World!
Hello, CUBRID!
Hello, DBMS_OUTPUT!

```

## PL/CSQL

CUBRID는 PL/CSQL(Procedural Language extension of CUBRID SQL)로 저장 함수와 프로시저를 개발할 수 있도록 지원한다. PL/CSQL은 CUBRID SQL의 절차적 언어로의 확장으로서, 선언적 언어인 SQL만으로 구현하기 어려운 조건문, 반복문, 변수, 예러 처리, 내부 프로시저/함수를 통한 모듈화 같은 절차적 언어의 특성을 지원한다. 물론, SQL 문을 PL/CSQL 프로그램 안에 포함시켜 함께 사용할 수 있다.

## 개요

### 저장 프로시저/함수 생성

PL/CSQL은 저장 프로시저나 저장 함수를 생성하는데 사용된다. 다음 문법을 따르는 CREATE PROCEDURE 문과 CREATE FUNCTION 문의 AS (또는 IS) 키워드 뒤에 PL/CSQL 코드를 써서 생성하고자 하는 저장 프로시저/함수의 동작을 기술한다.

```

<create_procedure> ::=
 CREATE [OR REPLACE] PROCEDURE [schema_name.]<identifier> [(
<seq_of_parameters>)]
 [<procedure_properties>] { IS | AS } [LANGUAGE PLCSQL] [
<seq_of_declare_specs>] <body> ;
<create_function> ::=
 CREATE [OR REPLACE] FUNCTION [schema_name.]<identifier> [(
<seq_of_parameters>)] RETURN <type_spec>
 [<function_properties>] { IS | AS } [LANGUAGE PLCSQL] [
<seq_of_declare_specs>] <body> ;

```

위 문법에서 저장 프로시저/함수의 *body*는 PL/CSQL 실행문들을 포함하고 그 앞의 선언부 *seq\_of\_declare\_specs*는 실행문들 안에서 사용될 변수, 상수, Exception 등을 선언한다. 이들 문법 요소에 대한 자세한 내용은 [CREATE PROCEDURE](#), [CREATE FUNCTION](#), [선언문](#), [실행문](#) 절을 참고한다.

저장 프로시저/함수는 Auto Commit 기능이 언제나 비활성화 된 상태로 실행된다. 이는 호출한 세션에서 Auto Commit 기능이 활성화 되어 있어도 마찬가지이다.

저장 프로시저/함수는 **큐브리드 내장 함수**와 동일한 이름을 가질 수 없다. 동일한 이름으로 선언하면 컴파일 과정에서 (CREATE 문 실행 과정에서) 에러가 발생한다.

다음은 PL/CSQL을 사용해서 작성한 저장 프로시저/함수의 예이다.

```
CREATE OR REPLACE PROCEDURE insert_athlete(
 p_name VARCHAR,
 p_gender VARCHAR,
 p_nation_code VARCHAR,
 p_event VARCHAR)
AS
BEGIN
 INSERT INTO athlete (name, gender, nation_code, event)
 VALUES (p_name, p_gender, p_nation_code, p_event);

 COMMIT;
EXCEPTION
 WHEN OTHERS THEN
 ROLLBACK;
END;
```

```
CREATE OR REPLACE PROCEDURE delete_athlete(c INTEGER)
AS
 n_deleted INTEGER;
BEGIN
 DELETE FROM athlete
 WHERE code = c;

 n_deleted := SQL%ROWCOUNT; -- number of deleted rows
 DBMS_OUTPUT.put_line(n_deleted || ' rows deleted');
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.put_line('exception occurred');
END;
```

```
CREATE OR REPLACE FUNCTION fibonacci(n INTEGER) RETURN INTEGER
IS
 invalid_input EXCEPTION;
BEGIN
 IF n <= 0 THEN
 RAISE invalid_input;
 END IF;

 IF n = 1 OR n = 2 THEN
 RETURN 1;
 ELSE
 RETURN fibonacci(n-1) + fibonacci(n-2);
 END IF;
```

```

EXCEPTION
 WHEN invalid_input THEN
 DBMS_OUTPUT.put_line('invalid input: ' || n);
 RETURN -1;
 WHEN OTHERS THEN
 DBMS_OUTPUT.put_line('unknown exception');
 RETURN -1;
END;

```

위 예제들에서 DBMS\_OUTPUT.put\_line() 문은 인자로 주어진 문자열을 서버의 DBMS\_OUTPUT 버퍼에 저장한다. 인자가 문자열 타입이 아닐 때는 문자열로 형변환한 값을 저장한다. DBMS\_OUTPUT 버퍼에 저장된 문자열 메시지들은 CSQL에서 세션 명령어 ;server-output on을 실행해서 확인할 수 있다. 자세한 내용은 [CSQL 세션명령어 server-output](#)을 참조한다.

CREATE PROCEDURE/FUNCTION 문을 실행하면 저장 프로시저/함수의 문법과 실행 의미에 관련된 각종 규칙들을 검사한다. 검사에서 오류가 발견되면 발생한 위치와 원인을 설명하는 오류 메시지를 출력한다. 다음은 오류를 가지고 있는 저장 프로시저가 CSQL에서 에러를 발생시키는 예이다.

```

CREATE OR REPLACE PROCEDURE athlete_code(p_name VARCHAR) AS
BEGIN
 -- 오류: Static SQL SELECT 문은 INTO 절을 가져야 함
 SELECT code
 FROM athlete a
 WHERE a.name = p_name;
END;

ERROR: In line 4, column 5
Stored procedure compile error: SELECT statement must have an INTO
clause

0 command(s) successfully processed.

```

## Static SQL

Static SQL은 문자열에 담지 않고 코드 사이에 직접 기술한 SQL문을 말하며, 접근 방식이 미리 정의되는 형태의 SQL으로, 컴파일 시점에 Syntax, Semantics 체크가 가능하다.

유연하지 않다는 단점이 있지만, 컴파일 시점에 접근 방식이 정의되어 최적화되기 때문에 런타임시 접근 방식이 정의되는 Dynamic SQL보다 빠르고 효과적이라는 장점이 있다.

PL/CSQL 실행문으로 직접 사용할 수 있는 SQL구문은 다음과 같다

- SELECT (CTE, UNION, INTERSECT, MINUS 포함)
- INSERT, UPDATE, DELETE, MERGE, REPLACE
- COMMIT, ROLLBACK

• TRUNCATE

이들의 자세한 문법과 의미는 **CUBRID SQL**을 참고하도록 한다. 위 목록에 포함되지 않는 다른 SQL 문들은 직접 사용할 수는 없으나, 아래에서 설명하는 Dynamic SQL 문을 써서 실행할 수 있다.

SELECT 문은 실행문으로 사용될 뿐만 아니라 **커서를 선언**할 때나 **OPEN-FOR** 문에도 사용된다. SELECT 문의 INTO 절에 프로그램의 변수나 OUT 인자를 사용하여 조회 결과를 담을 수 있다. 이 때 조회 결과 값들의 개수는 INTO 절 안의 변수나 OUT 인자의 개수와 일치해야 하며 값들은 대응되는 변수나 OUT 인자에 대입 가능한 타입을 가져야 한다. SELECT 문을 실행문으로 사용할 때는 INTO 절을 반드시 포함해야 하는 반면 SELECT 문을 **커서 선언**이나 **OPEN-FOR** 문에서 사용할 때는 INTO 절을 포함하지 않아야 한다.

```
CREATE OR REPLACE PROCEDURE test_into_clause_1
AS
 h int;
 s varchar(10);
 CURSOR c IS SELECT host_year, score INTO h, s FROM history;
-- Error: INTO clause
BEGIN
 ...
END;

CREATE OR REPLACE PROCEDURE test_into_clause_2
AS
 h int;
 s varchar(10);
 r SYS_REFCURSOR;
BEGIN
 OPEN r FOR SELECT host_year, score INTO h, s FROM history;
-- Error: INTO clause
 ...
END;

CREATE OR REPLACE PROCEDURE test_into_clause_3
AS
BEGIN
 SELECT host_year, score FROM history WHERE event_code = 20023;
-- Error: no INTO clause
 ...
END;
```

INTO 절을 포함한 SELECT 문의 조회 결과는 한 건의 결과 레코드여야 하며, 두건 이상의 결과를 가져오는 경우는 TOO\_MANY\_ROWS Exception이 발생한다. 결과가 없을 경우에는 NO\_DATA\_FOUND Exception이 발생한다.

Static SQL 문의 WHERE 절이나 VALUES 절 안에서처럼 값을 필요로 하는 자리에 PL/CSQL에서 선언한 변수, 상수, 프로시저/함수 인자를 쓸 수 있다. 단, 이들은 BOOLEAN이나 SYS\_REFCURSOR 타입을 가져서는 안된다. **SQL 데이터타입**이 이들을 포함하지 않기 때문이다.

Static SQL 중에서 SELECT 문에 대해서는 DBLink 기능이 지원되지만, DML (INSERT, UPDATE, DELETE, MERGE, REPLACE) 문에 대해서는 지원되지 않는다. DML 문장에서 DBLink 기능을 사용하려면, 아래에서 설명하는 Dynamic SQL 을 사용해야 한다.

```
CREATE OR REPLACE PROCEDURE test_dblink_in_dml
AS
BEGIN
 INSERT INTO athlete@remote_svr(name, gender, nation_code, event)
 VALUES ('Park Taehwan', 'M', 'KOR', 'Swimming');
END;
```

In line 4, column 8,

```
ERROR: Stored procedure compile error: Semantic: before '
VALUES ('Park Taehwan', 'M', 'KOR', 'Swimming')'
DBLink DML is not yet supported for PL/CSQL Static SQL.
```

다음은 Static SQL 사용 예이다.

```
CREATE OR REPLACE FUNCTION get_medal_count(p_name VARCHAR, p_medal
CHAR) RETURN INTEGER
AS
 n INTEGER;
BEGIN
 -- 일반 실행문으로서의 SELECT 문
 SELECT COUNT(medal)
 INTO n
 FROM athlete a, record r
 WHERE a.code = r.athlete_code /* 조인 조건 */
 AND a.name = p_name AND r.medal = p_medal; /* 필터 조건 */

 RETURN n;
END;
```

```
CREATE OR REPLACE PROCEDURE athlete_history(p_name VARCHAR)
AS
BEGIN
 -- For 루프 안에서의 SELECT 문
 FOR r IN (SELECT host_year, score FROM history WHERE athlete =
p_name) LOOP
 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '
score: ' || r.score);
 END LOOP;
END;
```

```

CREATE OR REPLACE PROCEDURE athlete_history(p_name VARCHAR)
AS
 -- 커서 정의에서의 SELECT 문
 CURSOR my_cursor IS
 SELECT host_year, score
 FROM history
 WHERE athlete = p_name;
BEGIN
 FOR r IN my_cursor LOOP
 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '
score: ' || r.score);
 END LOOP;
END;

```

Static SQL 실행 중에 에러가 나면 SQL\_ERROR Exception이 발생한다.

## Dynamic SQL

Dynamic 문자열에 담아서 기술하는 SQL문을 말하며, 실행 시간에 SQL 구문에 해당하는 문자열을 만들어 **EXECUTE IMMEDIATE** 문으로 실행하는 방식이다.

Dynamic SQL은 주로 다음 두 가지 경우에 필요하다.

- 실행하려는 SQL 구문을 프로그램 작성 시에 결정하는 것이 어렵거나 불가능한 경우
- DDL 문처럼 Static SQL이 지원하지 않는 구문을 실행해야 할 경우

아래 예제에서 새 테이블 이름은 프로시저 인자를 포함하므로 프로그램 작성 시에 결정할 수 없고 프로그램 실행 시간에야 결정되는 값이다. 그리고, DROP TABLE 문과 CREATE TABLE 문은 Static SQL 기능에서 지원하지 않는 DDL 문이다.

```

CREATE OR REPLACE PROCEDURE collect_athlete_history(p_name VARCHAR)
AS
 new_table VARCHAR := p_name || '_history';
BEGIN
 EXECUTE IMMEDIATE 'drop table if exists ' || new_table;
 EXECUTE IMMEDIATE 'create table ' || new_table || ' like
history';
 EXECUTE IMMEDIATE 'insert into ' || new_table || ' select * from
history where athlete = ?'
 USING p_name;
END;

```

## 작성 규칙

식별자, 예약어, 주석, 리터럴을 작성할 때 **Static/Dynamic SQL** 안에서는 **SQL의 작성 규칙**을 따른다.

Static/Dynamic SQL 밖의 PL/CSQL 문 작성 규칙도 대체로 같은 규칙을 따르지만 다음 몇 가지 예외가 있다.

- SQL과 달리 식별자에 '#'을 쓸 수 없다. 즉, 식별자는 영문 대소문자, 한글, 숫자, '\_'(underscore)로만 이루어져야 한다.
- 큰따옴표, 대괄호, 백틱 부호로 둘러싸더라도 식별자에 특수 문자를 쓸 수 없다. 영문 대소문자, 한글, 숫자, '\_'(underscore)만 사용 가능하다.
- 비트열 리터럴을 사용할 수 없다.

허용되는 식별자의 예

```
a
a_b
athleteName2
"select" -- " "로 둘러싸인 예약어
```

허용되지 않는 식별자의 예

```
1a -- 숫자로 시작
a@b -- 특수문자
athlete-name-2 -- 특수문자
[a@b] -- []로 둘러싸더라도 특수문자 불가
select -- 예약어
```

PL/CSQL의 예약어는 아래 표에 나열되어 있다. Static/Dynamic SQL 밖의 PL/CSQL 문장에서 아래 표의 단어들을 변수, 상수, Exception, 내부 프로시저/함수 등의 이름을 나타내는 식별자로 쓸 수 없다. 단, SQL 문에서처럼 큰따옴표(" "), 대괄호([ ]), 백틱(` `)으로 감싸면 식별자로 쓸 수 있다. Static/Dynamic SQL 안에서는 아래 목록이 아니라 일반 SQL 문에 적용되는 **CUBRID 예약어 목록**이 적용된다.

ADDDATE

AND

AS

AUTHID

AUTONOMOUS\_TRANSACTION

BEGIN

BETWEEN

BIGINT

BOOLEAN

BOTH

BY



|                 |                  |               |             |
|-----------------|------------------|---------------|-------------|
| CALLER          | CASE             | CAST          | CHAR        |
| CHARACTER       | CHR              | CLOSE         | COMMENT     |
| COMMIT          | CONSTANT         | CONTINUE      | CREATE      |
| CURRENT_USER    | CURSOR           |               |             |
| DATE            | DATETIME         | DATETIMELTZ   | DATETIMETZ  |
| DATE_ADD        | DATE_SUB         | DAY           | DAY_HOUR    |
| DAY_MILLISECOND | DAY_MINUTE       | DAY_SECOND    | DBMS_OUTPUT |
| DEC             | DECIMAL          | DECLARE       | DEFAULT     |
| DEFINER         | DELETE           | DETERMINISTIC | DIV         |
| DOUBLE          |                  |               |             |
| ELSE            | ELSIF            | END           | ESCAPE      |
| EXCEPTION       | EXECUTE          | EXIT          | EXTRACT     |
| FALSE           | FETCH            | FLOAT         | FOR         |
| FROM            | FUNCTION         |               |             |
| HOUR            | HOUR_MILLISECOND | HOUR_MINUTE   | HOUR_SECOND |
| IF              | IMMEDIATE        | IN            | INOUT       |

|                    |                    |                         |                    |
|--------------------|--------------------|-------------------------|--------------------|
| INSERT             | INT                | INTEGER                 | INTERNAL           |
| INTO               | IS                 | ISO88591                | LANGUAGE           |
| LEADING            | LIKE               | LIST                    | LOOP               |
| MERGE              | MILLISECOND        | MINUTE                  | MINUTE_MILLISECOND |
| MINUTE_SECOND      | MOD                | MONTH                   | MULTISET           |
| NOT                | NULL               | NUMERIC                 |                    |
| OF                 | OPEN               | OR                      | OUT                |
| OWNER              |                    |                         |                    |
| PLCSQL             | POSITION           | PRAGMA                  | PRECISION          |
| PROCEDURE          |                    |                         |                    |
| QUARTER            |                    |                         |                    |
| RAISE              | REAL               | REPLACE                 | RETURN             |
| REVERSE            | ROLLBACK           | RAISE_APPLICATION_ERROR |                    |
| SECOND             | SECOND_MILLISECOND | SEQUENCE                | SELECT             |
| SECOND_MILLISECOND | SEQUENCE           | SELECT                  | SET                |

|              |               |           |               |
|--------------|---------------|-----------|---------------|
| SETEQ        | SETNEQ        | SHORT     | SMALLINT      |
| SQL          | SQLCODE       | SQLERRM   | STRING        |
| SUBDATE      | SUBSET        | SUBSETEQ  | SUPERSET      |
| SUPERSETEQ   | SYS_REFCURSOR |           |               |
| THEN         | TIME          | TIMESTAMP | TIMESTAMPPLTZ |
| TIMESTAMPPTZ | TRAILING      | TRIM      | TRUE          |
| TRUNCATE     |               |           |               |
| UPDATE       | USING         | UTF8      |               |

PL/CSQL 저장 프로시저/함수를 생성하는 CREATE PROCEDURE/FUNCTION 문은 일반 SQL 문법 검사 과정이 아닌, 별도로 구현된 PL/CSQL 문법 검사 과정을 거치며, 예약어 목록도 별개의 것이 적용된다. 단, AS/IS 키워드까지는 일반 문법 검사 과정도 함께 거치기 때문에 일반 SQL에 적용되는 **CUBRID 예약어 목록**도 함께 적용된다. 다음 예제에서 인자 이름 add 는 PL/CSQL 예약어는 아니지만 CUBRID 예약어이기 때문에 문법 에러를 발생시킨다.

```
CREATE OR REPLACE PROCEDURE test_cubrid_reserved_word(add INT) AS
BEGIN
 NULL;
END;

ERROR: invalid create procedure
... ..
```

위 목록 중에서 AUTONOMOUS\_TRANSACTION은 향후 추가할 기능을 위해서 미리 포함되어 있는 예약어이다.

## 데이터 타입

Static/Dynamic SQL에서는 SQL에서 제공하는 모든 **데이터 타입**을 사용할 수 있다.

반면, Static/Dynamic SQL 밖의 PL/CSQL 문에서 사용할 수 있는 데이터 타입은 BOOLEAN, SYS\_REFCURSOR와 SQL에서 제공하는 데이터 타입 중 일부이다.

- BOOLEAN: TRUE, FALSE, NULL을 값으로 가질 수 있다. SQL에서 BOOLEAN 타입을 지원하지 않기 때문에 CREATE PROCEDURE/FUNCTION 문에서 인자 타입이나 리턴 타입으로 BOOLEAN을 사용할 수는 없다. 단, **내부 프로시저/함수**를 선언할 때는 인자 타입이나 리턴 타입으로 BOOLEAN을 사용할 수 있다.
- SYS\_REFCURSOR: 커서 변수를 선언할 때 사용한다. 커서 변수의 용도는 **OPEN-FOR** 문을 참고한다. BOOLEAN과 마찬가지로 CREATE PROCEDURE/FUNCTION 문에서 인자 타입이나 리턴 타입으로 SYS\_REFCURSOR를 사용할 수 없고 **내부 프로시저/함수**에는 사용할 수 있다.

SQL에서 제공하는 데이터 타입 중 PL/CSQL에서 지원하는 것과 지원하지 않는 것은 다음과 같다. (단, 위에서 언급한 대로 Static/Dynamic SQL에서는 SQL에서 제공하는 모든 데이터 타입을 쓸 수 있다.)

| 유형    | 지원                                                                                                             | 미지원                                                      |
|-------|----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| 수치    | SHORT, SMALLINT,<br>INTEGER, INT,<br>BIGINT,<br>NUMERIC, DECIMAL,<br>FLOAT, REAL,<br>DOUBLE, DOUBLE PRECISION, |                                                          |
| 날짜/시간 | DATE, TIME, TIMESTAMP,<br>DATETIME,                                                                            | TIMESTAMPLTZ,<br>TIMESTAMPTZ,<br>DATETIMELTZ, DATETIMETZ |
| 문자열   | CHAR, VARCHAR, STRING,<br>CHAR VARYING                                                                         |                                                          |

컬렉션

| 유형 | 지원 | 미지원                                     |
|----|----|-----------------------------------------|
|    |    | SET, MULTISSET, LIST, SEQUENCE          |
| 기타 |    | BIT, BIT VARYING, ENUM, BLOB/CLOB, JSON |

## %TYPE

테이블 컬럼 이름 뒤에 '%TYPE'을 덧붙여 해당 컬럼의 타입을 나타낼 수 있다. 아래는 %TYPE을 사용하는 예제이다.

```
CREATE OR REPLACE FUNCTION get_athlete_name(p_code
athlete.code%TYPE) RETURN athlete.name%TYPE
AS
 name athlete.name%TYPE;
BEGIN
 SELECT a.name
 INTO name
 FROM athlete a
 WHERE a.code = p_code;

 RETURN name;
END;
```

<table>.<column>%TYPE은 CREATE PROCEDURE/FUNCTION 문을 실행하는 시점에 지정된 테이블 컬럼의 타입을 나타내지만, 나중에 그 컬럼의 타입이 변경되어도 자동으로 <table>.<column>%TYPE을 사용한 저장 프로시저/함수의 동작에 반영되지는 않는다. 그러므로, %TYPE을 적용한 테이블 컬럼의 타입이 변경되었을 때는 그 %TYPE을 사용한 저장 프로시저/함수에 대해서 모두 ALTER PROCEDURE/FUNCTION <name> COMPILE 문을 실행해서 재컴파일해 주어야 한다.

테이블 컬럼 뿐만 아니라 프로시저/함수의 인자나 변수 이름 뒤에 %TYPE을 덧붙여 그 인자나 변수의 타입을 나타낼 수 있다.

```
...
a VARCHAR(10);
a_like a%TYPE; -- 변수 a와 동일한 타입으로 변수 a_like 을 선언
...
```

## %ROWTYPE

테이블 이름 뒤에 %ROWTYPE을 덧붙여서 그 테이블 컬럼들의 이름과 타입을 갖는 필드들로 이루어진 레코드 타입을 나타낼 수 있다. 예를 들어, 다음과 같이 선언된 테이블 tbl에 대해서

```
CREATE TABLE tbl(a INT, b CHAR, c VARCHAR);
```

변수 r을 tbl%ROWTYPE 타입으로 선언하면

```
r tbl%ROWTYPE;
```

r의 값은 필드 a, b, c를 갖는 레코드가 되고 r.a, r.b, r.c는 각각 INT, CHAR, VARCHAR 타입을 갖는다.

커서 이름 뒤에도 %ROWTYPE을 덧붙일 수 있다. 이 때는 커서 정의에 주어진 SELECT 문의 결과에 해당하는 레코드 타입을 나타내게 된다.

```
CURSOR c IS SELECT a, b from tbl;
p c%ROWTYPE; -- p.a, p.b는 각각 INT, CHAR 타입
```

레코드 변수의 선언문에서 초기값을 주지 않았을 때 그 변수는 모든 필드가 NULL인 '빈레코드'로 초기화 된다.

```
r tbl%ROWTYPE; -- r.a, r.b, r.c 모두 NULL. 그러나 r은 NULL 아닌 빈레코드
```

레코드 변수에 NULL을 대입하면 각 필드가 NULL로 초기화 되지만 레코드 변수 값 자체가 NULL이 되지는 않는다. 즉, 레코드 변수는 선언 이후로 NULL 값을 갖는 일이 없다.

동일한 타입의 레코드끼리는 =와 != 연산자로 비교할 수 있다. 여기서 동일 타입 레코드란 하나의 테이블로부터 얻어진 레코드 타입만을 의미하는 것이 아니라 다른 테이블이라도 대응하는 필드들의 이름과 타입이 일치하는 경우까지 포함하는 것이다. 두 레코드에 대한 = 연산의 결과는 대응하는 필드끼리 <=> 연산을 한 결과가 모두 TRUE일 때 TRUE이고 그렇지 않으면 FALSE이다. != 연산의 결과는 = 연산 결과의 반대이다. 다른 타입의 레코드에 =와 != 연산자를 사용했을 때는 컴파일 과정에서 에러가 발생한다.

```

create table tblA(a INT, b CHAR, c VARCHAR);
create table tblB(a INT, b CHAR, c VARCHAR); -- tblA%ROWTYPE과
tblB%ROWTYPE은 동일 타입
create table tblC(aa INT, bb CHAR, cc VARCHAR); -- tblA%ROWTYPE과
tblC%ROWTYPE은 동일 타입 아님

CREATE OR REPLACE PROCEDURE test_record_equality AS
 r1 tblA%ROWTYPE;
 r2 tblB%ROWTYPE;
 r3 tblC%ROWTYPE;
BEGIN
 ...
 if (r1 = r2) then -- OK
 ...
 if (r1 = r3) then -- Error
 ...
END;

```

=와 != 아닌 다른 비교 연산자 <=>, <, >, <=, >= 들은 레코드 비교에 적용할 수 없다.

다음 조건이 만족되는 경우, 레코드 변수 s로부터 다른 레코드 변수 t로의 대입이 가능하다.

- s와 t의 필드 개수가 동일함.
- 각각의 필드 순번 i에 대해서, s와 t의 i번째 필드들의 타입을  $S_i$ 와  $T_i$ 라고 할 때,  $S_i$ 에서  $T_i$ 로 대입 가능함.

레코드 변수 사이에 대입이 가능하기 위해서 같은 순번의 필드명이 동일할 필요는 없다.

```

create table tblAA(a NUMERIC, b DATETIME);
create table tblBB(m INT, n VARCHAR);
create table tblCC(x INT, y TIME);

CREATE OR REPLACE PROCEDURE test_record_assign AS
 r1 tblAA%ROWTYPE;
 r2 tblBB%ROWTYPE;
 r3 tblCC%ROWTYPE;
BEGIN
 ...
 r1 := r2; -- OK
 r1 := r3; -- Error: TIME에서 DATETIME으로 대입 불가 (형변환 불가)
END;

```

%ROWTYPE은 내부 프로시저/함수의 인자 타입과 리턴 타입으로 사용할 수 있다. 그러나, SQL문에서 레코드 타입을 지원하지 않기 때문에 저장 프로시저/함수의 인자 타입과 리턴 타입으로는 사용할 수 없다.

```
CREATE OR REPLACE PROCEDURE sp(a tbl%ROWTYPE) AS -- Error

 PROCEDURE inner(b tbl%ROWTYPE) AS -- OK
 BEGIN
 ...
 END;
BEGIN
 ...
END;
```

Static/Dynamic SQL SELECT 문과 FETCH 문의 INTO 절에 레코드 변수를 사용할 수 있으며, 레코드 변수를 사용할 경우 다른 변수와 함께 사용할 수는 없다. 그리고, 조회 결과와 레코드 변수의 컬럼명은 동일할 필요는 없지만, 조회 컬럼 개수와 레코드 변수의 컬럼 개수는 동일해야 하며, 조회 컬럼 타입과 레코드 변수의 컬럼 타입이 동일하거나 호환 가능해야 한다.

```
CURSOR c IS SELECT a, b from tbl;
whole tbl%ROWTYPE;
part c%ROWTYPE;

-- Static SQL
SELECT * INTO whole from tbl;

-- Dynamic SQL
EXECUTE IMMEDIATE 'SELECT * from tbl' INTO whole;
EXECUTE IMMEDIATE 'SELECT a, b from tbl' INTO part;

-- Fetch
FETCH c INTO part;
```

Static SQL INSERT/REPLACE 문의 VALUES 절에 레코드 변수를 사용할 수 있으며, 레코드 변수를 사용할 경우 다른 변수와 함께 사용할 수 없다. 그리고, 대입 컬럼명과 레코드 변수의 컬럼명은 동일할 필요는 없지만, 대입 컬럼 개수와 레코드 변수의 컬럼 개수는 동일해야 하며, 대입 컬럼 타입과 레코드 변수의 컬럼 타입이 동일하거나 호환 가능해야 한다.

```
INSERT INTO tbl VALUES whole;
INSERT INTO tbl(a, b) VALUES part;
```

이 때 다음과 같은 형태도 가능하다.

```
INSERT INTO tbl SET ROW = whole;
INSERT INTO tbl(a, b) SET ROW = part;
```



Static SQL UPDATE 문에도 다음과 같이 'SET ROW = <record>' 구문을 사용하여 레코드 변수를 사용할 수 있다. 단, 단일 테이블 갱신에만 사용되며 각각의 레코드 필드로부터 동일 순번의 테이블 컬럼으로 대입 가능해야 한다.

```
UPDATE tbl SET ROW = whole WHERE a % 2 = 0;
```

### 정밀도와 스케일 지정 예외

PL/CSQL에서 지원하는 데이터 타입 중에 NUMERIC은 정밀도와 스케일을, CHAR와 VARCHAR는 길이를 지정할 수 있다. 그러나, 저장 프로시저/함수의 인자 타입과 리턴 타입에는 정밀도와 스케일 지정이 허용되지 않는다. 내부 프로시저/함수에서도 마찬가지이다.

```
CREATE OR REPLACE FUNCTION sf(a NUMERIC(5, 3)) RETURN VARCHAR(10)
AS ... -- Error
CREATE OR REPLACE FUNCTION sf(a NUMERIC) RETURN VARCHAR
AS ... -- OK
```

그리고, 일반적으로 정밀도와 스케일이 생략된 NUMERIC은 NUMERIC(15, 0)을 의미하지만 예외적으로 인자 타입 자리에서는 임의의 정밀도와 스케일을 허용함을 의미하고 (단, 정밀도는 1 이상 38 이하, 스케일은 0 이상 정밀도 이하 범위), 리턴 타입 자리에서는 NUMERIC(p, s)를 의미한다. 여기서 p와 s는 시스템 설정 파라미터 stored\_procedure\_return\_numeric\_size에서 지정하는 정밀도와 스케일을 나타낸다. p와 s의 디폴트 값은 각각 38과 15이다.

```
CREATE OR REPLACE FUNCTION test_any_precision_scale(a NUMERIC)
return NUMERIC
AS
BEGIN
 return a;
END;

SELECT test_any_precision_scale(1.23); -- 결과: 1.2300000000000000
SELECT test_any_precision_scale(1.234); -- 결과: 1.2340000000000000
SELECT test_any_precision_scale(1.2345); -- 결과: 1.2345000000000000
```

또한, CHAR와 VARCHAR도 인자 타입과 리턴 타입 자리에서는 다른 자리에서처럼 CHAR(1)과 VARCHAR(1073741823)를 나타내는 것이 아니라 임의의 길이를 갖는 문자열을 허용하는 것으로 동작한다 (단, CHAR 길이는 2048 이하, VARCHAR의 길이는 1073741823 이하 범위).

```
CREATE OR REPLACE FUNCTION test_any_length(a CHAR) return CHAR
AS
BEGIN
 return a;
END;
```

```

SELECT test_any_length('ab'); -- 결과: 'ab'
SELECT test_any_length('abc'); -- 결과: 'abc'
SELECT test_any_length('abcd'); -- 결과: 'abcd'

```

인자 타입과 리턴 타입을 %TYPE을 사용해서 지정했을 때에도 참조되는 원래 타입의 정밀도, 스케일 및 길이 지정은 무시되고 위에 기술한 방식으로 동작한다.

```

CREATE TABLE tbl(p NUMERIC(3,2), q CHAR(3));

CREATE OR REPLACE FUNCTION test_ptype_precision_scale(a tbl.p%TYPE)
RETURN tbl.p%TYPE
AS
BEGIN
 RETURN a;
END;

SELECT test_ptype_precision_scale(1.23); -- 결과:
1.2300000000000000
SELECT test_ptype_precision_scale(1.234); -- 결과:
1.2340000000000000
SELECT test_ptype_precision_scale(1.2345); -- 결과:
1.2345000000000000

CREATE OR REPLACE FUNCTION test_ptype_length(a tbl.q%TYPE) RETURN
tbl.q%TYPE
AS
BEGIN
 RETURN a;
END;

SELECT test_ptype_length('ab'); -- 결과: 'ab'
SELECT test_ptype_length('abc'); -- 결과: 'abc'
SELECT test_ptype_length('abcd'); -- 결과: 'abcd'

```

## 연산자와 함수

Static/Dynamic SQL에서는 SQL에서 제공하는 모든 연산자와 함수를 쓸 수 있다. 반면, Static/Dynamic SQL 밖의 PL/CSQL 문에서는 SQL에서 제공하는 모든 연산자와 함수를 대부분 동일하게 쓸 수 있으나 다음 몇 가지 예외가 있다.

- 지원하지 않는 타입(BIT, ENUM, BLOB/CLOB, JSON, 등)의 값을 인자나 결과로 갖는 연산자와 함수는 쓸 수 없다.
- 나머지 연산자 %를 쓸 수 없다. 단, 동일한 의미의 MOD를 대신 쓸 수 있다.
- 논리 연산자 &&, ||, ! 등을 쓸 수 없다. 단, 각각 동일한 의미의 AND, OR, NOT을 대신 쓸 수 있다.

다음 예제는 문자열 함수 locate과 substr, 그리고 문자열 병합 연산자 ||를 Static/Dynamic SQL 밖의 PL/CSQL 실행문에서도 사용할 수 있음을 보여준다.

```
CREATE OR REPLACE PROCEDURE family_name_to_last
AS
 delim INTEGER;
 family_name VARCHAR;
 given_name VARCHAR;
BEGIN
 FOR r IN (SELECT a.name FROM athlete a limit 5,5) LOOP
 delim := locate(' ', r.name); -- 함수 locate
 family_name := substr(r.name, 1, delim - 1); -- 함수 substr
 given_name := substr(r.name, delim + 1); -- 함수 substr
 DBMS_OUTPUT.put_line(given_name || ' ' || family_name);
 -- 문자열 병합 연산자 ||
 END LOOP;
END;
```

## Exception

PL/CSQL은 다른 많은 프로그래밍 언어와 마찬가지로 Exception 핸들러를 통한 에러 처리를 지원한다 (참고: [Block 실행문](#)). 사용자가 프로그램 선언부에서 자신만의 Exception을 정의하고 실행부에서 사용할 수 있다 (참고: [Exception 선언](#)). 그리고, 주요 에러 상황에 대해서는 다음과 같이 시스템 Exception들이 미리 정의되어 있다.

| 이름                  | SQLCODE | 설명                                     |
|---------------------|---------|----------------------------------------|
| CASE_NOT_FOUND      | 0       | CASE 문에서 조건이 참인 WHEN 절이 없고 ELSE 절도 없음  |
| CURSOR_ALREADY_OPEN | 1       | 이미 열려 있는 커서에 다시 열기 시도                  |
| INVALID_CURSOR      | 2       | 허용되지 않는 커서 조작 (예: 열려 있지 않은 커서를 닫으려고 함) |
| NO_DATA_FOUND       | 3       | SELECT INTO 문 실행 결과 0 개의 Row가 반환됨      |

| 이름            | SQLCODE | 설명                                  |
|---------------|---------|-------------------------------------|
| PROGRAM_ERROR | 4       | 시스템 내부 에러                           |
| STORAGE_ERROR | 5       | 메모리 부족으로 인한 할당 실패                   |
| SQL_ERROR     | 6       | Static/Dynamic SQL 실행 실패            |
| TOO_MANY_ROWS | 7       | SELECT INTO 문 실행 결과 2개 이상의 Row가 반환됨 |
| VALUE_ERROR   | 8       | 잘못된 값에 의한 에러                        |
| ZERO_DIVIDE   | 9       | 0으로 나누기 시도                          |

위에서 각 Exception의 SQLCODE는 **OTHERS Exception 핸들러 block** 안에서 Exception의 종류를 식별하는데 사용할 수 있다.

- 999 이하의 SQLCODE 값들은 시스템 Exception을 위해서 예약되어 있다.
- **사용자가 선언한 Exception**은 SQLCODE 1000 값을 갖는다.
- **RAISE\_APPLICATION\_ERROR**의 첫번째 인자로 지정하는 SQLCODE는 1000보다 큰 값을 가져야 한다.

다음은 Static SQL SELECT 문을 실행할 때 발생할 수 있는 시스템 Exception NO\_DATA\_FOUND와 TOO\_MANY\_ROWS를 처리하는 간단한 예제이다.

```
CREATE OR REPLACE FUNCTION athlete_code(p_name VARCHAR) RETURN
integer
AS
 c INTEGER;
BEGIN
 -- SELECT INTO 문은 단 하나, 그리고 오직 하나의 Row를 결과로 가져야 함
 SELECT code
 INTO c
 FROM athlete a
 WHERE a.name = p_name;
```

```

 RETURN c;
EXCEPTION
 WHEN NO_DATA_FOUND THEN
 DBMS_OUTPUT.put_line('error: no rows found for athlete name
' || p_name);
 RETURN -1;
 WHEN TOO_MANY_ROWS THEN
 DBMS_OUTPUT.put_line('error: more than one rows found for
athlete name ' || p_name);
 RETURN -1;
END;
```

발생한 Exception에 대한 처리를 명시적 (WHEN ... THEN ... 절)으로 처리하지 않은 경우에는 코드상의 Exception 발생 위치와 에러메시지가 출력된다. 예를 들어, 위 athlete\_code()에서 Exception 처리 절들을 삭제하고

```

CREATE OR REPLACE FUNCTION athlete_code(p_name VARCHAR) RETURN
integer
AS
 c INTEGER;
BEGIN
 -- SELECT INTO 문은 단 하나, 그리고 오직 하나의 Row를 결과로 가져야 함
 SELECT code
 INTO c
 FROM athlete a
 WHERE a.name = p_name;

 RETURN c;
END;
```

CSQL에서 athlete 테이블에 존재하지 않는 이름을 인자로 주어 NO\_DATA\_FOUND Exception을 일으켰을 때 결과는 다음과 같다.

```

select athlete_code('x');
```

In the command from line 1,

```

ERROR: Stored procedure execute error:
(line 6, column 5) no data found

0 command(s) successfully processed.
```

위에서 위치 1은 SELECT 문 안에서의 위치를 나타내고, (6, 5)는 athlete\_code()를 선언한 CREATE 문 안에서의 위치를 나타낸다.

## 시스템 설정 적용

Static/Dynamic SQL 문의 동작은 **시스템 설정 파라미터** 전체의 영향을 동일하게 받는다.

Static/Dynamic SQL 밖의 PL/CSQL 문에서는 다음 4개 시스템 설정 파라미터만이 유효하다.

- compat\_numeric\_division\_scale
- oracle\_compat\_number\_behavior
- oracle\_style\_empty\_string
- timezone

```
CREATE OR REPLACE PROCEDURE test_system_config
AS
BEGIN
 -- compat_numeric_division_scale가 no일 때 0.125000000, yes일 때
 0.1
 dbms_output.put_line(1.0 / 8.0);

 -- oracle_compat_number_behavior가 no일 때 1, yes일 때 2
 dbms_output.put_line(3 / 2);

 -- oracle_style_empty_string가 no일 때 'false', yes일 때 'true'
 if '' IS NULL THEN
 dbms_output.put_line('true');
 ELSE
 dbms_output.put_line('false');
 END IF;
END;
```

이들 설정의 자세한 의미는 **시스템 설정 파라미터**를 참조할 수 있다.

위 4개 외 다른 설정은 Static/Dynamic SQL 밖의 PL/CSQL 문에서 유효하지 않다. 특히,

- no\_backslash\_escapes 설정 파라미터값과 상관없이 backslash 문자는 escape 문자로 사용되지 않는다.
- pipes\_as\_concat 설정 파라미터값과 상관없이 ||는 논리합(OR) 연산자로 사용되지 않는다.
- plus\_as\_concat 설정 파라미터값과 상관없이 +는 문자열에 적용되었을 때 병합 연산자로 사용된다.

```
CREATE OR REPLACE PROCEDURE test_system_config_2
AS
BEGIN
 -- no_backslash_escapes 값에 상관없이 'Hello\nworld'
 dbms_output.put_line('Hello\nworld');
```

```
-- pipes_as_concat 값에 상관없이 'ab'
dbms_output.put_line('a' || 'b');

-- plus_as_concat 값에 상관없이 '12'
dbms_output.put_line('1' + '2');

END;
```

## 선언문

프로시저/함수 선언문, 그리고 Block 실행문에는 선언부 *seq\_of\_declare\_specs*가 존재한다. 선언부에서는 아래 문법에서 정의하는 바와 같이 변수, 상수, Exception, 커서, 내부 프로시저/함수를 선언할 수 있다. 선언된 각 항목들은 선언부를 뒤따르는 *body* 안에서 참조할 수 있다.

```
<seq_of_declare_specs> ::= <declare_spec> [<declare_spec> ...]
<declare_spec> ::=
 <variable_decl>
 | <constant_decl>
 | <exception_decl>
 | <cursor_decl>
 | <inner_procedure_decl>
 | <inner_function_decl>
```

선언 가능한 각 항목에 대한 설명은 아래 내용을 참고한다.

**내부 프로시저/함수 선언**과 **Block 실행문**은 자신만의 선언부와 실행부를 가지면서 중첩된 scope들을 이룬다. 이 때 안쪽 scope 선언부에서 바깥에서 선언한 항목과 동일한 이름을 가진 다른 항목을 선언하면 안쪽 scope에서 그 이름은 새로 정의한 항목을 가리키며 바깥쪽의 동일 이름은 가려진다. 단, 안쪽 scope 밖에서 그 이름은 여전히 바깥 항목을 가리킨다.

```
CREATE OR REPLACE PROCEDURE hidden_variable
AS
 a INT := 3;
 b VARCHAR(10);

 -- 내부 프로시저
 PROCEDURE inner_proc
 AS
 a INT := 5;
 b FLOAT;
 BEGIN
 -- 여기서 a = 5, b는 FLOAT 타입
 ...
 END;

BEGIN
 -- 여기서 a = 3, b는 VARCHAR(10) 타입
```

```

...

-- Block 실행문
DECLARE
 a INT := 7;
 b DATETIME;
BEGIN
 -- 여기서 a = 7, b는 DATETIME 타입
 ...
END;

-- 다시 a = 3, b는 VARCHAR(10) 타입
...
END;

```

이러한 “이름 가림”(Name Hiding)은 다른 종류의 항목(상수, 프로시저/함수 인자, Exception, 커서, 내부 프로시저/함수)들에 대해서도 동일하게 적용된다.

단, 가려지는 항목이 동일 선언부 위쪽에서 다른 변수나 상수의 초기값 표현식에 사용되었다면 컴파일 과정에서 에러가 발생한다. 다음은 그 간단한 예이다. 프로시저 poo의 인자 a를 내부 프로시저 inner 안에서 선언한 변수 i의 초기값으로 사용하고 그 아래쪽에서 a 이름으로 다시 변수를 선언하였다. 이런 경우에는 ‘... already been used ... in the same declaration block’ 이라는 메시지의 에러가 발생한다.

```

CREATE OR REPLACE PROCEDURE poo(a INT) AS

 PROCEDURE inner AS
 i INT := a;
 a NUMERIC;
 BEGIN
 ...
 END;

BEGIN
 ...
END;

ERROR: In line 5, column 9
Stored procedure compile error: name A has already been used at line
4 and column 18 in the same declaration block

```

## 변수 선언

```

<variable_decl> ::=
 <identifier> <type_spec> [[NOT NULL] <initial_value_part>] ;

```



```

<type_spec> ::=
 <builtin_type>
 | <table>.<column>%TYPE
 | <variable>%TYPE
 | <table>%ROWTYPE
 | <cursor>%ROWTYPE
<initial_value_part> ::= { := | DEFAULT } <expression>

```

· *builtin\_type*: 데이터 타입 절에서 설명한 시스템 제공 타입

변수 선언에 선택적으로 NOT NULL 조건과 초기값을 지정할 수 있다. NOT NULL 조건이 지정된 경우에는 반드시 NULL이 아닌 초기값이 함께 지정되어야 한다. 선언할 때 초기값이 지정되지 않은 변수는 묵시적으로 NULL 값을 갖게 된다.

```

CREATE OR REPLACE PROCEDURE test_variable
AS
 a INT NOT NULL := 3;
 b VARCHAR(1) := 's';
 c FLOAT; -- c = NULL
BEGIN
 ...
END;

```

## 상수 선언

```

<constant_decl> ::=
 <identifier> CONSTANT <type_spec> [NOT_NULL] <value_part> ;

<type_spec> ::=
 <builtin_type>
 | <table>.<column>%TYPE
 | <variable>%TYPE
 | <table>%ROWTYPE
 | <cursor>%ROWTYPE
<value_part> ::= { := | DEFAULT } <expression>

```

· *builtin\_type*: 데이터 타입 절에서 설명한 시스템 제공 타입

상수 선언에는 필수적으로 값 지정이 포함되어야 한다. NOT NULL 조건이 지정된 경우, 이 값은 NULL이 아니어야 한다.

```

CREATE OR REPLACE PROCEDURE test_constant
AS
 a CONSTANT INT NOT NULL := 3;
 b CONSTANT VARCHAR := 's';

```

```
--c CONSTANT FLOAT; -- 예러
BEGIN
 ...
END;
```

## Exception 선언

```
<exception_decl> ::=
 <identifier> EXCEPTION ;
```

사용자가 원하는 이름의 Exception을 선언할 수 있다. 이렇게 선언된 Exception을 **RAISE** 문과 Exception 처리의 **WHEN** 절에서 사용할 수 있다.

```
CREATE OR REPLACE FUNCTION text_exception(n INT) RETURN INT
AS
 negative_argument EXCEPTION;
 too_big_argument EXCEPTION;
BEGIN
 IF n < 0 THEN
 RAISE negative_argument;
 ELSIF n > 100 THEN
 RAISE too_big_argument;
 ELSIF n = 0 THEN
 RETURN 0;
 END IF;
 ...
EXCEPTION
 WHEN negative_argument THEN
 DBMS_OUTPUT.put_line('error: negative argument ' || n);
 return -1;
 WHEN too_big_argument THEN
 DBMS_OUTPUT.put_line('error: too big argument ' || n);
 return -2;
END;
```

## 커서 선언

```
<cursor_decl> ::=
 CURSOR <identifier> [(<seq_of_cursor_parameters>)] IS
 <select_statement> ;

<seq_of_cursor_parameters> ::= <cursor_parameter> [,
 <cursor_parameter>, ...]
```

```

<cursor_parameter> ::= <identifier> [IN] <type_spec>
<type_spec> ::=
 <builtin_type>
 | <table>.<column>%TYPE
 | <variable>%TYPE
 | <table>%ROWTYPE
 | <cursor>%ROWTYPE

```

· *builtin\_type*: 데이터 타입 절에서 설명한 시스템 제공 타입

커서에도 프로시저/함수와 유사하게 인자를 선언할 수 있지만 오직 IN 인자만 선언할 수 있다는 차이가 있다. 이 인자를 *select\_statement* 문 안에서 참조할 수 있다. 커서를 OPEN 할 때 이 인자에 실제 선언된 개수와 타입이 일치하도록 인자값을 채워 해당 SELECT 문을 실행한다.

```

CREATE OR REPLACE PROCEDURE test_cursor(p_name VARCHAR, p_year
INTEGER)
AS
 CURSOR my_cursor(a VARCHAR, y INTEGER) IS
 SELECT host_year, score
 FROM history
 WHERE athlete = a AND host_year >= y;

 target_year INT;
 target_score VARCHAR(10);
BEGIN
 OPEN my_cursor(p_name, p_year);
 LOOP
 FETCH my_cursor INTO target_year, target_score;
 EXIT WHEN my_cursor%NOTFOUND;
 DBMS_OUTPUT.put_line('host_year: ' || target_year || '
score: ' || target_score);
 END LOOP;
 CLOSE my_cursor;
END;

```

커서는 위 예제처럼 명시적으로 OPEN, FETCH, CLOSE 실행문을 통해 이용할 수 있다. 반면, 아래 예제처럼 OPEN, FETCH, CLOSE 동작이 묵시적으로 이루어지는 For-Loop 문을 통해서 커서를 이용할 수도 있다.

```

CREATE OR REPLACE PROCEDURE test_cursor_loop(p_name VARCHAR, p_year
INTEGER)
AS
 CURSOR my_cursor(a VARCHAR, y INTEGER) IS
 SELECT host_year, score
 FROM history
 WHERE athlete = a AND host_year >= y;
BEGIN
 FOR r IN my_cursor(p_name, p_year) LOOP

```

```

 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '
score: ' || r.score);
 END LOOP;
END;
```

커서를 선언할 때 사용하는 SELECT 문에는 INTO 절을 쓸 수 없다.

## 내부 프로시저/함수 선언

정의 중인 저장 프로시저/함수 안에서만 사용할 내부 프로시저/함수를 다음 문법에 따라 정의할 수 있다. 어느 정도 규모를 이루거나 두 번 이상 반복되는 실행 과정을 내부 프로시저/함수로 묶어 모듈화하면 프로그램 가독성이 높아지고 모듈화한 부분의 코드 재사용성이 높아진다.

```

<inner_procedure_decl> ::=
 PROCEDURE <identifier> [(<seq_of_parameters>)] { IS | AS } [
<seq_of_declare_specs>] <body> ;
<inner_function_decl> ::=
 FUNCTION <identifier> [(<seq_of_parameters>)] RETURN
<type_spec> { IS | AS } [<seq_of_declare_specs>] <body> ;

<seq_of_parameters> ::= [<parameter> [, <parameter> ...]]
<parameter> ::= <identifier> [{ IN | IN OUT | INOUT | OUT }]
<type_spec> [COMMENT 'param_comment_string']
<type_spec> ::=
 <builtin_type>
 | <table>.<column>%TYPE
 | <variable>%TYPE
 | <table>%ROWTYPE
 | <cursor>%ROWTYPE
<body> ::= BEGIN <seq_of_statements> [EXCEPTION <seq_of_handlers>]
END [<label_name>]
<seq_of_declare_specs> ::= <declare_spec> [<declare_spec> ...]
<seq_of_statements> ::= <statement> ; [<statement> ; ...]
<seq_of_handlers> ::= <handler> [<handler> ...]
<handler> ::= WHEN <exception_name> [OR <exception_name> OR ...]
THEN <seq_of_statements>
<exception_name> ::= identifier | OTHERS
```

- *parameter*: 인자는 IN, IN OUT, INOUT, OUT 네 가지 경우로 선언할 수 있다. IN OUT과 INOUT은 동일한 효과를 갖는다.
- *param\_comment\_string*: 인자 커멘트 문자열을 지정한다.
- *builtin\_type*: 데이터 타입 절에서 설명한 시스템 제공 타입
- *body*: 하나 이상의 실행문과 선택적으로 몇 개의 Exception 핸들러로 구성된다.

- *label\_name*: 프로시저/함수 이름과 일치해야 한다.
- *declare\_spec*: 변수, 상수, Exception, 커서, 내부 프로시저/함수 선언 중 하나
- *statement*: 아래 **실행문** 절 참조
- *handler*: 지정된 Exception이 발생했을 때 실행할 실행문들을 지정한다.
- *exception\_name*: Exception 이름 *identifier*는 **시스템 Exception**이거나 **사용자가 선언한 것**이어야 한다. OTHERS는 아직까지 매치되지 않은 모든 Exception에 매치되며 OR로 다른 exception 이름과 연결할 수 없다.

내부 프로시저/함수는 **저장 프로시저/함수**와 달리 **큐브리드 내장 함수**와 동일한 이름을 가질 수 있다. 이 때 내장 함수는 내부 프로시저/함수가 선언된 scope 안에서 가려진다.

함수의 경우에는 *body*에서 RETURN 문으로 선언된 리턴 타입에 맞는 값을 반환해야 한다. 함수가 *body* 끝에 도달할 때까지 RETURN 문을 만나지 못하는 실행경로가 존재하면 컴파일 과정에서 에러가 발생한다. 프로시저의 경우에는 RETURN 문에 반환값을 지정할 수 없다.

프로시저/함수는 자기 자신을 실행부에서 참조할 수 있다. 즉, 재귀 호출이 가능하다.

```
CREATE OR REPLACE FUNCTION choose(m INT, n INT) RETURN INT
AS
 invalid_argument EXCEPTION;

 -- 내부 함수 선언
 FUNCTION factorial(n INT) RETURN INT
 AS
 BEGIN
 IF n < 0 THEN
 RAISE invalid_argument;
 ELSIF n <= 1 THEN
 RETURN 1;
 ELSE
 RETURN n * factorial(n - 1); -- 재귀 호출
 END IF;
 END;
BEGIN
 IF n > m OR n < 0 THEN
 RAISE invalid_argument;
 ELSE
 RETURN factorial(m) / factorial(n) / factorial(m - n);
 END IF;
END;
```

동일한 선언부에서 선언된 내부 프로시저/함수끼리는 상호 재귀 호출도 가능하다.

```
CREATE OR REPLACE PROCEDURE ping_pong(cnt INT)
AS
```

```

PROCEDURE ping(n INT)
AS
BEGIN
 IF n <= 0 THEN
 DBMS_OUTPUT.put_line('-- end --');
 ELSE
 DBMS_OUTPUT.put_line('ping ->');
 pong(n - 1); -- 상호 재귀 호출
 END IF;
END;

PROCEDURE pong(n INT)
AS
BEGIN
 IF n <= 0 THEN
 DBMS_OUTPUT.put_line('-- end --');
 ELSE
 DBMS_OUTPUT.put_line(' <- pong');
 ping(n - 1); -- 상호 재귀 호출
 END IF;
END;
BEGIN
 ping(cnt);
END;

```

재귀 호출을 사용할 때는 무한 루프에 빠지지 않도록 종료 조건을 적절히 주어야 한다.

## 실행문

현재 PL/SQL은 다음과 같이 15가지 종류의 실행문을 제공한다.

```

<statement> ::=
 <block>
 | <sql_statement>
 | <cursor_manipulation>
 | <raise_application_error>
 | <execute_immediate>
 | <assignment_statement>
 | <continue_statement>
 | <exit_statement>
 | <null_statement>
 | <raise_statement>
 | <return_statement>
 | <procedure_call>
 | <if_statement>
 | <loop_statement>
 | <case_statement>

```

## BLOCK

BLOCK 문은 실행문들 중간에 중첩 scope을 만들어 그 안에서 새로운 변수, 상수 등을 선언하고 사용할 수 있게 한다. BLOCK은 프로시저/함수와 마찬가지로 Exception 처리 구조를 가질 수 있다.

```
<block> ::=
 [DECLARE <seq_of_declare_specs>] <body>

<body> ::= BEGIN <seq_of_statements> [EXCEPTION <seq_of_handlers>]
END [<label_name>]
<seq_of_declare_specs> ::= <declare_spec> [<declare_spec> ...]
<seq_of_statements> ::= <statement> ; [<statement> ; ...]
<seq_of_handlers> ::= <handler> [<handler> ...]
<handler> ::= WHEN <exception_name> [OR <exception_name> OR ...]
THEN <seq_of_statements>
<exception_name> ::= identifier | OTHERS
```

- *body*: 필수적으로 하나 이상의 실행문과 선택적으로 몇 개의 Exception 핸들러로 구성된다.
- *declare\_spec*: 변수, 상수, Exception, 커서, 내부 프로시저/함수 선언. (참조: 선언문)
- *handler*: 지정된 Exception이 발생했을 때 실행할 실행문들을 지정한다.
- *exception\_name*: Exception 이름 *identifier*는 시스템 Exception이거나 사용자가 선언한 것이어야 한다. OTHERS는 선언되지 않은 모든 Exception에 매치되며 OR로 다른 exception 이름과 연결할 수 없다.

BLOCK 안에서 선언된 아이템들은 그 BLOCK을 벗어나면 참조할 수 없다. BLOCK에서 선언된 아이템이 바깥 scope에서 선언된 다른 아이템과 이름이 겹칠 경우 바깥 아이템은 그 BLOCK 안에서 참조할 수 없게 된다 (가려진다).

```
CREATE OR REPLACE PROCEDURE test_block
IS
 a INT := 3;
 b INT := 3;
 c INT := 3;
BEGIN
 DECLARE
 a INT := 5;
 b INT := 5;
 BEGIN
 DECLARE
 a INT := 7;
 BEGIN
 DBMS_OUTPUT.put_line(a || b || c); -- '753'
 END;
 DBMS_OUTPUT.put_line(a || b || c); -- '553'
 END;
END;
```

```
DBMS_OUTPUT.put_line(a || b || c); -- '333'
END;
```

body 내부의 실행문들 중 수행시 도달할 수 없는 실행문이 있는 경우 컴파일 과정에서 에러를 발생한다. 다음은 도달할 수 없는 실행문이 있는 간단한 예이다.

```
CREATE OR REPLACE PROCEDURE test_unreachable_statement
AS
BEGIN
 RETURN;
 DBMS_OUTPUT.put_line('Hello world');
END;

ERROR: In line 5, column 5
Stored procedure compile error: unreachable statement
```

## Static SQL

**Static SQL** 절에서 설명한대로 SQL 문 중에서 SELECT, INSERT, UPDATE, DELETE, MERGE, REPLACE, COMMIT, ROLLBACK, TRUNCATE 문은 프로그램의 실행문으로서 직접 사용 가능하다.

## 커서 조작문

커서 조작문은 아래와 같이 4 가지 종류가 있다.

```
<cursor_manipulation> ::=
 <open_statement>
 | <fetch_statement>
 | <close_statement>
 | <open_for_statement>

<open_statement> ::= OPEN <cursor> [<function_argument>]
<fetch_statement> ::= FETCH <cursor_expression> INTO <identifier>
[, <identifier>, ...]
<close_statement> ::= CLOSE <cursor_expression>

<open_for_statement> ::= OPEN <identifier> FOR <select_statement>
```

- *cursor\_expression*: 계산 결과로 커서나 SYS\_REFCURSOR 변수를 갖는 표현식
- *open\_statement*: 커서를 연다. SYS\_REFCURSOR 변수가 아닌 커서에 대해서만 사용가능함에 주의하자. 인자를 갖도록 선언된 커서에 대해서는 선언된 인자 개수와 타입에 맞는 값을 주면서 열어야 한다. 이미 열려 있는 커서를 다시 열려고 시도하면 CURSOR\_ALREADY\_OPEN Exception이 발생한다.



- *fetch\_statement*: 커서로부터 하나의 row를 가져와 지정된 변수나 OUT 인자에 대입한다. row 안의 컬럼 개수는 지정된 변수나 OUT 인자 개수와 일치해야 하고 각각의 컬럼값은 해당 변수나 OUT 인자에 대입 가능한 타입을 가져야 한다. 열려 있지 않은 커서로부터 FETCH를 시도하면 INVALID\_CURSOR Exception이 발생한다.
- *close\_statement*: 커서를 닫는다. 열려 있지 않은 커서를 닫으려고 시도하면 INVALID\_CURSOR Exception이 발생한다.
- *open\_for\_statement*: *identifier*는 SYS\_REFCURSOR 타입으로 선언된 변수이어야 한다. 지정된 *select\_statement*를 실행하는 커서를 내부적으로 열어서 지정된 변수에 할당한다. *select\_statement*가 INTO 절을 포함하면 컴파일 과정에서 에러가 발생한다.

다음은 OPEN, FETCH, CLOSE 문의 사용 예이다.

```
CREATE OR REPLACE PROCEDURE test_cursor(p_name VARCHAR, p_year
INTEGER)
AS
 CURSOR my_cursor(a VARCHAR, y INTEGER) IS
 SELECT host_year, score
 FROM history
 WHERE athlete = a AND host_year >= y;

 target_year INT;
 target_score VARCHAR(10);
BEGIN

 OPEN my_cursor(p_name, p_year);
 LOOP
 FETCH my_cursor INTO target_year, target_score;
 EXIT WHEN my_cursor%NOTFOUND;
 DBMS_OUTPUT.put_line('host_year: ' || target_year || '
score: ' || target_score);
 END LOOP;
 CLOSE my_cursor;
END;
```

다음 예제는 SYS\_REFCURSOR를 OUT 인자로 갖는 내부 프로시저와 OPEN-FOR 문을 이용해서 특정 SELECT 문을 SYS\_REFCURSOR 변수에 연결하고 SELECT 문의 결과를 조회하는 예제이다.

```
CREATE OR REPLACE PROCEDURE test_ref_cursor(p_name VARCHAR)
AS
 my_refcursor SYS_REFCURSOR;

 target_year INT;
 target_score VARCHAR(10);

 PROCEDURE open_refcursor(athlete_name VARCHAR, rc OUT
SYS_REFCURSOR)
```

```

AS
 refcursor SYS_REFCURSOR;
BEGIN
 OPEN refcursor FOR SELECT host_year, score FROM history
WHERE athlete = athlete_name;
 rc := refcursor;
END;
BEGIN
 open_refcursor(p_name, my_refcursor);
 LOOP
 FETCH my_refcursor INTO target_year, target_score;
 EXIT WHEN my_refcursor%NOTFOUND;
 DBMS_OUTPUT.put_line('host_year: ' || target_year || '
score: ' || target_score);
 END LOOP;
 CLOSE my_refcursor;
END;

```

## RAISE\_APPLICATION\_ERROR

RAISE\_APPLICATION\_ERROR는 사용자가 원하는 코드와 에러메시지로 Exception을 발생하고자 할 때 사용한다. RAISE\_APPLICATION\_ERROR의 사용 형태는 Built-in 프로시저 호출처럼 보이지만 내부적으로는 PL/SQL 실행문이다. 첫번째 인자로 주는 코드는 1000보다 큰 INTEGER 값을 가져야 한다. 1000 이하의 값은 시스템을 위해 예약되어 있기 때문이다. 두번째 인자로 주는 에러메시지는 임의의 문자열이 가능하다.

```

CREATE OR REPLACE PROCEDURE test_raise_app_err(i INT)
AS
BEGIN
 CASE i
 WHEN 1 THEN
 RAISE_APPLICATION_ERROR(1001, 'my error 1');
 WHEN 2 THEN
 RAISE_APPLICATION_ERROR(1002, 'my error 2');
 WHEN 3 THEN
 RAISE_APPLICATION_ERROR(1003, 'my error 3');
 END CASE;
EXCEPTION
 WHEN OTHERS THEN
 dbms_output.put_line('code=' || SQLCODE || ', message=' ||
SQLERRM || '');
END;

CALL test_raise_app_err(1); -- DBMS_OUTPUT 출력: code=1001,
message='my error 1'
CALL test_raise_app_err(2); -- DBMS_OUTPUT 출력: code=1002,
message='my error 2'

```

```
CALL test_raise_app_err(3); -- DBMS_OUTPUT 출력: code=1003,
message='my error 3'
```

## EXECUTE IMMEDIATE

**Dynamic SQL** 절에서 설명한 바와 같이 실행 시간에 임의의 SQL을 문자열로 구성하여 EXECUTE IMMEDIATE 문을 통해 실행할 수 있다. USING 절을 써서 프로그램의 어떤 값을 SQL문의 호스트 변수 자리에 채우는 것이 가능하다. INTO 절을 써서 SELECT 문의 조회 결과를 프로그램의 변수나 OUT 인자에 담아오는 것도 가능하다. 이 때 조회 결과 값들의 개수는 INTO 절 안의 변수나 OUT 인자의 개수와 일치해야 하고 값들은 대응되는 변수나 OUT 인자에 대입 가능한 타입을 가져야 한다.

SQL 문 실행 중에 에러가 나면 SQL\_ERROR Exception이 발생한다. INTO 절을 포함한 경우 SELECT 문의 조회 결과는 단 한 건의 결과 레코드를 가져야 한다. 결과가 없을 때는 NO\_DATA\_FOUND Exception이 발생하고 결과가 두 건 이상일 때는 TOO\_MANY\_ROWS Exception이 발생한다.

```
<execute_immediate> ::=
 EXECUTE IMMEDIATE <dynamic_sql> { [<into_clause>] [
<using_clause>] | <using_clause> <into_clause> }
 <using_clause> ::= USING <using_element> [,
<using_element>, ...]
 <using_element> ::= [IN] <expression>
 <into_clause> ::= INTO <identifier> [, <identifier>, ...]
```

- *dynamic\_sql*: 문자열 타입을 갖는 표현식. 표현식은 SQL 규약에 맞는 SQL 구문 문자열을 계산 결과로 가져야 한다. SQL 구문 내부에서 값을 필요로 하는 자리에 ?(물음표)를 대신 쓸 수 있으며 사용한 ?의 개수와 *using\_clause*에 포함된 표현식의 개수는 일치해야 한다.
- *using\_clause*: *dynamic\_sql*을 실행할 때 문자열의 ? 자리에 채워질 값들을 지정한다. BOOLEAN이나 SYS\_REFCURSOR 타입을 갖는 표현식을 가질 수 없다. %ROWTYPE으로 선언된 레코드 타입 값이나 커서도 표현식 자리에 올 수 없다.
- *into\_clause*: *dynamic\_sql*이 SELECT문을 나타내는 경우에 조회 결과를 담을 변수나 OUT 인자를 지정한다. *dynamic\_sql*이 SELECT문을 나타내는데 INTO 절이 없거나 *dynamic\_sql*이 SELECT문을 나타내지 않는데 INTO 절이 있으면 SQL\_ERROR Exception이 발생한다.

다음은 EXECUTE IMMEDIATE의 사용 예이다.

```
CREATE OR REPLACE PROCEDURE collect_athlete_history(p_name VARCHAR)
AS
 new_table VARCHAR := p_name || '_history';
BEGIN
 EXECUTE IMMEDIATE 'drop table if exists ' || new_table;
 EXECUTE IMMEDIATE 'create table ' || new_table || ' like
history';
 EXECUTE IMMEDIATE 'insert into ' || new_table || ' select * from
```

```
history where athlete = ?'
 USING p_name;
END;
```

## 대입문

```
<assignment_statement> ::=
 <identifier> := <expression>
```

- *identifier*: 변수이거나 OUT 인자이어야 한다.
- *expression*: 대입될 값을 계산하는 표현식. 아래 표현식 절 참조

*expression*의 타입은 *identifier*의 타입과 같거나 *identifier*의 타입으로 형변환이 가능해야 한다. 그렇지 않으면 컴파일 과정에서 에러가 발생한다.

## CONTINUE, EXIT

```
<continue_statement> ::=
 CONTINUE [<label_name>] [WHEN <expression>]
```

```
<exit_statement> ::=
 EXIT [<label_name>] [WHEN <expression>]
```

CONTINUE와 EXIT 문은 루프문 안에서만 사용할 수 있다. CONTINUE 문은 아래쪽으로는 실행 흐름을 멈추고 루프의 처음으로 분기해서 다음 iteration을 실행하도록 한다. EXIT 문은 아래쪽으로는 실행 흐름을 멈추고 루프를 빠져나가 그 루프 다음 실행문으로 분기한다. *label\_name*이 없는 경우 그 CONTINUE/EXIT 문을 포함하는 가장 안쪽의 루프를 재시작한다/빠져나간다. *label\_name*이 있는 경우 그 CONTINUE/EXIT 문을 포함하는 루프들 중 하나에 선언된 것이어야 한다. 아니면 컴파일 과정에서 에러가 발생한다. 루프가 여럿 중첩된 경우 *label\_name*을 지정하여 분기할 루프를 지정할 수 있다. WHEN 절이 있는 경우 BOOLEAN 타입의 *expression*이 TRUE로 계산될 경우에만 분기한다.

```
CREATE OR REPLACE PROCEDURE test_continue_exit
AS
 i INT := 0;
BEGIN
 LOOP
 DBMS_OUTPUT.put_line(i); -- 0, 1, 2, 3, 4, 5
 i := i + 1;
 CONTINUE WHEN i < 3;
 DBMS_OUTPUT.put_line(i); -- 3, 4, 5
 END LOOP;
END;
```

```

 EXIT WHEN i = 5;
 END LOOP;

 DBMS_OUTPUT.put_line(i); -- 5
END;
```

## NULL

```

<null_statement> ::=
 NULL
```

아무 일도 하지 않는다는 것을 명시적으로 표시하고 싶을 경우, 혹은 나중에 구현할 실행문 자리를 임시로 채워 넣고 싶을 경우 등에 NULL을 사용할 수 있다. 문법상 실행문 자리를 채우기 위한 place holder 구문이다.

```

CREATE OR REPLACE PROCEDURE test_null(medal CHAR)
AS
BEGIN
 CASE medal
 WHEN 'G' THEN
 DBMS_OUTPUT.put_line('Gold');
 WHEN 'S' THEN
 DBMS_OUTPUT.put_line('Silver');
 WHEN 'B' THEN
 DBMS_OUTPUT.put_line('Bronze');
 ELSE
 NULL;
 END CASE;
END;
```

## RAISE

```

<raise_statement> ::=
 RAISE [<identifier>]
```

Exception을 발생시킨다. Exception 이름 *identifier*는 시스템 Exception이거나 사용자가 선언한 것이어야 한다. Exception의 THEN 절 안의 RAISE는 Exception 이름을 생략할 수 있다. 이 경우, 현재 처리 중인 Exception을 다시 일으키는 것으로 동작한다.

```

CREATE OR REPLACE FUNCTION fibonacci(n INTEGER) RETURN INTEGER
IS
 invalid_input EXCEPTION;
```

```

BEGIN
 IF n <= 0 THEN
 RAISE invalid_input;
 END IF;

 IF n = 1 OR n = 2 THEN
 RETURN 1;
 ELSE
 RETURN fibonacci(n-1) + fibonacci(n-2);
 END IF;
EXCEPTION
 WHEN invalid_input THEN
 DBMS_OUTPUT.put_line('invalid input: ' || n);
 RAISE; -- 현재 처리 중인 invalid_input을 다시 일으킴
 WHEN OTHERS THEN
 DBMS_OUTPUT.put_line('unknown exception');
 RAISE; -- 현재 처리 중인 Exception을 다시 일으킴
END;

```

## RETURN

```

<return_statement> ::=
 RETURN [<expression>]

```

현재 루틴을 호출한 호출문 다음으로 분기한다. 현재 루틴이 함수인 경우에는 그 함수의 리턴 타입으로 변환 가능한 반환값 *expression*을 지정해야 한다. 현재 루틴이 함수가 아닌 프로시저인 경우에는 반환값을 지정하면 에러이다.

## 프로시저 호출문

```

<procedure_call> ::=
 <identifier> [<function_argument>]
<function_argument> ::= ([<expression> [, <expression>, ...]])

```

이름 *identifier*로 지정된 프로시저를 인자 *function\_argument*를 주어 호출한다. 인자 개수와 각각의 타입은 해당 프로시저의 선언과 일치해야 한다. 호출되는 프로시저의 OUT 인자에 주어질 인자들은 프로시저 호출 결과로 변경이 될 것이므로 대입이 가능한 변수나 다른 OUT 인자이어야 한다.

```

CREATE OR REPLACE PROCEDURE callee(o OUT INT)
AS
BEGIN
 ...
END;

```

```

CREATE OR REPLACE PROCEDURE proc(i INT, o OUT INT)
AS
 v INT;
 c CONSTANT INT := 0;
BEGIN
 callee(i); -- Error: IN 인자
 callee(o); -- OK: OUT 인자
 callee(v); -- OK: 변수
 callee(c); -- Error: 상수
END;

```

호출되는 프로시저는 저장 프로시저이거나 내부 프로시저이다. 다른 저장 프로시저 호출문 실행 중에 문제가 발생했을 때는 SQL\_ERROR Exception이 발생한다.

## IF

```

<if_statement> ::=
 IF <expression> THEN <seq_of_statements> [<elsif_part> [
<elsif_part> ...]] [<else_part>] END IF
<elsif_part> ::= ELSIF <expression> THEN <seq_of_statements>
<else_part> ::= ELSE <seq_of_statements>

```

일반적인 프로그래밍 언어가 제공하는 If-Then-Else 문을 제공한다. IF와 ELSIF 다음의 *expression*는 BOOLEAN 타입이어야 한다.

## LOOP

PL/CSQL이 제공하는 루프문은 아래와 같이 다섯 가지 형태가 있다. 앞의 세 가지는 일반적인 프로그래밍 언어에서 제공하는 루프문과 유사하다. 뒤의 두 가지는 SELECT 문의 조회 결과를 순회하는 용도로 사용한다.

```

<loop_statement> ::=
 <label_declaration>? LOOP <seq_of_statements> END
LOOP # basic-loop
 | <label_declaration>? WHILE <expression> LOOP
<seq_of_statements> END LOOP # while-loop
 | <label_declaration>? FOR <iterator> LOOP <seq_of_statements>
END LOOP # for-iter-loop
 | <label_declaration>? FOR <for_cursor> LOOP <seq_of_statements>
END LOOP # for-cursor-loop
 | <label_declaration>? FOR <for_static_sql> LOOP
<seq_of_statements> END LOOP # for-static-sql-loop

<label_declaration> ::= '<<' <identifier> '>>'

```

```

<iterator> ::= <identifier> IN [REVERSE] <lower_bound> ..
<upper_bound> [BY <step>]

<for_cursor> ::= <record> IN <cursor> [<function_argument>]
<for_static_sql> ::= <record> IN (<select_statement>)

```

- *label\_declaration*: 오직 루프문 시작 부분에서만 라벨 선언을 할 수 있다. 이 라벨은 루프 바디 안 쪽의 CONTINUE 문이나 EXIT 문이 분기 기준이 될 루프를 지정하는데 사용된다.
- *while-loop* 형태의 루프에서 조건 *expression*은 BOOLEAN 타입이어야 한다.
- *for-iter-loop* 형태의 루프에서 *lower\_bound*, *upper\_bound*, *step*은 모두 INTEGER로 변환가능한 타입을 가져야 한다. 실행시간에 *step* 값이 0 이하이면 VALUE\_ERROR Exception이 발생한다. REVERSE가 지정되지 않은 경우, *identifier*는 *lower\_bound*로 초기화 된 후 *upper\_bound*보다 작거나 같다는 조건을 만족하면 루프 바디를 한번 실행하고 그 이후는 *step* 만큼 증가한 값이 *upper\_bound*보다 작거나 같다는 조건을 만족하는 한 반복한다. REVERSE가 지정된 경우에는, *identifier*는 *upper\_bound*로 초기화 된 후 *lower\_bound*보다 크거나 같다는 조건을 만족하면 루프 바디를 한번 실행하고 그 이후는 *step*만큼 감소한 값이 *lower\_bound*보다 크거나 같다는 조건을 만족하는 한 반복한다. 루프 변수 *identifier*는 루프 바디 안에서 INTEGER 타입 변수로 사용될 수 있다.
- *for-cursor-loop*, *for-static-sql-loop* 형태의 FOR 루프는 *record* IN 다음에 기술하는 커서나 SELECT 문의 조회 결과들을 순회하기 위해 사용된다. 이 때 사용되는 SELECT 문에 INTO 절이 있으면 컴파일 과정에서 에러가 발생한다. 매 iteration 마다 조회 결과가 한 row 씩 *record*에 할당된 상태로 루프 바디가 실행된다. 이 때, 결과 row의 각 컬럼들은 루프 바디 안에서 *record.column* 모양으로 참조할 수 있다.

기본 형태 LOOP는 보통 아래와 같이 반복 종료를 위한 조건을 내부에 가지게 된다.

```

CREATE OR REPLACE PROCEDURE print_nation_athletes(nation CHAR)
AS
 code INT;
 name VARCHAR(40);
 CURSOR c IS SELECT code, name from athlete where nation_code =
nation;
BEGIN
 OPEN c;
 LOOP
 FETCH c INTO code, name;
 EXIT WHEN c%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE('code: ' || code || ' name: ' || name);
 END LOOP;
 CLOSE c;
END;

```

다음은 While Loop 구문의 간단한 사용 예를 보여준다.



```

CREATE OR REPLACE FUNCTION sum_upto(n INT) RETURN INT
AS
 sum INT := 0;
 i INT := 1;
BEGIN
 WHILE i <= n LOOP
 sum := sum + i;
 i := i + 1;
 END LOOP;

 RETURN sum;
END;

```

다음은 For-Iterator Loop 구문의 사용 예를 보여준다.

```

CREATE OR REPLACE PROCEDURE mult_tables
AS
BEGIN
 FOR i IN 2 .. 9 LOOP
 DBMS_OUTPUT.put_line('table ' || i);

 FOR j IN 1 .. 9 LOOP
 DBMS_OUTPUT.put_line(i || ' x ' || j || ' = ' || i*j);
 END LOOP;

 DBMS_OUTPUT.put_line('');
 END LOOP;
END;

```

다음은 동일한 SELECT 문을 두 가지 다른 형태의 For Loop으로 조회하는 예를 보여준다.

```

CREATE OR REPLACE PROCEDURE athlete_history(p_name VARCHAR)
AS
 CURSOR my_cursor IS
 SELECT host_year, score
 FROM history
 WHERE athlete = p_name;
BEGIN
 -- For-Cursor Loop
 FOR r IN my_cursor LOOP
 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '
score: ' || r.score);
 END LOOP;

 -- For-Select Loop
 FOR r IN (SELECT host_year, score FROM history WHERE athlete =
p_name) LOOP
 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '

```

```
score: ' || r.score);
 END LOOP;
END;
```

## CASE 실행문

CASE 문은 여러 개의 조건을 순차적으로 검사해서 가장 처음 만족하는 조건에 연관되어 있는 실행문들을 실행한다.

```
<case_statement> ::=
 CASE <expression> { WHEN <expression> THEN
 <seq_of_statements> }... [ELSE <seq_of_statements>] END CASE
 | CASE { WHEN <expression> THEN <seq_of_statements> }... [ELSE
 <seq_of_statements>] END CASE
```

CASE 문은 두 가지 형태가 있다.

- 첫번째 형태는 CASE 키워드 직후에 표현식을 갖는다. 우선 이 최초 표현식을 계산한 다음, 이후 WHEN 절의 표현식을 하나씩 차례로 계산해서 최초 표현식과 일치하는 값을 찾고, 해당 THEN 절의 실행문들을 실행한다. 최초 표현식은 단 한번 계산된다.
- 두번째 형태는 CASE 키워드 직후에 표현식을 갖지 않는다. CASE 키워드 이후 여러 개의 WHEN 절의 표현식은 BOOLEAN 타입을 가져야 한다. 이들 표현식을 하나씩 차례로 계산하다가 처음으로 TRUE 값이 되는 표현식이 발견되면 해당 THEN 절의 실행문을 실행한다.

두 형태 모두 선택적으로 ELSE 절을 가질 수 있다. 이는 조건을 만족하는 WHEN 이후 표현식을 찾지 못했을 경우에 실행할 실행문들을 지정한다. 조건을 만족하는 WHEN 절이 없고 ELSE 절도 없을 때는 CASE\_NOT\_FOUND라는 시스템 Exception이 발생한다.

다음은 첫 번째 형태의 CASE 문 예제이다.

```
CREATE OR REPLACE PROCEDURE print_even_odd(i INTEGER)
AS
BEGIN
 CASE i MOD 2
 WHEN 0 THEN
 DBMS_OUTPUT.put_line('Even');
 WHEN 1 THEN
 DBMS_OUTPUT.put_line('Odd');
 ELSE
 DBMS_OUTPUT.put_line('Null');
 END CASE;
END;
```

다음은 유사한 동작을 하는 두 번째 형태의 CASE 문 예제이다.

```

CREATE OR REPLACE PROCEDURE print_even_odd(i INTEGER)
AS
BEGIN
 CASE
 WHEN i MOD 2 = 0 THEN
 DBMS_OUTPUT.put_line('Even');
 WHEN i MOD 2 = 1 THEN
 DBMS_OUTPUT.put_line('Odd');
 ELSE
 DBMS_OUTPUT.put_line('Null');
 END CASE;
END;

```

## 표현식

PL/CSQL의 표현식의 종류는 다음 문법으로 요약할 수 있다.

```

<expression> ::=
 <literal> # 상수
 | <identifier> # 식별자
 | SQL%ROWCOUNT # Static SQL 결과 크기
 | <cursor_expression> <cursor_attribute> # 커서 속성
 | <expression> <binary_op> <expression> # 이항 연산
 | <unary_op> <expression> # 단항 연산
 | (<expression>) # 괄호
 | <identifier>.<identifier> # 레코드 필드 참조
 | <identifier> <function_argument> # 함수 호출
 | <case_expression> # CASE 표현식
 | SQLCODE # Exception 코드
 | SQLERRM # Exception 메시지
 | <expression> IS [NOT] NULL # IS NULL 표현식
 | <expression> [NOT] BETWEEN <expression> AND
 <expression> # BETWEEN 표현식
 | <expression> [NOT] IN (<expression> [,
 <expression>, ...]) # IN 표현식
 | <expression> [NOT] LIKE <expression> [ESCAPE
 <expression>] # LIKE 표현식

<literal> ::=
 DATE <quoted_string>
 | TIME <quoted_string>
 | DATETIME <quoted_string>
 | TIMESTAMP <quoted_string>
 | <numeric>
 | <quoted_string>
 | NULL
 | TRUE
 | FALSE

```

```

<numeric> ::= UNSIGNED_INTEGER | FLOATING_POINT_NUM

<cursor_attribute> ::= { %ISOPEN | %FOUND | %NOTFOUND | %ROWCOUNT }

<binary_op> ::=
 AND | XOR | OR
 | = | <=> | != | <> | <= | >= | < | >
 | * | / | + | -
 | >> | << | & | ^ | '|'
 | ||

<unary_op> ::= + | - | NOT | ~

<case_expression> ::=
 CASE <expression> <case_expression_when_part>... [ELSE
<expression>] END
 | CASE <case_expression_when_part>... [ELSE <expression>] END

<case_expression_when_part> ::= WHEN <expression> THEN <expression>

```

## 리터럴

리터럴에는 날짜/시간, 숫자, 문자열, NULL, TRUE, FALSE 값이 있다. 비트열과 컬렉션을 사용할 수 없다는 점을 제외하고 리터럴 관련 규칙은 **SQL 리터럴**과 동일하다.

## 식별자

Static/Dynamic SQL 밖의 PL/CSQL 문에서 사용할 수 있는 식별자에는 다음 세 가지 종류가 있다.

- 선언부에서 선언된 변수, 상수, 커서, Exception, 내부 프로시저/함수
- 프로시저/함수의 인자
- 묵시적으로 선언된 **For 루프**의 iterator. integer 타입이거나 record 타입

명시적 혹은 묵시적 선언 없이 식별자를 사용하면 컴파일 에러가 발생한다.

## Static SQL 결과 크기

SQL%ROWCOUNT는 Static SQL을 실행한 직후에 결과 크기를 나타내는 표현식이다.

- 커서와 연관되지 않은 SELECT 문의 경우 반드시 INTO 절을 사용해야 하며 조회 결과는 1개이어야 한다. 따라서, 이 SELECT 문이 정상적으로 수행되었을 때 SQL%ROWCOUNT의 값은 1이다. 조회 결과 크기가 0이거나 1을 초과해서 실행시 에러가 발생했을 때는 SQL%ROWCOUNT의 값은 정의되지 않는다.
- INSERT, UPDATE, DELETE, MERGE, REPLACE, TRUNCATE 문의 경우 영향 받은 레코드 개수가 된다.

- COMMIT, ROLLBACK 문에 대해서는 0이 된다.

## 커서 속성

커서나 SYS\_REFCURSOR 변수를 계산 결과로 갖는 표현식 *cursor\_expression*에 %ISOPEN, %FOUND, %NOTFOUND, %ROWCOUNT 기호를 덧붙여 그 커서의 네 가지 속성을 조회할 수 있다.

- %ISOPEN: 커서가 열려 있는지 여부 (BOOLEAN)
- %FOUND: 첫 번째 FETCH 이전이면 NULL. 아니면 마지막 FETCH가 1개의 ROW를 결과로 갖는지 여부 (BOOLEAN). 열려 있지 않은 커서에 대해서 조회하면 INVALID\_CURSOR Exception 발생.
- %NOTFOUND: 첫 번째 FETCH 이전이면 NULL. 아니면 마지막 FETCH가 0개의 ROW를 결과로 갖는지 여부 (BOOLEAN). 열려 있지 않은 커서에 대해서 조회하면 INVALID\_CURSOR Exception 발생.
- %ROWCOUNT: 첫 번째 FETCH 이전이면 0. 아니면 현재까지 FETCH된 ROW의 개수 (BIGINT). 열려 있지 않은 커서에 대해서 조회하면 INVALID\_CURSOR Exception 발생.

아래 예제에서 내부 함수 `iterate_cursor()`는 커서 속성을 사용하여 레코드들을 순회하고 전체 레코드 개수를 리턴한다. 인자로 넘겨 받은 커서가 열려 있지 않을 때는 (커서의 %ISOPEN 속성이 False 일 때는) -1을 리턴한다. 더 이상 조회할 레코드가 없는지는 FETCH 후 커서의 %NOTFOUND 속성을 검사해서 알아낸다. 커서의 %ROWCOUNT 속성은 FETCH 문으로 조회된 레코드 각각마다 1씩 증가하다가 FETCH 반복문이 종료된 후에는 조회된 전체 레코드 개수를 나타내게 된다.

```
CREATE OR REPLACE PROCEDURE cursor_attributes AS
...

FUNCTION iterate_cursor(rc SYS_REFCURSOR) RETURN INT
AS
 v VARCHAR;
BEGIN
 IF rc%ISOPEN THEN
 LOOP
 FETCH rc INTO v;
 EXIT WHEN rc%NOTFOUND;

 -- do something with v
 ...

 END LOOP;

 RETURN rc%ROWCOUNT; -- number of records
 ELSE
 RETURN -1; -- error
 END IF;
END;
begin
...

```

```
end;
```

`cursor_expression`의 계산 결과가 NULL이면 `INVALID_CURSOR Exception`이 발생한다.

## 이항 연산, 단항 연산, 괄호

PL/SQL은 다음과 같이 연산자 우선 순위를 갖는다.

| 연산자                             | 연산                    |
|---------------------------------|-----------------------|
| <code>+, -, ~</code>            | 부호, 비트역 (단항)          |
| <code>*, /, DIV, MOD</code>     | 곱하기, 나누기, 정수 나누기, 나머지 |
| <code>+, -</code>               | 더하기, 빼기               |
| <code>  </code>                 | 문자열 병합                |
| <code>&lt;&lt;, &gt;&gt;</code> | 비트 이동                 |
| <code>&amp;</code>              | 비트곱                   |
| <code>^</code>                  | 배타적 비트합               |
| <code> </code>                  | 비트합                   |
| <code>IS NULL</code>            | NULL 테스트              |
| <code>LIKE</code>               | 문자열 패턴 테스트            |
| <code>BETWEEN</code>            | 값 범위 테스트              |
| <code>IN</code>                 | 값 포함 테스트              |

| 연산자                           | 연산      |
|-------------------------------|---------|
| =, <=>, <, >, <=, >=, <>, !=, | 비교      |
| NOT                           | 논리역     |
| AND                           | 논리곱     |
| XOR                           | 배타적 논리합 |
| OR                            | 논리합     |

- %는 Static/Dynamic SQL 밖에서는 MOD와 동일한 의미의 나머지 연산자로 사용할 수 없다.
- &&, ||, !은 Static/Dynamic SQL 밖에서는 AND, OR, NOT과 동일한 의미의 논리 연산자로 사용할 수 없다.
- Static/Dynamic SQL 밖에서의 문자열은 DB 설정과 상관없이 UTF8 encoding을 따르며 이들 문자열들 사이의 비교는 해당 Unicode 배열들 사이의 사전식 비교법을 따른다. Static/Dynamic SQL 안에서의 문자열의 encoding과 비교는 DB와 테이블 설정을 따른다.

명시적으로 연산 순서를 지정하기 위해 괄호를 사용할 수 있다.

## 레코드 필드 참조

PL/CSQL에서는 다음 두 가지 경우에 레코드 변수를 사용할 수 있다.

- FOR 문에서 SELECT 결과를 순회하기 위해 묵시적으로 선언되는 레코드 변수
- %ROWTYPE으로 선언된 레코드 변수

레코드 변수 뒤에 .(dot)과 필드 이름을 덧붙여 레코드 필드를 참조할 수 있다.

```
CREATE PROCEDURE athlete_history(p_name VARCHAR)
AS
 CURSOR my_cursor IS
 SELECT host_year, score
 FROM history
 WHERE athlete = p_name;
BEGIN
 FOR r IN my_cursor LOOP -- r: 묵시적으로 선언됨
```

```

 DBMS_OUTPUT.put_line('host_year: ' || r.host_year || '
score: ' || r.score); -- r.<column-name>
 END LOOP;
END;
```

## 함수 호출

함수 호출 표현식에서 인자 개수와 각각의 타입은 해당 함수의 선언과 일치해야 한다. 호출되는 함수의 OUT 인자에 주어질 인자들은 호출 결과 변경이 일어나게 되므로 대입이 가능한 변수나 다른 OUT 인자이어야 한다.

```

CREATE OR REPLACE FUNCTION callee(o OUT INT) RETURN INT
AS
BEGIN
 ...
END;

CREATE OR REPLACE PROCEDURE proc(i INT, o OUT INT)
AS
 v INT;
 c CONSTANT INT := 0;
BEGIN
 ... callee(i) ... -- Error: IN 인자
 ... callee(o) ... -- OK: OUT 인자
 ... callee(v) ... -- OK: 변수
 ... callee(c) ... -- Error: 상수
END;
```

호출되는 함수는 저장 함수, 내부 함수, 빌트인 함수 이렇게 세 가지 종류이다. 이 중에서 빌트인 함수는 **연산자와 함수** 장에 나열된 큐브리드 내장 함수들을 말한다. 단, 빌트인 함수들 중에서 **IF**는 PL/CSQL 문법과 충돌을 일으키므로 사용할 수 없다.

다른 저장 함수나 빌트인 함수 호출문 실행 중에 에러가 나면 SQL\_ERROR Exception이 발생한다.

## CASE 표현식

CASE 표현식은 여러 개의 조건을 순차적으로 검사해서 가장 처음 만족하는 조건에 연관되어 있는 값을 갖는다.

CASE 표현식은 **CASE 실행문**(Statement)과 마찬가지로 CASE 키워드 직후에 표현식을 갖는 형태와 갖지 않는 형태가 있다.

- CASE 키워드 직후에 표현식을 갖는 형태에서는 우선 이 최초 표현식을 계산한 다음, WHEN 절들의 표현식을 하나씩 차례로 계산해서 최초 표현식과 일치하는 값을 찾고, 해당 THEN 절의 표현식을 계산해서 CASE문의 최종값으로 한다. 최초 표현식은 단 한번 계산된다.



- CASE 키워드 직후에 표현식을 갖지 않는 형태에서는 CASE 키워드 이후 여러 개의 WHEN 절의 표현식은 BOOLEAN 타입을 가져야 한다. 이들 표현식을 하나씩 차례로 계산하다가 처음으로 TRUE 값이 되는 표현식이 발견되면 해당 THEN 절의 표현식을 계산해서 CASE문의 최종값으로 한다.

두 형태 모두 선택적으로 ELSE 절을 가질 수 있다. 이는 조건을 만족하는 WHEN 이후 표현식을 찾지 못했을 경우에 값으로 가질 표현식을 지정한다. 조건을 만족하는 WHEN 절이 없고 ELSE 절도 없을 때 전체 CASE 표현식은 NULL 값을 갖는다.

다음은 첫 번째 형태의 CASE 표현식 예제이다.

```
CREATE OR REPLACE PROCEDURE print_even_odd(i INTEGER)
AS
 s VARCHAR(5);
BEGIN
 s := CASE i MOD 2
 WHEN 0 THEN 'Even'
 WHEN 1 THEN 'Odd'
 ELSE 'NULL'
 END;

 DBMS_OUTPUT.put_line(s);
END;
```

다음은 유사한 동작을 하는 두 번째 형태의 CASE 표현식 예제이다.

```
CREATE OR REPLACE PROCEDURE print_even_odd(i INTEGER)
AS
 s VARCHAR(5);
BEGIN
 s := CASE
 WHEN i MOD 2 = 0 THEN 'Even'
 WHEN i MOD 2 = 1 THEN 'Odd'
 ELSE 'NULL'
 END;

 DBMS_OUTPUT.put_line(s);
END;
```

## SQLCODE, SQLERRM

Exception 처리 block 안에서 SQLCODE와 SQLERRM은 각각 현재 처리 중인 Exception의 코드 (INTEGER 타입)와 에러메시지 (STRING 타입)를 나타낸다. Exception 처리 block 밖에서 SQLCODE와 SQLERRM은 각각 0과 'no error' 값을 갖는다.

```
CREATE OR REPLACE PROCEDURE test_sql_code_errm
AS
BEGIN
 ...
EXCEPTION
 WHEN OTHERS THEN
 DBMS_OUTPUT.put_line('code=' || SQLCODE);
 DBMS_OUTPUT.put_line('error message' || SQLERRM);
END;
```

## 자바 저장 프로시저

CUBRID는 Java로 저장 함수와 프로시저를 개발할 수 있도록 지원한다. Java 저장 함수와 프로시저는 CUBRID에서 호스팅한 Java 가상 머신(JVM, Java Virtual Machine)에서 실행된다. Java 저장 함수/프로시저는 Java 가상머신을 사용하므로 뛰어난 상호 운용성과 이식성을 갖는다.

## 자바 저장 함수/프로시저 작성과 사용을 위한 확인

Java 저장함수/프로시저를 사용하기 위해서 프로시저 언어 서버가 준비되어 있어야 한다. 프로시저 언어 서버는 데이터베이스 서버 구동 시 자동으로 시작된다. 자세한 내용은 [CUBRID 프로시저 언어 서버](#) 와 [프로시저 언어 서버 설정](#) 을 참고한다.

## 자바 저장 함수/프로시저 작성과 로드

자바 저장 함수/프로시저를 사용하기 위해서는 다음의 순서에 따라 자바 저장 함수/프로시저를 작성하고 등록한다.

- Step 1: Java 소스 작성
- Step 2: Java 소스 컴파일
- Step 3: Java 클래스 로드
- Step 4: 저장 함수/프로시저 등록

### Java 소스 작성

다음은 Java 저장 함수/프로시저를 작성하는 예이다. 구현하는 Java 클래스의 메서드는 반드시 **public static**이어야 한다.

```
public class SpCubrid{
 public static String HelloCubrid() {
```

```

 return "Hello, Cubrid !!";
 }

 public static int SpInt(int i) {
 return i + 1;
 }

 public static void outTest(String[] o) {
 o[0] = "Hello, CUBRID";
 }
}

```

Java 저장 함수/프로시저에서 데이터베이스에 접근하기 위해서는 서버 측 JDBC 드라이버를 사용해야 한다. **서버 내부 JDBC 드라이버 사용**을 참조하여 서버 측 JDBC 드라이버 사용한다.

### Java 소스 컴파일

다음과 같이 SpCubrid.java를 컴파일 한다.

```
javac SpCubrid.java
```

서버 측 JDBC 드라이버를 사용하는 경우 **classpath(cp)** 옵션을 사용하여 JDBC의 경로를 지정하여 다음과 같이 컴파일 해야한다. 이때 로드 할 데이터베이스 서버의 최신 JDBC 드라이버를 사용해야 한다.

```
javac SpCubrid.java -cp $CUBRID/jdbc/cubrid_jdbc.jar
```

### Java 클래스 로드

컴파일된 Java 클래스를 loadjava 유틸리티를 사용해 CUBRID로 로딩한다. **loadjava 유틸리티**를 참조하여 사용한다.

```
% loadjava demodb SpCubrid.class
```

### 저장 함수/프로시저 등록

CUBRID는 SQL 문이나 응용 프로그램에서 Java 메서드를 호출할 수 있도록 Java 클래스를 등록 (publish)하는 과정이 필요하다. 다음과 같이 CUBRID 저장 함수를 생성하여 Java 클래스를 등록한다.

```

...
AS LANGUAGE JAVA NAME '<java_call_specification>';

```

위와 같이 JAVA NAME 이하 문자열에 Java 클래스의 메서드를 호출하기 위한 클래스와 메서드 이름을 지정한다. 이것을 **Java Call Specification** 라고 한다.

```
CREATE FUNCTION hello() RETURN STRING
AS LANGUAGE JAVA
NAME 'SpCubrid.HelloCubrid() return java.lang.String';
```

## Java Call Specification

Java 클래스가 로딩될 때 SQL 문이나 응용 프로그램에서 클래스 내의 함수를 어떻게 호출할지 모르기 때문에 Call Specification을 사용하여 Java 클래스를 등록해야 한다.

Call Specification을 사용하면 SQL 문이나 응용 프로그램에서 Java 함수 이름, 인자 타입, 리턴 값 및 리턴 값의 타입에 접근할 수 있다. Call Specification을 작성하려면 **CREATE FUNCTION** 또는 **CREATE PROCEDURE** 문을 사용한다.

- Java 저장 함수/프로시저 이름은 대소문자를 구분하지 않는다.
- Java 저장 함수/프로시저 이름은 최대 254바이트까지 가능하다.
- Java 저장 함수/프로시저는 최대 64개의 인자를 가질 수 있다.
- Java 저장 함수/프로시저의 인자를 **OUT** 으로 설정한 경우 길이가 1인 1차원 배열로 전달된다. 그러므로 Java 메서드는 배열의 첫번째 공간에 전달할 값을 저장해야 한다.

```
CREATE PROCEDURE test_out(x OUT STRING)
AS LANGUAGE JAVA
NAME 'SpCubrid.outTest(java.lang.String[] o)';
```

## Java 저장 함수/프로시저 호출

Java 저장 함수/프로시저가 등록되고 나면 **PL/CSQL** 과의 차이점 없이 동일하게 호출할 수 있다. 자세한 내용은 **저장 프로시저의 호출**을 참고한다.

Java 저장 함수/프로시저를 호출하여 수행 중 exception이 발생하면 `dbname_pl.log` 파일에 exception 내용이 기록되어 저장된다. 만약 화면으로 exception 내용을 확인하고자 할 경우는 **\$CUBRID/java/logging.properties** 파일의 handlers 값을 "java.lang.logging.ConsoleHandler"로 수정하면 화면으로 exception 내용을 출력한다.

## Java 저장 프로시저 예시

응용 프로그램에서 저장 함수/프로시저를 호출하기 위해서는 **CallableStatement** 를 사용한다.

CUBRID 데이터베이스에 Phone 클래스를 생성한다.

```
CREATE TABLE phone(
 name VARCHAR(20),
 phoneno VARCHAR(20)
);
```

다음의 PhoneNumber.java Java 파일을 컴파일하여 Java 클래스 파일을 CUBRID로 로드하고 등록한다.

```
import java.sql.*;
import java.io.*;

public class PhoneNumber{
 public static void Phone(String name, String phoneno) throws
Exception {
 String sql="INSERT INTO PHONE(NAME, PHONENO)"+"VALUES
(?, ?)";
 try{
 Connection conn =
DriverManager.getConnection("jdbc:default:connection:");
 PreparedStatement pstmt = conn.prepareStatement(sql);

 pstmt.setString(1, name);
 pstmt.setString(2, phoneno);
 pstmt.executeUpdate();

 pstmt.close();
 conn.commit();
 conn.close();
 } catch (SQLException e) {
 System.err.println(e.getMessage());
 }
}
}
```

```
create PROCEDURE phone_info(name varchar, phoneno varchar) as
language java
name 'PhoneNumber.Phone(java.lang.String, java.lang.String)';
```

다음과 같은 응용 프로그램을 작성하고 실행한다.

```
import java.sql.*;

public class StoredJDBC {
 public static void main() {
 Connection conn = null;
 Statement stmt= null;
 int result;
```

```

 int i;

 try{
 conn =
DriverManager.getConnection("jdbc:CUBRID:localhost:
33000:demodb::", "", "");

 CallableStatement cs;
 cs = conn.prepareCall("CALL PHONE_INFO(?, ?)");

 cs.setString(1, "Jane");
 cs.setString(2, "010-1111-1111");
 cs.executeUpdate();

 conn.commit();
 cs.close();
 conn.close();
 } catch (Exception e) {
 e.printStackTrace();
 }
 }
}

```

위의 프로그램 실행한 후 PHONE 클래스 조회를 하면 다음과 같은 결과가 출력된다.

```
SELECT * FROM phone;
```

```

name phoneno
=====
'Jane' '010-111-1111'

```

## 서버 내부 JDBC 드라이버 사용

Java 저장 함수/프로시저에서 데이터베이스에 접근하기 위해서는 서버 측 JDBC 드라이버(Server-Side JDBC Driver)를 사용해야 한다. 서버 측 JDBC 드라이버로 다음의 기능이 가능하다.

- 질의문 수행
- 질의 결과셋 처리

다음은 서버측 JDBC 드라이버에서 지원하는 클래스이다. JDBC API 지원에 대한 자세한 내용은 [부록](#)을 참고한다.

- **java.sql.Connection**
- **java.sql.Statement**

- `java.sql.PreparedStatement`
- `java.sql.CallableStatement`
- `java.sql.ResultSet`
- `java.sql.ResultSetMetaData`

#### ⚠ 경고

`java.sql.DatabaseMetaData` 의 기능은 지원하지 않는다.

서버측 JDBC 드라이버를 사용하는 데이터베이스 연산은 다음의 특징을 가진다.

- 수행되는 데이터베이스 연산은 Java 저장 함수/프로시저를 호출한 트랜잭션에 포함된다
- 트랜잭션 관련 API는 무시된다.
- 서버 측 JDBC 드라이버의 연결을 다시 설정할 필요가 없다

### Connection 생성

데이터베이스에 접근하기 위해서 서버 측 JDBC Connection을 생성해야한다. 서버 측 JDBC 드라이버로 해당 데이터베이스의 Connection을 얻는 방법은 아래와 같이 JDBC 연결 URL로 "`jdbc:default:connection:`" 을 사용한다.

```
Connection conn =
DriverManager.getConnection("jdbc:default:connection:");
```

#### i 참고

CUBRID 11.2 버전부터 다음과 같은 방법으로 Connection을 얻는 방법은 더이상 지원하지 않는다.

```
Connection conn = cubrid.jdbc.driver.CUBRIDDriver.getDefaultConnection();
```

#### i 참고

서버 측 JDBC 드라이버는 이미 등록되어 있기 때문에  
`Class.forName("cubrid.jdbc.driver.CUBRIDDriver")`를 호출하지 않아도 된다

## 질의문 수행

자바 저장/프로시저를 구현할 때 자바 어플리케이션을 개발하는 것과 동일하게 다음의 JDBC 인터페이스를 이용하여 질의문을 수행할 수 있다.

- `java.sql.Statement`
- `java.sql.PreparedStatement`
- `java.sql.CallableStatement`

다음은 위의 클래스를 사용하여 수행할 수 있는 질의문이다.

- **DML (Data Manipulation Language): 데이터 조작문**
- **DDL (Data Definition Language): 데이터 정의문**

### 참고

질의를 수행할 때 생성하는 JDBC 객체는 하나의 SQL 구문만 포함해야한다. 따라서 다음의 경우에 에러가 발생한다.

```
stmt = new Statement ("select * from t1;select * from t2;");
```

다음의 구문에 해당하는 기능은 지원하지 않는다.

- **TCL (Transaction Control Language): 데이터베이스 트랜잭션**

### 참고

- **COMMIT, ROLLBACK** 구문에 해당하는 함수인 `commit()`, `rollback()`은 무시된다.
- **SAVEPOINT** 구문에 해당하는 함수는 지원하지 않는다.



## 질의문 수행 예시

### 결과셋을 반환하는 질의 수행과 질의 결과셋 처리

다음 예시는 결과셋을 반환하는 **SELECT** 문을 실행하는 방법이다. **SELECT** 문은 **java.sql.Statement** 또는 **java.sql.PreparedStatement** 객체를 생성하여 수행할 수 있다. 수행한 질의 결과셋 (**java.sql.ResultSet**) 을 사용하여 수행한 질의의 결과를 처리할 수 있다.

#### 참고

- **java.sql.ResultSet**은 forward-only, read-only 이다.
- 클라이언트 측 JDBC 드라이버의 경우 질의 결과셋을 생성하면 기본적으로 **커서 유지(cursor holdability)**를 한다. 서버 측 JDBC 드라이버에서는 자원을 서버에서 관리하므로 질의 결과셋은 커서를 유지하지 않고 저장 함수/프로시저 종료 시에 내부적으로 정리한다.

또한 질의 결과셋으로부터 **getMetaData()** 함수를 이용하여 결과셋 메타 데이터 (**java.sql.ResultSetMetaData**)를 생성할 수 있다.

```
CREATE OR REPLACE FUNCTION sp_get_athlete_by_ncode (nc STRING)
RETURN STRING as language java name
'TestQuery.printAthelete(java.lang.String) return java.lang.String';
```

```
import java.sql.*;

public class TestQuery {
 public static String printAthelete(String nation_code_filter)
 throws SQLException {
 String sql = "SELECT * FROM public.athlete WHERE nation_code = ?";

 StringBuilder builder = new StringBuilder();
 Connection conn = null;
 PreparedStatement pstmt = null;

 try {
 conn =
 DriverManager.getConnection("jdbc:default:connection:");
 pstmt = conn.prepareStatement(sql);

 pstmt.setString(1, nation_code_filter);

 ResultSet rs = pstmt.executeQuery();
 ResultSetMetaData rsmd = rs.getMetaData();

 builder.append("<Column Details>:\n");
```

```

 int colCount = rsmd.getColumnCount();
 for (int i = 1; i <= colCount; i++) {
 String colName = rsmd.getColumnName(i);
 String colType = rsmd.getColumnTypeName(i);
 builder.append(colName + "," + colType);

 if (i != colCount) builder.append("|");
 }

 builder.append("\n<Rows>:\n");
 while (rs.next()) {
 for (int i = 1; i <= rsmd.getColumnCount(); i++) {
 Object object = rs.getObject(i);
 if (object != null) {
 readColumn(i, rsmd, rs, builder);

 if (i != rsmd.getColumnCount()) builder.append
("|");
 }
 builder.append("\n");
 }

 rs.close();
 } catch (Exception e) {
 builder.append(e.getMessage());
 } finally {
 if (pstmt != null) pstmt.close();
 if (conn != null) conn.close();
 }

 return builder.toString();
 }

 private static void readColumn(int idx, ResultSetMetaData rsmd,
 ResultSet rs, StringBuilder stringBuilder) throws SQLException {
 switch (rsmd.getColumnType(idx)) {
 case java.sql.Types.DOUBLE:
 stringBuilder.append(rs.getDouble(idx));
 break;
 case java.sql.Types.FLOAT:
 stringBuilder.append(rs.getFloat(idx));
 break;
 case java.sql.Types.VARCHAR:
 stringBuilder.append("\"").append(rs.getString(idx)).append("\"");
 break;
 case java.sql.Types.INTEGER:
 case java.sql.Types.TINYINT:
 case java.sql.Types.SMALLINT:
 case java.sql.Types.BIGINT:
 stringBuilder.append(rs.getInt(idx));

```

```

 break;
 case java.sql.Types.DATE:

 stringBuilder.append("\").append(rs.getDate(idx)).append("\");
 break;
 case java.sql.Types.TIMESTAMP:

 stringBuilder.append("\").append(rs.getTimestamp(idx)).append("\");
 break;
 default:
 stringBuilder.append(rs.getObject(idx));
 break;
 }
}
}

```

```

SELECT sp_get_athlete_by_ncode ('ESP');

sp_get_athlete_by_ncode('ESP')
=====
'<Column Details>:
code,INTEGER|name,VARCHAR|gender,CHAR|nation_code,CHAR|event, VARCHAR
<Rows>:
10999|"Fernandez Jesus"|M|ESP|"Handball"
10997|"Fernandez Isabel"|W|ESP|"Judo"
10994|"Fernandez Abelardo"|M|ESP|"Football"
10948|"Etxaburu Aitor"|M|ESP|"Handball"
10941|"Estiarte Manuel"|M|ESP|"Water Polo"
...

```

## INSERT, UPDATE, DELETE

다음은 **INSERT** 문을 수행하는 예시이다. **INSERT, UPDATE, DELETE** 문은 **executeUpdate()** 함수를 통해 수행한다.

```

import java.sql.*;

public class Athlete {
 public static void insertAthlete(String name, String gender,
String nation_code, String event) throws SQLException {
 String sql = "INSERT INTO ATHLETE(NAME, GENDER, NATION_CODE,
EVENT)" + "VALUES (?, ?, ?, ?)";

 Connection conn = null;
 PreparedStatement pstmt = null;

 try{
 conn =
DriverManager.getConnection("jdbc:default:connection:");

```

```

 pstmt = conn.prepareStatement(sql);

 pstmt.setString(1, name);
 pstmt.setString(2, gender);
 pstmt.setString(3, nation_code);
 pstmt.setString(4, event);
 pstmt.executeUpdate();

 pstmt.close();
 conn.commit();
 conn.close();
 } catch (Exception e) {
 System.err.println(e.getMessage());
 } finally {
 if (pstmt != null) pstmt.close();
 if (conn != null) conn.close();
 }
}

```

#### 참고

위의 Athlete 클래스 예시에서 conn.commit()은 무시된다.

## OUT, IN/OUT 정의

CUBRID의 Java 저장 함수/프로시저에서 Java에서 인자 값을 변경할 경우 변경 값이 전달이 되도록 인자가 OUT 인자로 전달될 때는 1차원 배열로 전달해야 한다.

```

CREATE PROCEDURE sp_increment_me(x IN OUT INT) AS LANGUAGE JAVA NAME
'OutTest.incrementInt(int[])';

```

```

public class OutTest {
 public static void incrementInt(int[] arg) {
 arg[0] = arg[0] + 1;
 }
}

```

## Set 타입의 IN/OUT 정의

CUBRID의 Java 저장 함수/프로시저에서 Set 타입이 IN OUT인 경우 Java에서 인자 값을 변경할 경우 변경 값이 전달이 되도록 Set 타입이 OUT 인자로 전달될 때는 2차원 배열로 전달해야 한다.

```
CREATE PROCEDURE setoid(x in out set, z object) AS LANGUAGE JAVA
NAME 'SetOIDTest.SetOID(cubrid.sql.CUBRIDOID[][],
cubrid.sql.CUBRIDOID');
```

```
import cubrid.sql.CUBRIDOID;

public class SetOIDTest {
 public static void SetOID(CUBRIDOID[][] set, CUBRIDOID aoid) {
 String ret="";
 Vector v = new Vector();

 CUBRIDOID[] set1 = set[0];

 try {
 if(set1 != null) {
 int len = set1.length;
 int i = 0;

 for (i = 0; i < len; i++)
 v.add(set1[i]);
 }

 v.add(aoid);
 set[0] = (CUBRIDOID[]) v.toArray(new CUBRIDOID[]{});
 } catch(Exception e) {
 e.printStackTrace();
 System.err.println("SQLException:"+e.getMessage());
 }
 }
}
```

Java 저장 함수/프로시저에서 OID 사용

CUBRID 저장 프로시저에서 OID 타입의 값을 IN/OUT으로 사용할 경우 서버의 값을 전달 받아 사용한다.

```
CREATE PROCEDURE tOID(i inout object, q string) AS LANGUAGE JAVA
NAME 'OIDtest.tOID(cubrid.sql.CUBRIDOID[], java.lang.String)';
```

```
import java.sql.*;
import cubrid.sql.CUBRIDOID;

public class OIDTest {
 public static void tOID(CUBRIDOID[] oid, String query)
 {
```

```

Connection conn = null;
Statement stmt = null;
String ret = "";

try {
 conn =
DriverManager.getConnection("jdbc:default:connection:");

 conn.setAutoCommit(false);
 stmt = conn.createStatement();
 ResultSet rs = stmt.executeQuery(query);
 System.out.println("query:" + query);

 while(rs.next()) {
 oid[0] = (CUBRIDOID) rs.getObject(1);
 System.out.println("oid:" + oid[0].getTableName());
 }

 stmt.close();
 conn.close();

} catch (SQLException e1) {
 e1.printStackTrace();
 System.err.println("SQLException:" + e1.getMessage());
} catch (Exception e2) {
 e2.printStackTrace();
 system.err.println("Exception:" + e2.getMessage());
}
}
}

```

## 질의 결과셋 반환

CUBRID에서는 질의 결과셋 (**java.sql.ResultSet**)을 반환할 수 있고, 선언 시 반환하는 데이터 타입으로 **CURSOR**를 사용한다.

### 참고

- **java.sql.ResultSet** 은 함수의 입력 인자로 사용할 수 없으며, 이를 IN 인자로 전달할 경우에는 에러가 발생한다.
- Java가 아닌 환경에서 **ResultSet** 을 반환하는 함수를 호출할 경우에도 에러가 발생한다.

```
CREATE FUNCTION rset() RETURN CURSOR AS LANGUAGE JAVA
NAME 'JavaSP2.TResultSet() return java.sql.ResultSet'
```

```
import java.sql.*;

public class JavaSP2 {
 public static ResultSet TResultSet(){
 try {
 Connection conn =
DriverManager.getConnection("jdbc:default:connection:");

 String sql = "select * from station";
 Statement stmt=conn.createStatement();
 ResultSet rs = stmt.executeQuery(sql);

 return rs;
 } catch (Exception e) {
 e.printStackTrace();
 }

 return null;
 }
}
```

호출하는 쪽에서는 **Types.JAVA\_OBJECT** 로 OUT 인자를 설정하고 **getObject ()** 함수로 가져온 후 **java.sql.ResultSet** 으로 변환(Casting)하여 사용해야 한다. 또한, **java.sql.ResultSet** 은 JDBC의 **CallableStatement** 에서만 사용할 수 있다.

```
import java.sql.*;

public class TestResultSet{
 public static void main(String[] args) {
 Connection conn = null;

 try {
 conn =
DriverManager.getConnection("jdbc:CUBRID:localhost:
33000:demodb::", "", "");

 CallableStatement cstmt = conn.prepareCall("=?=CALL
rset()");
 cstmt.registerOutParameter(1, Types.JAVA_OBJECT);
 cstmt.execute();
 ResultSet rs = (ResultSet) cstmt.getObject(1);

 while(rs.next()) {
 System.out.println(rs.getString(1));
 }
 }
 }
}
```

```

 rs.close();
 } catch (Exception e) {
 e.printStackTrace();
 }
}
}

```

## 연결 중인 클라이언트 정보 획득

```

CREATE OR REPLACE FUNCTION sp_client_info () RETURN STRING as
language java name 'SpTestClientInfo.getClientInfo()' return
java.lang.String';

```

```

import java.util.Properties;
import java.sql.*;

public class SpTestClientInfo {
 public static String getClientInfo() {
 Connection conn = null;
 String result = "";

 try {
 conn =
DriverManager.getConnection("jdbc:default:connection:");

 Properties props = conn.getClientInfo();

 // How to get from the Properties
 // String user = props.getProperty ("user");

 result = props.toString ();
 } catch (Exception e) {
 result = e.getMessage ();
 }
 return result;
 }
}

```

```

SELECT sp_client_info ();

sp_client_info()
=====

```



```
'{pid=200270, user=DBA, login=cubrid, program=csql, type=2,
host=cubrid, ip=192.168.2.201}'
```

## 다른 데이터베이스 연결

서버 측 JDBC 드라이버를 사용하더라도 현재 연결된 데이터베이스를 사용하지 않고, 외부의 다른 데이터베이스에 연결할 수도 있다. 외부의 데이터베이스에 대한 Connection을 얻는 것은 일반적인 JDBC Connection과 다르지 않다. 이에 대한 자세한 내용은 JDBC API를 참조한다.

### ⚠ 경고

다른 데이터베이스에 연결하는 경우, Java 메서드의 수행이 종료되더라도 CUBRID 데이터베이스와의 Connection이 자동으로 종료되지 않는다. 따라서, 반드시 Connection 종료를 명시해주어야 **COMMIT, ROLLBACK** 과 같은 트랜잭션 연산이 해당 데이터베이스에 반영된다. 즉, Java 저장 함수/프로시저를 호출한 데이터베이스와 실제 연결된 데이터베이스가 다르기 때문에 **별도의 트랜잭션**으로 수행되는 것이다.

```
import java.sql.*;

public class SelectData {
 public static void SearchSubway(String[] args) throws Exception {
 Connection conn = null;
 Statement stmt = null;
 ResultSet rs = null;

 try {
 conn =
DriverManager.getConnection("jdbc:CUBRID:localhost:
33000:demodb::", "", "");

 String sql = "select line_id, line from line";
 stmt = conn.createStatement();
 rs = stmt.executeQuery(sql);

 while(rs.next()) {
 int host_year = rs.getString("host_year");
 String host_nation = rs.getString("host_nation");

 System.out.println("Host Year ==> " + host_year);
 System.out.println(" Host Nation==> " + host_nation);
 System.out.println("\n=====\n");
 }

 rs.close();
 } catch (SQLException e1) {
```

```

 System.err.println(e1.getMessage());
 } catch (Exception e2) {
 System.err.println(e2.getMessage());
 } finally {
 if (stmt != null) stmt.close();
 if (conn != null) conn.close();
 }
}
}

```

수행 중인 Java 저장 함수/프로시저가 데이터베이스 서버의 JVM에서만 구동되어야 할 때, Java 프로그램 소스에서 System.getProperty("cubrid.server.version")를 호출함으로써 어디서 수행되는 지를 점검할 수 있다. 결과 값은 데이터베이스에서 호출하면 데이터베이스 버전이 되고, 그 외는 **NULL** 이 된다.

## Java Native Interface (JNI) 지원

Java Native Interface (JNI)를 사용하여 JVM에서 C/C++ 와 같은 네이티브 언어의 함수를 호출할 수 있다. CUBRID의 Java SP에서는 JNI 기능을 사용할 수 있도록 제공하고 있지만 네이티브 코드의 문제가 저장 프로시저 서버 (cub\_pl) 프로세스 및 그 동작에 예상하지 못한 영향을 줄 수 있으므로 주의해서 사용해야 한다. 네이티브 라이브러리를 로드하는 클래스는 동적으로 로드되지 않도록 loadjava의 **-j** 옵션을 사용하여 등록한다. 자세한 내용은 **loadjava 유틸리티**를 참조한다.

다음은 Java 저장 함수에서 JNI를 통해 네이티브 함수를 호출하는 예제이다.

HelloJNI.h

```

#include <jni.h>

#ifndef _Included_HelloJNI
#define _Included_HelloJNI
#ifdef __cplusplus
extern "C" {
#endif

/*
 * Class: HelloJNI
 * Method: sayHello
 * Signature: ()V
 */
JNIEXPORT jstring JNICALL Java_HelloJNI_sayHello(JNIEnv *,
jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif

```

HelloJNI.c

```

#include <jni.h>
#include <stdio.h>
#include <string.h>
#include "HelloJNI.h"

// Implementation of native method sayHello() of HelloJNI class
JNIEXPORT jstring JNICALL Java_HelloJNI_sayHello(JNIEnv *env,
 jobject thisObj, jstring javaString) {
 const char *nativeString = (*env)->GetStringUTFChars(env,
 javaString, 0);
 // printf("Java_HelloJNI_sayHello : %sn", nativeString);
 const char *greeting = " Hello!";
 char cap[1024];
 strcpy(cap, nativeString);
 strcat(cap, greeting);
 (*env)->ReleaseStringUTFChars(env, javaString, nativeString);
 return (*env)->NewStringUTF(env, cap);
}

```

HelloJNI.java

```

import java.io.File;

public class HelloJNI {
 static {
 try {
 String cubridPath = System.getenv("CUBRID"); // get
 $CUBRID
 System.load(
 cubridPath
 + File.separator
 + "jni"
 + File.separator
 + "libhello.so"); // $CUBRID/jni/libhello.so
 } catch(UnsatisfiedLinkError e) {
 e.printStackTrace();
 }
 }

 // Declare
 private native String sayHello(String string);

 // CUBRID
 public static String cubridSayHello(String string) {
 return new HelloJNI().sayHello(string); // invoke the
 native method
 }
}

```

```
}
}
```

```
-- compile and copy HelloJNI.c
gcc -fPIC -I${JAVA_HOME}/include -I${JAVA_HOME}/include/linux -
shared -o libhello.so HelloJNI.c
mkdir -p $CUBRID/jni
cp libhello.so $CUBRID/jni

-- loadjava
javac HelloJNI.java
loadjava -j demodb HelloJNI.class
```

```
CREATE FUNCTION hello(str VARCHAR) RETURN VARCHAR AS LANGUAGE JAVA
NAME 'HelloJNI.cubridSayHello(java.lang.String) return
java.lang.String';

SELECT hello ('CUBRID');
```

```
hello('CUBRID')
=====
'CUBRID Hello!'
```

### ⚠ 경고

JNI를 호출하는 자바 저장 프로시저/함수를 **-j** 또는 **-jni** 옵션 없이 등록한 후 실행 시 다음과 같은 오류를 반환한다. 'Library load not allowed. Please load your class by using 'loadjava' with '-jni' option'

다음의 사항을 확인한다.

- 동일한 네이티브 라이브러리 경로에 대해 **System.load ()** 를 호출하는 자바 클래스 파일을 여러 개 로드하는 경우
  - 하나의 자바 클래스 파일만 네이티브 라이브러리를 로드하도록 수정한다
  - 네이티브 라이브러리를 로드하는 클래스를 loadjava의 **-j** 옵션을 사용하여 등록한다
  - PL 서버를 재시작한다 ([CUBRID 프로시저 언어 서버 참고](#))
- 이미 로드된 자바 클래스 파일을 loadjava로 다시 덮어쓰는 경우
  - **-j** 옵션을 사용하여 loadjava로 해당 클래스를 다시 등록한다

- PL 서버를 재시작한다 (CUBRID 프로시저 언어 서버 참고)

## loadjava 유틸리티

컴파일된 Java 파일이나 JAR(Java Archive) 파일을 CUBRID로 로드하기 위해서 **loadjava** 유틸리티를 사용한다. **loadjava** 유틸리티를 사용하여 Java \*.class 파일이나 \*.jar 파일을 로드하면 해당 파일이 해당 데이터베이스 경로로 이동한다.

```
loadjava [option] database-name java-class-file
```

- *database-name*: Java 파일을 로드하려고 하는 데이터베이스 이름
- *java-class-file*: 로드하려는 Java 클래스 파일 이름 또는 jar 파일 이름
- *[option]*
  - **-y\*\***(또는 **\*\*--overwrite**): 이름이 같은 클래스 파일이 존재하면 자동으로 덮어쓰기 한다. 기본값은 **no** 이다. 만약 **-y** 옵션을 명시하지 않고 로드할 때 이름이 같은 클래스 파일이 존재하면 덮어쓰기를 할 것인지 묻는다.
  - **-j\*\***(또는 **\*\*--jni**): Java \*.class 파일이나 \*.jar 파일을 정적으로 로드하기 위한 경로로 이동한다. 네이티브 라이브러리의 로딩 오류를 방지하기 위해서 JNI(네이티브 라이브러리)를 사용한 class는 반드시 이 옵션을 사용하여 로드해야 한다.
  - **-p\*\***(또는 **\*\*--package**): 클래스 파일을 로드할 때 패키지 이름을 지정한다.

\* 매뉴얼의 예제에서는 Java 파일 코드 상단에 모두 패키지가 명시되어 있지 않다. 이 경우에는 기본 패키지이며 **\*\*--p\*\*** 옵션을 사용하지 않아도 된다.  
\* 그러나 다음과 같이 패키지가 포함된 Java 파일을 로드할 때는 **\*\*--p\*\*** 옵션을 사용하여 패키지 이름을 지정해야 한다.

Java 패키지 이름이 포함된 클래스 로드 예제

Java 클래스에 패키지 선언이 포함된 경우 예를 들어:

```
package org.cubrid;

public class NumberUtils {
 public static int addIntegers(int num1, int num2) {
 return num1 + num2;
 }
}
```

```
}
}
```

#### 1. Java 파일 컴파일

```
javac NumberUtils.java
```

#### 2. 컴파일된 파일을 데이터베이스에 로드

```
loadjava demodb -p org.cubrid NumberUtils.class
```

#### 3. SQL에서 저장 함수 생성

```
CREATE OR REPLACE FUNCTION add_integer(a INT, b INT) RETURN INT
AS LANGUAGE JAVA
NAME 'org.cubrid.NumberUtils.addIntegers(int, int) return int';
```

#### 4. 함수 사용

```
SELECT add_integer(1, 2);
```

출력:

```
add_integer(1, 2)
=====
3
```

### 주의 사항

- java.sql.DatabaseMetaData는 지원하지 않는다.
- BLOB/CLOB 타입과 관련한 JDBC API 지원하지 않는다.
- 질의 수행과 관련 없고 클라이언트 측 JDBC에서만 사용하는 기능은 지원하지 않는다. 자세한 내용은 **부록**를 참조한다.
- 하나의 JDBC 객체로 질의 수행 시 여러 SQL 구문을 지원하지 않는다.
- 질의 수행으로 만들어지는 ResultSet은 non-updatable, non-scrollable, non-sensitive이다.

- 리턴 값 및 IN/OUT 파라미터의 타입 자릿수는 Java에서 무시하고 타입만 맞추어 그대로 데이터베이스에 전달한다.
- 저장 프로시저는 다른 저장 프로시저를 호출하거나 재귀적으로 자신을 호출할 수 있다. 최대 중첩 깊이는 16이다.

### 리턴 값 및 IN/OUT 파라미터에 대한 타입 자릿수 제한사항

리턴 값과 IN/OUT의 데이터 타입에 자릿수를 한정하는 경우, Java 저장 함수/프로시저 생성 시 정의한 자릿수는 무시하고 타입만 맞추어 Java에서 반환하는 값을 그대로 데이터베이스에 전달한다. 전달한 데이터에 대한 조작은 사용자가 데이터베이스에서 직접 처리하는 것을 원칙으로 한다.

다음과 같은 **typestring ()** Java 저장 함수를 살펴보자.

```
public class JavaSP1 {
 public static String typestring() {
 String temp = " ";
 for(int i = 0; i < 1; i++) {
 temp = temp + "1234567890";
 }
 return temp;
 }
}
```

```
CREATE FUNCTION typestring() RETURN CHAR(5) AS LANGUAGE JAVA
NAME 'JavaSP1.typestring() return java.lang.String';

CALL typestring();
```

Result

```
=====
' 1234567890'
```

## 부록

### 데이터 타입 매핑

Java 저장 함수/프로시저를 등록할 때, Java 저장 함수/프로시저의 반환 정의와 Java 파일의 선언부의 반환 정의가 일치하는지에 대해서는 검사하지 않는다. 따라서, Java 저장 함수/프로시저의 경우 등록할 때의 반환 정의를 따르고, Java 파일 선언부의 반환 정의는 사용자 정의 정보로서만 의미를 가지게 된다.

Call Specification에서는 SQL의 데이터 타입과 Java의 매개변수, 리턴 값의 데이터 타입이 맞게 대응되어야 한다. 또한 Java 저장함수/프로시저 구현 시, 질의 결과 (ResultSet)의 데이터 타입과 Java의 데이터 타입이 맞게 대응되어야 한다. CUBRID에서 허용되는 SQL과 Java의 데이터 타입의 관계는 다음의 표와 같다.

### 데이터 타입 매핑

| Category        | SQL Type          | Java Type                |
|-----------------|-------------------|--------------------------|
| Numeric Types   | SHORT, SMALLINT   | short, java.lang.Short   |
|                 | INT, INTEGER      | int, java.lang.Integer   |
|                 | BIGINT            | long, java.lang.Long     |
|                 | NUMERIC, DECIMAL  | java.math.BigDecimal     |
|                 | FLOAT, REAL       | float, java.lang.Float   |
|                 | DOUBLE, PRECISION | double, java.lang.Double |
| Date/Time Types | DATE              | java.sql.Date            |
|                 | TIME              | java.sql.Time            |
|                 | TIMESTAMP         | java.sql.Timestamp       |
|                 | DATETIME          | java.sql.Timestamp       |
|                 | TIMESTAMPPLTZ     | X (not supported)        |
|                 | TIMESTAMPPTZ      | X (not supported)        |



| Category               | SQL Type      | Java Type                                                                         |
|------------------------|---------------|-----------------------------------------------------------------------------------|
| Bit String Types       | DATETIMELTZ   | X (not supported)                                                                 |
|                        | DATETIMEZ     | X (not supported)                                                                 |
|                        | BIT           | X (not supported)                                                                 |
|                        | VARBIT        | X (not supported)                                                                 |
| Character String Types | CHAR          | java.lang.String                                                                  |
|                        | VARCHAR       | java.lang.String                                                                  |
| Enum Type              | ENUM          | X (not supported)                                                                 |
| LOB Types              | CLOB, BLOB    | X (not supported)                                                                 |
| Collection Types       | SET, SEQUENCE | MULTISET, java.lang.Object[], java primitive type array, java wrapper class array |
| Special Types          | JSON          | X (not supported)                                                                 |
|                        | OBJECT, OID   | cubrid.sql.CUBRIDOID<br><interface>                                               |
|                        | CURSOR        | java.sql.ResultSet<br><interface>                                                 |

묵시적 데이터 타입 변환

위의 표와 같이 SQL의 데이터 타입과 Java의 데이터 타입이 일치하지 않는 경우, CUBRID는 다음 표에 따라 묵시적으로 데이터 타입 변환을 시도한다. 묵시적 데이터 변환으로 인해 데이터가 손실될 수 있음을 주의해야 한다.

[illegible]

# Java Data Types

|     |      |      |      |      |      |      |      |      |       |       |
|-----|------|------|------|------|------|------|------|------|-------|-------|
|     |      |      |      |      |      | dou  |      |      |       |       |
|     | byt  | sho  | int, | lon  | flor | ble, | java |      |       |       |
| SQL | e,   | rt,  | java | g,   | t,   | java | .ma  |      |       | java  |
| Dat | java | java | .lan | java | java | .lan | th.B | java | java  | .sql. |
| a   | .lan | .lan | g.In | .lan | .lan | g.D  | igD  | .lan | .sql. | Tim   |
| Typ | g.By | g.Sh | teg  | g.Lo | g.Fl | oub  | eci  | g.St | Tim   | esta  |
| es  | te   | ort  | er   | ng   | oat  | le   | mal  | ring | e     | mp    |

## PRECISION

[illegible]

### Java Data Types

| SQL<br>Data<br>Types | byte,<br>java.<br>.lang.<br>Byte | short,<br>java.<br>.lang.<br>Short | int,<br>java.<br>.lang.<br>Integer | long,<br>java.<br>.lang.<br>Long | float,<br>java.<br>.lang.<br>Float | double,<br>java.<br>.lang.<br>Double | java.<br>.math.<br>BigDecimal | java.<br>.lang.<br>String | java.<br>.sql.<br>Timestamp |
|----------------------|----------------------------------|------------------------------------|------------------------------------|----------------------------------|------------------------------------|--------------------------------------|-------------------------------|---------------------------|-----------------------------|
|----------------------|----------------------------------|------------------------------------|------------------------------------|----------------------------------|------------------------------------|--------------------------------------|-------------------------------|---------------------------|-----------------------------|

### SEQUENCE

- X: 묵시적 변환을 허용하지 않음
- O: 묵시적 변환 발생

### JDBC API 지원표

| JDBC 인터페이스                 | 지원 여부              |
|----------------------------|--------------------|
| java.sql.CallableStatement | 지원                 |
| java.sql.Connection        | 지원                 |
| java.sql.Driver            | 지원 (Connection 생성) |
| java.sql.PreparedStatement | 지원                 |
| java.sql.ResultSet         | 지원                 |
| java.sql.ResultSetMetaData | 지원                 |
| CUBRIDOID                  | 지원                 |

| JDBC 인터페이스                 | 지원 여부 |
|----------------------------|-------|
| java.sql.Statement         | 지원    |
| java.sql.DriverManager     | 지원    |
| java.sql.SQLException      | 지원    |
| java.sql.Array             | 미지원   |
| java.sql.Blob              | 미지원   |
| java.sql.Clob              | 미지원   |
| java.sql.DatabaseMetaData  | 미지원   |
| java.sql.ParameterMetaData | 미지원   |
| java.sql.Ref               | 미지원   |
| java.sql.Savepoint         | 미지원   |
| java.sql.SQLData           | 미지원   |
| java.sql.SQLInput          | 미지원   |
| java.sql.Struct            | 미지원   |

다음의 표에서 지정하지 않은 JDBC API는 지원하지 않고 SQLException을 반환한다.

java.sql.Connection

| Method                                                                                                           | Description       |
|------------------------------------------------------------------------------------------------------------------|-------------------|
| Properties getClientInfo()                                                                                       | 연결 중인 클라이언트 정보 획득 |
| void rollback()                                                                                                  | do nothing        |
| Statement createStatement()                                                                                      | 질의문 수행            |
| Statement createStatement(int resultSetType, int resultSetConcurrency)                                           | 질의문 수행            |
| Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability)                 | 질의문 수행            |
| CallableStatement prepareCall(String sql)                                                                        | 질의문 수행            |
| CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)                           | 질의문 수행            |
| CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) | 질의문 수행            |
| PreparedStatement prepareStatement(String sql)                                                                   | 질의문 수행            |

| Method                                                                                                                         | Description          |
|--------------------------------------------------------------------------------------------------------------------------------|----------------------|
| PreparedStatement<br>prepareStatement(String sql,<br>int autoGeneratedKeys)                                                    | 질의문 수행               |
| PreparedStatement<br>prepareStatement(String sql,<br>int[] columnIndexes)                                                      | 질의문 수행               |
| PreparedStatement<br>prepareStatement(String sql,<br>int resultSetType, int resultSetConcurrency)                              | 질의문 수행               |
| PreparedStatement<br>prepareStatement(String sql,<br>int resultSetType, int resultSetConcurrency,<br>int resultSetHoldability) | 질의문 수행               |
| PreparedStatement<br>prepareStatement(String sql, String[] columnNames)                                                        | 질의문 수행               |
| void clearWarnings()                                                                                                           | do nothing           |
| void close()                                                                                                                   | close all statements |
| void commit()                                                                                                                  | do nothing           |
| boolean getAutoCommit()                                                                                                        | return false         |
| String getCatalog()                                                                                                            | return               |
| int getHoldability()                                                                                                           |                      |

| Method | Description                                   |
|--------|-----------------------------------------------|
|        | return<br>ResultSet.HOLD_CURSORS_OVER_COMMIT; |

int getTransactionIsolation()

|                          |             |
|--------------------------|-------------|
| SQLWarning getWarnings() | return null |
|--------------------------|-------------|

boolean isClosed() return false

|                      |              |
|----------------------|--------------|
| boolean isReadOnly() | return false |
|----------------------|--------------|

boolean isValid(int timeout) return true

|                                        |            |
|----------------------------------------|------------|
| void setAutoCommit(boolean autoCommit) | do nothing |
|----------------------------------------|------------|

void setCatalog(String catalog) do nothing

|                                      |            |
|--------------------------------------|------------|
| void setHoldability(int holdability) | do nothing |
|--------------------------------------|------------|

void setReadOnly(boolean readOnly) do nothing

|                                         |            |
|-----------------------------------------|------------|
| void setTransactionIsolation(int level) | do nothing |
|-----------------------------------------|------------|

java.sql.Statement

| Method | Description |
|--------|-------------|
|--------|-------------|

Connection getConnection()



| Method                                | Description                                                                          |
|---------------------------------------|--------------------------------------------------------------------------------------|
| int getFetchDirection()               | retruns ResultSet.FETCH_FORWARD                                                      |
| int getFetchSize()                    |                                                                                      |
| int getMaxFieldSize()                 |                                                                                      |
| int getMaxRows()                      |                                                                                      |
| int getQueryTimeout()                 | retruns 0                                                                            |
| int getResultSetConcurrency()         | retruns ResultSet.CONCUR_UPDATABLE                                                   |
| int getResultSetHoldability()         | return<br>ResultSet.HOLD_CURSORS_OVER_COMMIT or<br>ResultSet.CLOSE_CURSORS_AT_COMMIT |
| int getResultSetType()                | return ResultSet.TYPE_FORWARD_ONLY                                                   |
| int getUpdateCount()                  | return -1                                                                            |
| boolean isClosed()                    |                                                                                      |
| void setFetchDirection(int direction) |                                                                                      |
| void setFetchSize(int rows)           |                                                                                      |
| void setMaxFieldSize(int max)         |                                                                                      |
| void setMaxRows(int max)              |                                                                                      |

| Method                                               | Description         |
|------------------------------------------------------|---------------------|
| void setQueryTimeout(int seconds)                    |                     |
| void close()                                         |                     |
| boolean execute(String sql)                          |                     |
| boolean execute(String sql, int autoGeneratedKeys)   |                     |
| boolean execute(String sql, int[] columnIndexes)     |                     |
| boolean execute(String sql, String[] columnNames)    |                     |
| executeBatch()                                       | throws SQLException |
| ResultSet executeQuery(String sql)                   |                     |
| int executeUpdate(String sql)                        |                     |
| int executeUpdate(String sql, int autoGeneratedKeys) |                     |
| int executeUpdate(String sql, int[] columnIndexes)   |                     |
| int executeUpdate(String sql, String[] columnNames)  |                     |

| Method                                   | Description |
|------------------------------------------|-------------|
| ResultSet getGeneratedKeys()             |             |
| boolean getMoreResults()                 |             |
| ResultSet getResultSet()                 |             |
| void cancel()                            | do nothing  |
| void clearWarnings()                     |             |
| SQLWarning getWarnings()                 |             |
| void setCursorName(String name)          |             |
| void setEscapeProcessing(boolean enable) |             |
| java.sql.PreparedStatement               |             |
| Method                                   | Description |
| boolean execute()                        |             |
| ResultSet executeQuery()                 |             |
| int executeUpdate()                      |             |
| ResultSetMetaData getMetaData()          |             |

| Method                                                         | Description |
|----------------------------------------------------------------|-------------|
| void<br>setBigDecimal(int parameterIndex, BigDecimal x)        |             |
| void setBoolean(int parameterIndex, boolean x)                 |             |
| void setByte(int parameterIndex, byte x)                       |             |
| void setBytes(int parameterIndex, byte[] x)                    |             |
| void setDate(int parameterIndex, Date x)                       |             |
| void<br>setDate(int parameterIndex, Date x, Calendar cal)      |             |
| void setDouble(int parameterIndex, double x)                   |             |
| void setFloat(int parameterIndex, float x)                     |             |
| void setInt(int parameterIndex, int x)                         |             |
| void setLong(int parameterIndex, long x)                       |             |
| void setNull(int parameterIndex, int sqlType)                  |             |
| void setNull(int parameterIndex, int sqlType, String typeName) |             |
| void setObject(int parameterIndex, Object x)                   |             |

| Method                                                                             | Description |
|------------------------------------------------------------------------------------|-------------|
| void setObject(int parameterIndex, Object x, int targetSqlType)                    |             |
| void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength) |             |
| void setShort(int parameterIndex, short x)                                         |             |
| void setString(int parameterIndex, String x)                                       |             |
| void setTime(int parameterIndex, Time x)                                           |             |
| void setTime(int parameterIndex, Time x, Calendar cal)                             |             |
| void setTimestamp(int parameterIndex, Timestamp x)                                 |             |
| void setTimestamp(int parameterIndex, Timestamp x, Calendar cal)                   |             |
| java.sql.CallableStatement                                                         |             |

| Method                                       | Description |
|----------------------------------------------|-------------|
| BigDecimal getBigDecimal(int parameterIndex) |             |
| boolean getBoolean(int parameterIndex)       |             |

| Method                                                          | Description |
|-----------------------------------------------------------------|-------------|
| <code>byte getByte(int parameterIndex)</code>                   |             |
| <code>byte[] getBytes(int parameterIndex)</code>                |             |
| <code>Date getDate(int parameterIndex)</code>                   |             |
| <code>Date<br/>getDate(int parameterIndex, Calendar cal)</code> |             |
| <code>double getDouble(int parameterIndex)</code>               |             |
| <code>getFloat(int parameterIndex)</code>                       |             |
| <code>getInt(int parameterIndex)</code>                         |             |
| <code>getLong(int parameterIndex)</code>                        |             |
| <code>getObject(int parameterIndex)</code>                      |             |
| <code>getShort(int parameterIndex)</code>                       |             |
| <code>getString(int parameterIndex)</code>                      |             |
| <code>getTime(int parameterIndex)</code>                        |             |
| <code>getTime(int parameterIndex, Calendar cal)</code>          |             |
| <code>getTimestamp(int parameterIndex)</code>                   |             |

| Method | Description |
|--------|-------------|
|--------|-------------|

getTimestamp(int parameterIndex, Calendar cal)

registerOutParameter(int parameterIndex, int sqlType)

registerOutParameter(int parameterIndex, int sqlType, int scale)

registerOutParameter(int parameterIndex, int sqlType, String typeName)

wasNull()

java.sql.ResultSet

| Method | Description |
|--------|-------------|
|--------|-------------|

clearWarnings()

close()

deleteRow() throws SQLException

findColumn(String columnLabel)

first() throws SQLException

getBoolean(int columnIndex)

| Method                                                 | Description                                      |
|--------------------------------------------------------|--------------------------------------------------|
| <code>getBoolean(String columnLabel)</code>            |                                                  |
| <code>getByte(int columnIndex)</code>                  |                                                  |
| <code>getByte(String columnLabel)</code>               |                                                  |
| <code>getBytes(int columnIndex)</code>                 |                                                  |
| <code>getBytes(String columnLabel)</code>              |                                                  |
| <code>getConcurrency()</code>                          | return <code>ResultSet.CONCUR_READ_ONLY</code> ; |
| <code>getDate(int columnIndex)</code>                  |                                                  |
| <code>getDate(int columnIndex, Calendar cal)</code>    |                                                  |
| <code>getDate(String columnLabel)</code>               |                                                  |
| <code>getDate(String columnLabel, Calendar cal)</code> |                                                  |
| <code>getDouble(int columnIndex)</code>                |                                                  |
| <code>getDouble(String columnLabel)</code>             |                                                  |
| <code>getFetchDirection()</code>                       |                                                  |
| <code>getFetchSize()</code>                            |                                                  |
| <code>getFloat(int columnIndex)</code>                 |                                                  |



| Method                                     | Description |
|--------------------------------------------|-------------|
| <code>getFloat(String columnLabel)</code>  |             |
| <code>getHoldability()</code>              |             |
| <code>getInt(int columnIndex)</code>       |             |
| <code>getInt(String columnLabel)</code>    |             |
| <code>getLong(int columnIndex)</code>      |             |
| <code>getLong(String columnLabel)</code>   |             |
| <code>getMetaData()</code>                 |             |
| <code>getObject(int columnIndex)</code>    |             |
| <code>getObject(String columnLabel)</code> |             |
| <code>getRow()</code>                      |             |
| <code>getShort(int columnIndex)</code>     |             |
| <code>getShort(String columnLabel)</code>  |             |
| <code>getStatement()</code>                |             |
| <code>getString(int columnIndex)</code>    |             |
| <code>getString(String columnLabel)</code> |             |

| Method                                                      | Description                                      |
|-------------------------------------------------------------|--------------------------------------------------|
| <code>getTime(int columnIndex)</code>                       |                                                  |
| <code>getTime(int columnIndex, Calendar cal)</code>         |                                                  |
| <code>getTime(String columnLabel)</code>                    |                                                  |
| <code>getTime(String columnLabel, Calendar cal)</code>      |                                                  |
| <code>getTimestamp(int columnIndex)</code>                  |                                                  |
| <code>getTimestamp(int columnIndex, Calendar cal)</code>    |                                                  |
| <code>getTimestamp(String columnLabel)</code>               |                                                  |
| <code>getTimestamp(String columnLabel, Calendar cal)</code> |                                                  |
| <code>getType()</code>                                      | retruns <code>ResultSet.TYPE_FORWARD_ONLY</code> |
| <code>isAfterLast()</code>                                  |                                                  |
| <code>isBeforeFirst()</code>                                |                                                  |
| <code>isClosed()</code>                                     | return false                                     |
| <code>isFirst()</code>                                      |                                                  |
| <code>isLast()</code>                                       |                                                  |

| Method | Description |
|--------|-------------|
|--------|-------------|

wasNull()

getCursorName()

return

getWarnings()

return null

java.sql.ResultSetMetaData

| Method | Description |
|--------|-------------|
|--------|-------------|

getCatalogName (int column)

return

getColumnClassName(int column)

getColumnCount()

getColumnDisplaySize(int column)

getColumnLabel(int column)

getColumnName(int column)

getColumnType(int column)

getColumnTypeName(int column)

getPrecision(int column)

getScale(int column)

| Method                           | Description  |
|----------------------------------|--------------|
| getSchemaName(int column)        | return       |
| getTableName(int column)         |              |
| isAutoIncrement(int column)      |              |
| isCaseSensitive(int column)      |              |
| isCurrency(int column)           |              |
| isDefinitelyWritable(int column) | return false |
| isNullable(int column)           |              |
| isReadOnly(int column)           | return false |
| isSearchable(int column)         | return true  |
| isSigned(int column)             |              |
| isWritable(int column)           | return true  |

## 메서드

메서드(method)는 CUBRID 데이터베이스 시스템의 내장 함수로 C로 작성된 프로그램이고, **CALL** 문에 의해 호출된다. 프로그램은 메서드가 호출되었을 때 동적 로더에 의해 실행 중인 응용과 함께 로드(load)되고 연결(link)된다. 메서드 실행 결과 생성된 리턴 값(return value)은 호출자(caller)에게 전달된다.

## 제한사항

- 현재 메서드는 CUBRID 데이터베이스에서 **사용자 권한 관리 메서드**와 같은 일부 시스템에서만 사용 가능하다.
- 사용자 정의 메서드는 제약 사항이 많아서 사용을 권장하지 않으며, 시스템에 내장된 메서드에서만 사용된다.

## 메서드 타입

CSQL 언어는 클래스 메서드와 인스턴스 메서드 두 가지 타입의 메서드를 지원한다.

- **클래스 메서드** 는 클래스 객체에서 호출되는 메서드이다. 일반적으로 클래스의 새로운 인스턴스를 생성하거나 초기화하기 위하여 사용된다. 또한 클래스 속성에 접근하거나 갱신하기 위해서도 사용될 수 있다.
- **인스턴스 메서드** 는 클래스의 인스턴스에서 호출되는 메서드이다. 대부분의 연산들이 인스턴스에서 수행되기 때문에 클래스 메서드보다 더 자주 사용된다. 예를 들어 인스턴스 메서드는 인스턴스의 속성을 계산하거나 갱신하기 위해 작성될 수 있다. 이 메서드는 메서드가 정의된 클래스의 어떤 인스턴스에서도 호출될 수 있고, 메서드를 상속받은 어떠한 서브클래스의 인스턴스에서도 호출할 수 있다.

메서드에 대한 상속 법칙은 속성에 대한 상속 법칙과 비슷하다. 서브클래스는 슈퍼클래스로부터 클래스와 인스턴스 메서드를 상속받는다. 서브클래스는 슈퍼클래스로부터 클래스의 정의나 인스턴스 메서드의 정의를 따를 수 있다.

메서드 이름에 대한 충돌 해결 규칙은 속성 이름에 대한 충돌 해결 규칙과 같다. 속성과 메서드 상속 충돌에 대한 추가적인 정보는 **클래스 충돌 해결** 을 참조한다.

## 메서드 호출

**CALL** 문은 데이터베이스에 정의된 메서드를 호출하기 위해 사용된다. 클래스 메서드, 인스턴스 메서드 모두 **CALL** 문으로 호출이 가능하다. **CALL** 문으로 시스템에 정의된 메서드를 호출하는 예는 **사용자 권한 관리 메서드** 를 참고한다.

```
CALL <method_call> ;

<method_call> ::=
 method_name ([<arg_value> [{, <arg_value> } ...]]) ON
<call_target> [<to_variable>] |
 method_name (<call_target> [, <arg_value> [{,
<arg_value>} ...]]) [<to_variable>]

<arg_value> ::=
 any CSQL expression

<call_target> ::=
 an object-valued expression
```

```
<to_variable> ::=
 INTO variable |
 TO variable
```

- *method\_name*은 클래스에 정의된 메서드의 이름이거나, 시스템에 정의된 메서드의 이름이다. 메서드는 하나 혹은 그 이상의 인수 값을 필요로 한다. 메서드에 인수가 없으면 빈 괄호를 사용해야 한다.
- *<call\_target>*은 클래스 이름, 변수, 혹은 또 다른 메서드 호출(객체를 반환하는)을 포함하는 객체 값의 표현식(object-valued expression)이다. 만약 클래스 객체에서 동작하는 클래스 메서드를 호출하려면, *<call\_target>* 앞에 반드시 **CLASS** 키워드가 있어야 한다. 이 경우 클래스 이름은 클래스 메서드가 정의된 클래스의 이름이어야 한다. 만약 인스턴스 메서드를 호출하려면, 인스턴스 객체를 나타내는 식을 지정해야 한다. 클래스 메서드나 인스턴스 메서드에 의해 반환되는 값은 선택적으로 *<to\_variable>*에 저장할 수 있다. 이 반환 변수의 값은 *<call\_target>*이나 *<arg\_value>* 파라미터처럼 **CALL** 문 내에 사용될 수 있다.
- 중첩된 메서드 호출은 다른 *method\_call*이 메서드의 *<call\_target>* 이거나 *<arg\_value>* 인수의 하나로 주어질 때 성립된다.

## CUBRID 운영

이 장에서는 데이터베이스 관리자(**DBA**)가 CUBRID 시스템을 사용하는데 필요한 작업 방법을 설명한다.

- CUBRID 서버, 브로커, 프로시저 언어 서버 및 매니저 서버 등의 다양한 프로세스들을 구동하고 중지하는 방법을 설명한다. **CUBRID 프로세스 제어**를 참조한다.
- **cubrid** 유틸리티를 통해 데이터베이스 생성 및 삭제, 볼륨 추가와 같은 데이터베이스 관리 작업, 데이터베이스를 다른 곳으로 이동하거나 시스템 버전에 맞춰서 변경하는 마이그레이션 작업, 장애 대비를 위한 데이터베이스의 백업 및 복구 작업 등에 대해 설명한다. **cubrid 유틸리티**를 참조한다.
- 시스템 설정 방법에 대해 설명한다. **시스템 설정**을 참조한다.
- 실행 중인 프로세스를 동적으로 모니터링하고 추적할 수 있는 SystemTap 사용 방법에 대해 설명한다. **SystemTap**을 참조한다.
- 트러블슈팅 방법에 대해 설명한다. **트러블슈팅**을 참조한다.

**cubrid** 유틸리티는 CUBRID 서비스를 통합 관리할 수 있는 기능을 제공하며, CUBRID 서비스 프로세스를 관리하는 서비스 관리 유틸리티와 데이터베이스를 관리하는 데이터베이스 관리 유틸리티로 구분된다.

서비스 관리 유틸리티는 다음과 같다.

- 서비스 유틸리티: 마스터 프로세스를 구동 및 관리한다.
  - `cubrid service`
- 서버 유틸리티: 서버 프로세스를 구동 및 관리한다.
  - `cubrid server`
- 브로커 유틸리티: 브로커 프로세스 및 응용서버(CAS) 프로세스를 구동 및 관리한다.
  - `cubrid broker`
- 게이트웨이 유틸리티: 게이트웨이 프로세스 및 응용서버(CAS) 프로세스를 구동 및 관리한다.
  - `cubrid gateway`
- 매니저 유틸리티: 매니저 서버 프로세스를 구동 및 관리한다.
  - `cubrid manager`
- HA 유틸리티: HA 관련 프로세스를 구동 및 관리한다.
  - `cubrid heartbeat`
- PL 유틸리티: 프로시저 언어 서버를 관리한다.
  - `cubrid pl`

자세한 설명은 [CUBRID 프로세스 제어](#) 절을 참조한다.

데이터베이스 관리 유틸리티는 다음과 같다.

- 데이터베이스 생성, 볼륨 추가, 삭제
  - `cubrid createdb`
  - `cubrid addvoldb`
  - `cubrid deletedb`
- 데이터베이스 이름 변경, 호스트 변경, 복사/이동, 등록
  - `cubrid renamedb`
  - `cubrid alterdbhost`
  - `cubrid copydb`
  - `cubrid installdb`

- 데이터베이스 백업
  - `cubrid backupdb`
- 데이터베이스 복구
  - `cubrid restoredb`
- 내보내기과 가져오기
  - `cubrid unloaddb`
  - `cubrid loaddb`
- 데이터베이스 공간 확인, 공간 정리
  - `cubrid spacedb`
  - `cubrid compactdb`
- 통계 정보 갱신, 질의 계획 확인
  - `cubrid plandump`
  - `cubrid optimizedb`
  - `cubrid statdump`
- 잠금 확인, 트랜잭션 확인, 트랜잭션 제거
  - `cubrid lockdb`
  - `cubrid tranlist`
  - `cubrid killtran`
- 데이터베이스 진단/파라미터 출력
  - `cubrid checkdb`
  - `cubrid diagdb`
  - `cubrid paramdump`
- HA 모드 변경,로그 복제/반영
  - `cubrid changemode`
  - `cubrid applyinfo`



- 로컬 컴파일/출력
  - `cubrid genlocale`
  - `cubrid dumplocale`

자세한 설명은 `cubrid 유틸리티` 를 참조한다.

#### 참고

Windows Vista 이상 버전에서 **cubrid** 유틸리티를 사용하여 서비스를 제어하려면 명령 프롬프트 창을 관리자 권한으로 실행하여 사용하는 것을 권장한다. 명령 프롬프트 창을 관리자 권한으로 구동하지 않고 **cubrid** 유틸리티를 사용하면 UAC(User Account Control) 대화 상자를 통하여 관리자 권한으로 수행할 수는 있으나 수행 결과 메시지를 확인할 수 없다.

Windows Vista 이상 버전에서 명령 프롬프트 창을 관리자 권한으로 실행하려면 [시작] > [모든 프로그램] > [보조 프로그램] > [명령 프롬프트]를 마우스 오른쪽 버튼으로 클릭하여 [관리자 권한으로 실행]을 선택한다. 권한 상승을 확인하는 대화 상자가 나타났을 때 [예]를 클릭하면 명령 프롬프트가 관리자 권한으로 실행된다.

## CUBRID 프로세스 제어

`cubrid` 유틸리티를 통해서 CUBRID 프로세스들을 제어할 수 있다.

### CUBRID 서비스 제어

CUBRID 설정 파일에 등록된 서비스를 제어하기 위한 **cubrid** 유틸리티 구문은 다음과 같다.

```
cubrid service <command>

<command>: {start|stop|restart|status}
```

- start: 서비스 구동
- stop: 서비스 종료
- restart: 서비스 재시작
- status: 서비스 상태 확인

추가로 입력해야 할 옵션이나 인수는 없다.

## 데이터베이스 서버 제어

데이터베이스 서버 프로세스를 제어하기 위한 **cubrid** 유틸리티 구문은 다음과 같다.

```
cubrid server <command> [database_name]
<command>: {start|stop|restart|status}
```

- start: 데이터베이스 서버 프로세스 구동
- stop: 데이터베이스 서버 프로세스 종료
- restart: 데이터베이스 서버 프로세스 재시작
- status: 데이터베이스 서버 프로세스 상태 확인

모든 명령어는 특정 데이터베이스 이름 (**[database\_name]**) 을 인수로 지정할 수 있다.

데이터베이스 이름을 지정하지 않으면 **status** 명령어는 구동 중인 모든 데이터베이스 정보를 표시하고, **status**를 제외한 다른 명령어에서는 cubrid.conf에 **[service]** 섹션의 **server** 프로퍼티에서 데이터베이스 이름을 참조하여 실행한다.

```
cubrid.conf

[service]

...

server=demodb,testdb

...
```

```
% cubrid server start

@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success

@ cubrid server start: testdb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4
```

```
++ cubrid server start: success
```

## 브로커 제어

CUBRID 브로커 프로세스를 제어하기 위한 cubrid 유틸리티 구문은 다음과 같다. <command>로 올 수 있는 명령어는 브로커 프로세스 구동을 위한 start, 종료를 위한 stop, 재시작을 위한 restart, 상태 확인을 위한 status, 브로커 접속 제한을 위한 acl, 명시한 브로커만 사용 가능하게 하거나 불가능하게 하는 on/off, 브로커 접속을 리셋하기 위한 reset, 설정 정보 출력을 위한 info, SHARD key를 가지고 SHARD ID(SHARD 데이터베이스 ID)를 얻는 getid가 있다.

또한, SHARD 기능은 브로커가 구동되고 cubrid\_broker.conf의 SHARD라는 브로커 파라미터 값이 ON 일 때만 사용할 수 있다.

```
cubrid broker <command>
<command>: start
 |stop
 |restart
 |status [options] [broker_name_expr]
 |acl {status|reload} broker_name
 |on <broker_name> |off <broker_name>
 |reset broker_name
 |info
 |getid -b <broker_name> [-f] shard_key
```

## 게이트웨이 제어

CUBRID 게이트웨이 프로세스를 제어하기 위한 **cubrid** 유틸리티 구문은 다음과 같다.

```
cubrid gateway <command>
<command>: start
 |stop
 |restart
 |status [options] [gateway_name_expr]
 |acl {status|reload} gateway_name
 |on <gateway_name> |off <gateway_name>
 |reset gateway_name
 |info
```

- start: 게이트웨이 프로세스 구동
- stop: 게이트웨이 프로세스 종료
- restart: 게이트웨이 프로세스 재시작
- status: 게이트웨이 상태 확인

- acl: 게이트웨이 접속 제한
- on/off: 명시한 게이트웨이만 사용 가능하게 하거나 불가능하게 함
- reset: 게이트웨이 접속을 리셋함
- info: 게이트웨이 설정 정보 출력

## CUBRID 매니저 서버 제어

CUBRID 매니저를 사용하기 위해서는 데이터베이스 서버가 실행된 곳에 매니저 서버가 실행되어야 한다. CUBRID 매니저 프로세스를 제어하기 위한 **cubrid** 유틸리티 구문은 다음과 같다.

```
cubrid manager <command>
<command>: {start|stop|status}
```

- start: 매니저 서버 프로세스 구동
- stop: 매니저 서버 프로세스 종료
- status: 매니저 서버 프로세스 상태 확인

## CUBRID HA 제어

CUBRID HA 기능을 사용하기 위한 **cubrid heartbeat** 유틸리티 구문은 다음과 같다.

```
cubrid heartbeat <command>
<command>: {start|stop|copylogdb|applylogdb|reload|status}
```

- start: HA 관련 프로세스 구동
- stop: HA 관련 프로세스 종료
- copylogdb: copylogdb 프로세스를 시작 또는 정지
- applylogdb: applylogdb 프로세스를 시작 또는 정지
- reload: HA 구성정보를 다시 읽어서 새로운 구성에 맞게 실행
- status: HA 상태 정보를 확인

자세한 내용은 **cubrid heartbeat** 유틸리티를 참고한다.

## CUBRID 프로시저 언어 (PL) 서버 제어

CUBRID 프로시저 언어 (PL) 서버 제어 프로세스를 제어하기 위한 **cubrid** 유틸리티 구문은 다음과 같다.

```
cubrid pl <command> [database_name]
<command>: {restart|status}
```

- restart: 프로시저 언어 서버 프로세스 재구동
- status: 프로시저 언어 서버 프로세스 상태 확인

### 참고

- 모든 명령어는 특정 데이터베이스 이름 (**[database\_name]**) 을 인수로 지정할 수 있다.
- **restart** 명령어는 프로시저 언어 서버를 강제 종료하며, 데이터베이스 서버가 이를 감지하여 프로시저 언어 서버를 자동으로 재구동한다.
- **status** 명령어 수행시 데이터베이스 이름을 지정하지 않으면 구동 중인 모든 데이터베이스에 대해 프로시저 언어 서버의 상태 정보를 표시한다.

```
% cubrid pl restart demodb

@ cubrid pl start: demodb
++ cubrid pl start: success

% cubrid pl status demodb

@ cubrid pl status
Procedural Language Server (demodb, pid 12345, UDS)
VM arguments :

...

```

### 참고

이전 버전에서 사용했던 CUBRID 자바 저장 프로시저 (Java SP) 서버 대신 CUBRID 11.4 버전부터는 CUBRID 프로시저 언어 (PL) 서버로 변경되었다.

## CUBRID 서비스

### 서비스 등록

사용자는 임의로 데이터베이스 서버, CUBRID 브로커, CUBRID 게이트웨이, CUBRID 매니저, CUBRID HA를 데이터베이스 환경 설정 파일(cubrid.conf)에 CUBRID 서비스로 등록할 수 있다. 이를 위해 cubrid.conf의 service 파라미터 값으로 각각 server, broker, gateway, manager, heartbeat를 입력하면 되며, 이들을 쉼표(,)로 구분하여 여러 개를 같이 등록할 수 있다.

사용자가 별도로 서비스를 등록하지 않으면, 기본적으로 마스터 프로세스(cub\_master)만 등록된다. CUBRID 서비스에 등록되어 있으면 **cubrid service** 유틸리티를 사용해서 한 번에 관련된 프로세스들을 모두 구동, 정지하거나 상태를 알아볼 수 있어 편리하다.

- CUBRID HA를 설정하는 방법은 **cubrid service에 HA 등록**을 참고한다.

다음은 데이터베이스 환경 설정 파일에서 데이터베이스 서버와 브로커를 서비스로 등록하고, CUBRID 서비스 구동과 함께 demodb와 testdb라는 데이터베이스를 자동으로 시작하도록 설정한 예이다.

```
cubrid.conf
...

[service]

The list of processes to be started automatically by 'cubrid
service start' command
Any combinations are available with server, broker, gateway,
manager and heartbeat.
service=server,broker,gateway

The list of database servers in all by 'cubrid service start'
command.
This property is effective only when the above 'service' property
contains 'server' keyword.
server=demodb,testdb
```

### 서비스 구동

Linux 환경에서는 CUBRID 설치 후 CUBRID 서비스 구동을 위해 아래와 같이 입력한다. 데이터베이스 환경 설정 파일에서 서비스를 등록하지 않으면 기본적으로 마스터 프로세스(cub\_master)만 구동된다.

Windows 환경에서는 시스템 권한을 가진 사용자로 로그인한 경우에만 아래의 명령이 정상 수행된다. 관리자 또는 일반 사용자는 CUBRID 매니저 설치 후 작업 표시줄에 생성되는 CUBRID 서비스 트레이 아이콘을 클릭하여 CUBRID Server를 구동 또는 정지할 수 있다.

```
% cubrid service start

@ cubrid master start
++ cubrid master start: success
```

이미 마스터 프로세스가 구동 중이라면 다음과 같은 메시지가 표시된다.

```
% cubrid service start

@ cubrid master start
++ cubrid master is running.
```

마스터 프로세스의 구동에 실패한 경우라면 다음과 같은 메시지가 표시된다. 다음은 데이터베이스 환경 설정 파일(cubrid.conf)에 설정된 **cubrid\_port\_id** 파라미터 값이 충돌하여 구동에 실패한 예이다. 이런 경우에는 해당 포트를 변경하여 충돌 문제를 해결할 수 있다. 해당 포트를 점유하고 있는 프로세스가 없는데도 구동에 실패한다면 /tmp/CUBRID1523 파일을 삭제한 후 재시작한다.

```
% cubrid service start

@ cubrid master start
cub_master: '/tmp/CUBRID1523' file for UNIX domain socket exist....
Operation not permitted
++ cubrid master start: fail
```

**CUBRID 서비스**에 설명된 대로 서비스를 등록한 후, 서비스를 구동하기 위해 다음과 같이 입력한다. 마스터 프로세스, 데이터베이스 서버 프로세스, 브로커, 게이트웨이 및 등록된 *demodb*, *testdb*가 한번에 구동됨을 확인할 수 있다.

```
% cubrid service start

@ cubrid master start
++ cubrid master start: success
@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.
CUBRID 11.4

++ cubrid server start: success
@ cubrid server start: testdb

This may take a long time depending on the amount of recovery works
to do.
CUBRID 11.4

++ cubrid server start: success
```

```
@ cubrid broker start
++ cubrid broker start: success
@ cubrid gateway start
++ cubrid gateway start: success
```

## 서비스 종료

CUBRID 서비스를 종료하려면 다음과 같이 입력한다. 사용자에 의해 등록된 서비스가 없는 경우, 마스터 프로세스만 종료된다.

```
% cubrid service stop
@ cubrid master stop
++ cubrid master stop: success
```

등록된 CUBRID 서비스를 종료하려면 다음과 같이 입력한다. *demodb*, *testdb*는 물론, 서버 프로세스, 브로커 프로세스, 게이트웨이 프로세스, 마스터 프로세스가 모두 종료됨을 확인할 수 있다.

```
% cubrid service stop
@ cubrid server stop: demodb

Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
@ cubrid server stop: testdb
Server testdb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
@ cubrid broker stop
++ cubrid broker stop: success
@ cubrid gateway stop
++ cubrid gateway stop: success
@ cubrid master stop
++ cubrid master stop: success
```

## 서비스 재구동

CUBRID 서비스를 재구동하려면 다음과 같이 입력한다. 사용자에 의해 등록된 서비스가 없는 경우, 마스터 프로세스만 종료 후 재구동된다.

```
% cubrid service restart

@ cubrid master stop
++ cubrid master stop: success
@ cubrid master start
++ cubrid master start: success
```



등록된 CUBRID 서비스를 다음과 같이 입력한다. *demodb*, *testdb*는 물론, 서버 프로세스, 브로커 프로세스, 게이트웨이 프로세스, 마스터 프로세스가 모두 종료된 후 재구동되는 것을 확인할 수 있다.

```
% cubrid service restart

@ cubrid server stop: demodb
Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
@ cubrid server stop: testdb
Server testdb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
@ cubrid broker stop
++ cubrid broker stop: success
@ cubrid gateway stop
++ cubrid gateway stop: success
@ cubrid master stop
++ cubrid master stop: success
@ cubrid master start
++ cubrid master start: success
@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
@ cubrid server start: testdb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
@ cubrid broker start
++ cubrid broker start: success
@ cubrid gateway start
++ cubrid gateway start: success
```

## 서비스 상태 관리

등록된 마스터 프로세스, 데이터베이스 서버의 상태를 확인하기 위하여 다음과 같이 입력한다.

```
% cubrid service status

@ cubrid master status
```

```

++ cubrid master is running.
@ cubrid server status

Server testdb (rel 11.4, pid 31059)
Server demodb (rel 11.4, pid 30950)

@ cubrid broker status
NAME PID PORT AS JQ
TPS QPS SELECT INSERT UPDATE DELETE
OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-Q #CONNECT
#REJECT
=====
=====
=====
* query_editor 10877 30000 5 0
0 0 0 0 0 0
0 0/60.0 0/60.0 0 0
0 0
* broker1 10889 33000 5 0
0 0 0 0 0 0
0 0/60.0 0/60.0 0 0
0 0
@ cubrid gateway status
NAME PID PORT AS JQ
TPS QPS SELECT INSERT UPDATE DELETE
OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-Q #CONNECT
#REJECT
=====
=====
=====
* oracle_gateway 10903 53000 5 0
0 0 0 0 0 0
0 0/60.0 0/60.0 0 0
0 0

@ cubrid manager server status
++ cubrid manager server is not running.

```

만약, 마스터 프로세스가 중지된 상태라면, 다음과 같은 메시지가 출력된다.

```

% cubrid service status
@ cubrid master status
++ cubrid master is not running.

```

## cubrid 유틸리티 로깅

CUBRID는 cubrid 유틸리티의 수행 결과에 대한 로깅 기능을 제공한다.

## 로깅 내용

**\$CUBRID/log/cubrid\_utility.log** 파일에 다음의 내용들이 로깅된다.

- cubrid 유틸리티를 통해 수행된 모든 명령: usage, version, parsing 에러는 제외
- cubrid 유틸리티 명령들의 수행 결과: 성공/실패
- 실패 시 오류메시지

## 로그 파일 크기

**cubrid\_utility.log** 파일의 크기는 cubrid.conf의 error\_log\_size 파라미터에 설정한 값만큼 커지고, 해당 크기만큼 커지면 **cubrid\_utility.log.bak** 파일로 백업된다.

## 로그 포맷

시간 (cubrid PID) 내용

출력되는 로그 파일의 예는 다음과 같다.

```
13-11-19 15:27:19.426 (17724) cubrid manager stop
13-11-19 15:27:19.430 (17724) FAILURE: ++ cubrid manager server is
not running.
13-11-19 15:27:19.434 (17726) cubrid service start
13-11-19 15:27:19.439 (17726) FAILURE: ++ cubrid master is running.
13-11-19 15:27:22.931 (17726) SUCCESS
13-11-19 15:27:22.936 (17756) cubrid service restart
13-11-19 15:27:31.667 (17756) SUCCESS
13-11-19 15:27:31.671 (17868) cubrid service stop
13-11-19 15:27:34.909 (17868) SUCCESS
```

단, Windows 환경에서는 일부 **cubrid** 명령이 서비스 프로세스를 통해 다시 실행되는 구조이므로 Linux와 달리 중첩된 정보가 출력될 수 있다.

```
13-11-13 17:17:47.638 (3820) cubrid service stop
13-11-13 17:17:47.704 (7848) d:\CUBRID\bin\cubrid.exe service stop
--for-windows-service
13-11-13 17:17:56.027 (7848) SUCCESS
13-11-13 17:17:57.136 (3820) SUCCESS
```

또한 Windows 환경에서는 서비스 프로세스를 통해 수행되는 프로세스는 오류 메시지를 출력하지 못하므로, 서비스 구동과 관련된 오류메시지는 반드시 **cubrid\_utility.log** 를 통해 확인해야 한다.

## 데이터베이스 서버

### 데이터베이스 서버 구동

*demodb* 서버를 구동하기 위하여 다음과 같이 입력한다.

```
% cubrid server start demodb

@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
```

마스터 프로세스가 중지된 상태에서 *demodb* 서버를 시작하면 다음과 같이 자동으로 마스터 프로세스를 구동한 후 지정된 데이터베이스 서버를 구동한다.

```
% cubrid server start demodb

@ cubrid master start
++ cubrid master start: success
@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
```

이미 *demodb* 서버가 구동 중인 상태라면 다음과 같은 메시지가 출력된다.

```
% cubrid server start demodb

@ cubrid server start: demodb
++ cubrid server 'demodb' is running.
```

**cubrid server start** 명령은 HA 모드의 설정과는 상관없이 특정 데이터베이스의 *cub\_server* 프로세스만 구동한다. HA 환경에서 데이터베이스를 구동하려면 **cubrid heartbeat start**를 사용해야 한다.

## 데이터베이스 서버 종료

*demodb* 서버 구동을 종료하기 위하여 다음과 같이 입력한다.

```
% cubrid server stop demodb

@ cubrid server stop: demodb
Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
```

이미 *demodb* 서버가 종료된 상태라면, 다음과 같은 메시지가 출력된다.

```
% cubrid server stop demodb

@ cubrid server stop: demodb
++ cubrid server 'demodb' is not running.
```

**cubrid server stop** 명령은 HA 모드의 설정과는 상관없이 특정 데이터베이스의 *cub\_server* 프로세스만 종료하며, 데이터베이스 서버가 재시작되거나 failover가 일어나지 않으므로 주의해야 한다. HA 환경에서 데이터베이스를 중지하려면 **cubrid heartbeat stop** 를 사용해야 한다.

## 데이터베이스 서버 재구동

*demodb* 서버를 재구동하기 위하여 다음과 같이 입력한다. 이미 구동 중인 *demodb* 서버를 중지시킨 후 재구동하는 것을 알 수 있다.

```
% cubrid server restart demodb

@ cubrid server stop: demodb
Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
```

## 데이터베이스 상태 확인

데이터베이스 서버의 상태를 확인하기 위하여 다음과 같이 입력한다. 구동 중인 모든 데이터베이스 서버의 이름이 표시된다.

```
% cubrid server status

@ cubrid server status
Server testdb (rel 11.4, pid 24465)
Server demodb (rel 11.4, pid 24342)
```

마스터 프로세스가 중지된 상태라면, 다음과 같은 메시지가 출력된다.

```
% cubrid server status

@ cubrid server status
++ cubrid master is not running.
```

## 데이터베이스 서버 접속 제한

데이터베이스 서버에 접속하는 브로커 및 CSQL 인터프리터를 제한하려면 **cubrid.conf**의 **access\_ip\_control** 파라미터 값을 yes로 설정하고, **access\_ip\_control\_file** 파라미터 값에 접속을 허용하는 IP 목록을 작성한 파일 경로를 입력한다. 파일 경로는 절대 경로로 입력하며, 상대 경로로 입력하면 Linux에서는 **\$CUBRID/conf** 이하, Windows에서는 **%CUBRID%\conf** 이하의 위치에서 파일을 찾는다.

**cubrid.conf** 파일에는 다음과 같이 설정한다.

```
cubrid.conf
access_ip_control=yes
access_ip_control_file="/home1/cubrid1/CUBRID/db.access"
```

**access\_ip\_control\_file** 파일의 작성 형식은 다음과 같다.

```
[@<db_name>]
<ip_addr>
...
```

- <db\_name>: 접근을 허용할 데이터베이스 이름.
- <ip\_addr>: 접근을 허용할 IP 주소. 뒷자리를 \*로 입력하면 뒷자리의 모든 IP를 허용한다. 하나의 데이터베이스 이름 다음 줄에 여러 줄의 <ip\_addr>을 추가할 수 있다.

여러 개의 데이터베이스에 대해 설정하기 위해 [@<db\_name>]과 <ip\_addr>을 추가로 지정할 수 있다.

**access\_ip\_control**이 yes인 상태에서 **access\_ip\_control\_file**이 설정되지 않으면, 서버는 모든 IP를 차단하고 localhost만 접속을 허용한다. 서버 구동 시 잘못된 형식으로 인해 **access\_ip\_control\_file** 분석에 실패하면 서버는 구동되지 않는다.

다음은 **access\_ip\_control\_file**의 한 예이다.

```
[@dbname1]
192.10.20.30
192.156.*

[@dbname2]
*

[@dbname3]
192.168.1.15
```

위의 예에서 *dbname1* 데이터베이스는 192.10.20.30이거나 192.156으로 시작하는 IP의 접속을 허용한다. *dbname2* 데이터베이스는 모든 IP의 접속을 허용한다. *dbname3* 데이터베이스는 192.168.1.15인 IP의 접속을 허용한다.

이미 구동되어 있는 데이터베이스에 대해서는 다음 명령어를 통해 설정 파일을 다시 적용하거나, 현재 적용된 상태를 확인할 수 있다.

**access\_ip\_control\_file**의 내용을 변경하고 이를 서버에 적용하려면 다음 명령어를 사용한다.

```
cubrid server acl reload <database_name>
```

현재 구동 중인 서버의 IP 설정 내용을 출력하려면 다음 명령어를 사용한다.

```
cubrid server acl status <database_name>
```

## 데이터베이스 서버 로그

### 에러 로그

허용되지 않는 IP에서 접근하면 서버 에러 로그 파일에 다음과 같은 서버 에러 로그가 남는다.

```
Time: 10/29/10 17:32:42.360 - ERROR *** ERROR CODE = -1022, Tran =
0, CLIENT = (unknown):(unknown)(-1), EID = 2
Address(10.24.18.66) is not authorized.
```

데이터베이스 서버의 에러 로그는 **\$CUBRID/log/server** 디렉터리에 생성되며, 파일 이름은 <db\_name>\_<yyyymmdd>\_<hhmmi>.err 형식으로 저장된다. 확장자는 .err이다.

```
demodb_20130618_1655.err
```

### 참고

브로커에서의 접속 제한을 위해서는 **브로커 서버 접속 제한** 을 참고한다.

## 이벤트 로그

질의 성능에 영향을 주는 이벤트가 발생하면 해당 이벤트를 이벤트 로그에 기록한다.

이벤트 로그에 저장되는 이벤트는 *SLOW\_QUERY*, *MANY\_IOREADS*, *LOCK\_TIMEOUT*, *DEADLOCK*, 그리고 *TEMP\_VOLUME\_EXPAND* 가 있다.

해당 로그 파일은 **\$CUBRID/log/server** 디렉터리에 생성되며, 파일 이름은 <db\_name>\_<yyyymmdd>\_<hhmmi>.event 형식으로 저장된다. 확장자는 .event이다.

```
demodb_20130618_1655.event
```

## SLOW\_QUERY

슬로우 쿼리(slow query)가 발생했을 때 기록한다. cubrid.conf의 **sql\_trace\_slow** 파라미터 값이 설정되면 동작한다. 다음은 출력 예이다.

```
06/12/13 16:41:05.558 - SLOW_QUERY
client: PUBLIC@testhost|csql(13173)
sql: update [y] [y] set [y].[a]= ?:1 where [y].[a]= ?:0 using
index [y].[pk_y_a](+)
bind: 5
bind: 200
time: 1015
```



```
buffer: fetch=48, ioread=2, iowrite=0
wait: cs=1, lock=1010, latch=0
```

- client: <DB 사용자>@<응용 클라이언트 호스트 명>|<프로그램 이름>(<프로세스 ID>)
- sql: 슬로우 쿼리
- bind: 바인딩되는 값. sql 항목에 나타난 ?:<num>에서 <num>의 순서대로 출력된다. ?:0의 값이 5이고, ?:1의 값이 200이다.
- time: 수행 시간 (ms)
- buffer: buffer 수행 통계
  - fetch: 페치 페이지 개수
  - ioread: I/O 읽기 페이지 개수
  - iowrite: I/O 쓰기 페이지 개수
- wait: 대기 시간
  - cs: 크리티컬 섹션에서 대기한 시간(ms)
  - lock: 잠금을 획득하려고 대기한 시간(ms)
  - latch: 래치를 획득하려고 대기한 시간(ms)

위의 예에서 질의 수행 시간이 1015ms가 소요되었는데 lock wait 시간이 1010ms 소요되어, 질의 수행 시간의 대부분이 잠금 대기 시간이었음을 알 수 있다.

## MANY\_IOREADS

I/O 읽기를 많이 발생시킨 질의를 기록한다. cubrid.conf의 **sql\_trace\_ioread\_pages** 파라미터 설정 값 이상 I/O 읽기가 발생하면 로그를 기록한다. 다음은 출력 예이다.

```
06/12/13 17:07:29.457 - MANY_IOREADS
client: PUBLIC@testhost|csql(12852)
sql: update [x] [x] set [x].[a]= ?:1 where ([x].[a]> ?:0) using
index [x].[idx](+)
bind: 8
bind: 100
time: 528
ioreads: 15648
```

- client: <DB 사용자>@<응용 클라이언트 호스트 명>|<프로세스 이름>(<프로세스 ID>)
- sql: 많은 I/O 읽기를 유발한 SQL

- bind: 바인딩되는 값. sql 항목에 나타난 ?:<num>에서 <num>의 순서대로 출력된다. ?:0의 값이 8이고, ?:1의 값이 100이다.
- time: 수행 시간 (ms)
- ioreads: I/O 읽기 페이지 개수

## LOCK\_TIMEOUT

잠금 타임아웃(lock timeout)이 발생하면 waiter와 blocker의 질의문을 기록한다. 다음은 출력 예이다.

```
02/02/16 20:56:18.650 - LOCK_TIMEOUT
waiter:
 client: public@testhost|csql(21529)
 lock: X_LOCK (oid=0|650|3, table=t)
 sql: update [t] [t] set [t].[a]= ?:0 where [t].[a]= ?:1
 bind: 2
 bind: 1

blocker:
 client: public@testhost|csql(21541)
 lock: X_LOCK (oid=0|650|3, table=t)
 sql: update [t] [t] set [t].[a]= ?:0 where [t].[a]= ?:1
 bind: 3
 bind: 1
```

- waiter: 잠금(lock)을 획득하려고 대기하는 클라이언트
  - lock: 잠금 종류, 테이블 및 인덱스 이름
  - sql: 잠금을 획득하려고 대기하는 SQL
  - bind: 바인딩된 값
- blocker: 잠금(lock)을 소유하고 있는 클라이언트
  - lock: 잠금 종류, 테이블 및 인덱스 이름
  - sql: 잠금을 획득 중인 SQL
  - bind: 바인딩된 값

위에서 잠금 타임아웃을 유발한 blocker와 잠금을 대기한 waiter를 알 수 있다.

## DEADLOCK

교착 상태(deadlock)가 발생했을 때, cycle에 속해있는 트랜잭션의 잠금(lock) 정보들을 기록한다. 다음은 출력 예이다.

02/02/16 20:56:17.638 - DEADLOCK

hold:

```
client: public@testhost|csql(21541) (Deadlock Victim)
lock: X_LOCK (oid=0|650|5, table=t)
sql: update [t] [t] set [t].[a]=?:0 where [t].[a]=?:1
bind: 3
bind: 1
```

wait:

```
client: public@testhost|csql(21529)
lock: X_LOCK (oid=0|650|5, table=t)
sql: update [t] [t] set [t].[a]=?:0 where [t].[a]=?:1
bind: 4
bind: 1
```

hold:

```
client: public@testhost|csql(21529)
lock: X_LOCK (oid=0|650|6, table=t)
sql: update [t] [t] set [t].[a]=?:0 where [t].[a]=?:1
bind: 5
bind: 2
```

wait:

```
client: public@testhost|csql(21541) (Deadlock Victim)
lock: X_LOCK (oid=0|650|6, table=t)
sql: update [t] [t] set [t].[a]=?:0 where [t].[a]=?:1
bind: 6
bind: 2
```

• hold: 잠금을 소유하고 있는 객체

- client: <DB 사용자>@<응용 클라이언트 호스트 명>|<프로세스 이름>(<프로세스 ID>)  
(Deadlock Victim)
- lock: 잠금 종류, 테이블 이름
- sql: 잠금을 소유하고 있는 SQL
- bind: 바인딩된 값

• wait: 잠금을 대기하고 있는 객체

- client: <DB 사용자>@<응용 클라이언트 호스트 명>|<프로세스 이름>(<프로세스 ID>)  
(Deadlock Victim)
- lock: 잠금 종류, 테이블 이름
- sql: 잠금을 대기하고 있는 SQL
- bind: 바인딩된 값

위에서 교착 상태를 유발한 응용 클라이언트들과 SQL을 확인할 수 있다.

잠금(lock)에 대한 자세한 설명은 [lockdb](#)과 [잠금 프로토콜](#)을 참고한다.

## TEMP\_VOLUME\_EXPAND

일시적 볼륨(temporary volume)이 확장되면 해당 시각을 기록한다. 이를 통해 일시적 볼륨 확장을 유발한 트랜잭션을 확인할 수 있다.

```
06/15/13 18:55:43.458 - TEMP_VOLUME_EXPAND
 client: public@testhost|csql(17540)
 sql: select [x].[a], [x].[b] from [x] [x] where (([x].[a]< ?:0))
 group by [x].[b] order by 1
 bind: 1000
 time: 44
 pages: 24399
```

- client: <DB 사용자>@<응용 클라이언트 호스트 명>|<프로그램 이름>(<프로세스 ID>)
- sql: 일시적 볼륨을 사용하는 SQL. INSERT ... SELECT를 제외한 모든 INSERT 문, DDL 문 등은 DB 서버에 SQL이 전달되지 않기 때문에 EMPTY로 표시된다. SELECT, UPDATE, DELETE 문은 SQL 구문이 표시된다.
- bind: 바인딩된 값
- time: 일시적 볼륨 생성에 소요된 시간(ms)
- pages: 일시적 볼륨 생성에 필요한 페이지의 개수

## 데이터베이스 서버 에러

데이터베이스 서버 프로세스는 에러 발생 시 서버 에러 코드를 사용한다. 서버 에러는 서버 프로세스를 사용하는 모든 작업에서 발생할 수 있다. 예를 들어 질의를 처리하는 프로그램 또는 **cubrid** 유틸리티 사용 중에도 발생할 수 있다.

### 데이터베이스 서버 에러 코드의 확인

- **\$CUBRID/include/error\_code.h** 파일의 **#define ER\_**로 시작하는 정의문은 모두 서버 에러 코드를 나타낸다.
- **\$CUBRID/msg/en\_US** (한글은 ko\_KR.eucKR 혹은 ko\_KR.utf8) **/cubrid.msg** 파일의 “\$set 5 MSGCAT\_SET\_ERROR” 이하 메시지 그룹은 모두 서버 에러 메시지를 나타낸다.

CCI 드라이버를 사용하여 C로 프로그램을 작성할 때는 에러 코드 번호를 직접 사용하는 것보다는 에러 코드 이름을 사용할 것을 권장한다. 예를 들어, 고유 키 위반 시 에러 코드 번호는 -670 혹은 -886이지만 이 번호보다는 **ER\_BTREE\_UNIQUE\_FAILED** 혹은 **ER\_UNIQUE\_VIOLATION\_WITHKEY**을 사용하는 것이 프로그램 가독성을 높이기 때문이다.

하지만 JDBC 드라이버를 사용하여 JAVA로 프로그램을 작성할 때는 dbi.h 파일을 포함할 수 없으므로 에러 코드 번호를 직접 사용하도록 한다. JDBC의 경우 SQLException 클래스의 getErrorCode() 메서드를 통해 에러 번호를 얻을 수 있다.

```
$ vi $CUBRID/include/error_code.h

#define NO_ERROR 0
#define ER_FAILED -1
#define ER_GENERIC_ERROR -1
#define ER_OUT_OF_VIRTUAL_MEMORY -2
#define ER_INVALID_ENV -3
#define ER_INTERRUPTED -4
...
#define ER_LK_OBJECT_TIMEOUT_SIMPLE_MSG -73
#define ER_LK_OBJECT_TIMEOUT_CLASS_MSG -74
#define ER_LK_OBJECT_TIMEOUT_CLASSOF_MSG -75
#define ER_LK_PAGE_TIMEOUT -76
...
#define ER_PT_SYNTAX -493
...
#define ER_BTREE_UNIQUE_FAILED -670
...
#define ER_UNIQUE_VIOLATION_WITHKEY -886
...
#define ER_LK_OBJECT_DL_TIMEOUT_SIMPLE_MSG -966
#define ER_LK_OBJECT_DL_TIMEOUT_CLASS_MSG -967
#define ER_LK_OBJECT_DL_TIMEOUT_CLASSOF_MSG -968
...
```

몇 가지 서버 에러 코드 이름 및 에러 코드 번호, 에러 메시지를 살펴보면 다음과 같다.

| 에러 코드 이름                        | 에러 번호 | 에러 메시지                                                                                                                   |
|---------------------------------|-------|--------------------------------------------------------------------------------------------------------------------------|
| ER_LK_OBJECT_TIMEOUT_SIMPLE_MSG | -73   | Your transaction (index ?, ? @? ?) timed out waiting on ? lock on object ? ? ?. You are waiting for user(s) ? to finish. |
| ER_LK_OBJECT_TIMEOUT_CLASS_MSG  | -74   | Your transaction (index ?, ? @? ?) timed out waiting on ? lock on class ?. You are waiting for user(s) ? to finish.      |

| 에러 코드 이름                               | 에러 번호 | 에러 메시지                                                                                                                                                      |
|----------------------------------------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ER_LK_OBJECT_TIMEOUT_CLAS<br>SOF_MSG   | -75   | Your transaction (index ?, ?<br>@? ?) timed out waiting<br>on ? lock on instance ? ? ?<br>of class ?. You are waiting<br>for user(s) ? to finish.           |
| ER_LK_PAGE_TIMEOUT                     | -76   | Your transaction (index ?, ?<br>@? ?) timed out waiting<br>on ? on page ? ?. You are<br>waiting for user(s) ? to<br>release the page lock.                  |
| ER_PT_SYNTAX                           | -493  | Syntax: ?                                                                                                                                                   |
| ER_BTREE_UNIQUE_FAILED                 | -670  | Operation would have<br>caused one or more unique<br>constraint violations.                                                                                 |
| ER_UNIQUE_VIOLATION_WITHK<br>EY        | -886  | “?” caused unique<br>constraint violation.                                                                                                                  |
| ER_LK_OBJECT_DL_TIMEOUT_SI<br>MPLE_MSG | -966  | Your transaction (index ?, ?<br>@? ?) timed out waiting<br>on ? lock on object ? ? ?<br>because of deadlock. You<br>are waiting for user(s) ? to<br>finish. |
| ER_LK_OBJECT_DL_TIMEOUT_C<br>LASS_MSG  | -967  | Your transaction (index ?, ?<br>@? ?) timed out waiting<br>on ? lock on class ? because<br>of deadlock. You are waiting<br>for user(s) ? to finish.         |

| 에러 코드 이름                                | 에러 번호 | 에러 메시지                                                                                                                                                         |
|-----------------------------------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ER_LK_OBJECT_DL_TIMEOUT_C<br>LASSOF_MSG | -968  | Your transaction (index ?, ?<br>@?) timed out waiting<br>on ? lock on instance ? ? ?<br>of class ? because of<br>deadlock. You are waiting<br>for user(s) ? to |

## 브로커

### 브로커 구동

브로커를 구동하기 위하여 다음과 같이 입력한다. cubrid\_broker.conf 의 브로커 파라미터인 SHARD 가 ON으로 설정된 경우 SHARD 기능이 활성화된다.

```
$ cubrid broker start
@ cubrid broker start
++ cubrid broker start: success
```

이미 브로커가 구동 중이라면 다음과 같은 메시지가 출력된다.

```
cubrid broker start
@ cubrid broker start
++ cubrid broker is running.
```

### ⚠ 경고

Linux 시스템에서 샤드 구동 시 필요한 파일 디스크립터(file descriptor, fd) 개수는 cubrid\_broker.conf에서 설정하는 SHARD\_MAX\_CLIENTS보다 적당히 많은 정도이므로, “ulimit -n”으로 fd의 개수를 제약할 때 SHARD\_MAX\_CLIENTS보다 적당히 크게 설정해야 한다. Linux 시스템의 fd 개수 제약이 SHARD에서 필요한 fd 개수보다 작게 설정된 경우, SHARD 구동에 실패하면서 출력되는 오류 메시지에는 SHARD에서 필요한 fd 개수가 표시된다.

## 브로커 종료

브로커를 종료하기 위하여 다음과 같이 입력한다. cubrid\_broker.conf 의 브로커 파라미터인 SHARD 가 ON으로 설정된 경우 SHARD 기능이 정지된다.

```
$ cubrid broker stop
@ cubrid broker stop
++ cubrid broker stop: success
```

이미 브로커가 종료되었다면 다음과 같은 메시지가 출력된다.

```
$ cubrid broker stop
@ cubrid broker stop
++ cubrid broker is not running.
```

## 브로커 재시작

전체 브로커를 재시작하기 위하여 다음과 같이 입력한다.

```
$ cubrid broker restart
```

## 브로커 상태 확인

**cubrid broker status** 는 여러 옵션을 제공하여, 각 브로커의 처리 완료된 작업 수, 처리 대기중인 작업 수를 포함한 브로커 상태 정보를 확인할 수 있도록 한다. cubrid\_broker.conf의 SHARD 브로커 파라미터가 ON으로 설정된 경우 -c 옵션과 -m 옵션을 사용하여 SHARD에 접속한 클라이언트 또는 SHARD 상태를 확인할 수 있다. 또한 -S 옵션 또는 -P 옵션을 사용하여 shard DB 또는 proxy 별로 구분하여 정보를 출력할 수 있다.

```
cubrid broker status [options] [expr]
```

· *expr*: 브로커 이름의 일부 또는 "SERVICE=ON|OFF"

*expr* 이 명시되면 이름이 *expr* 을 포함하는 브로커에 대한 상태 모니터링을 수행하고, 생략되면 CUBRID 브로커 환경 설정 파일( **cubrid\_broker.conf** )에 등록된 전체 브로커에 대해 상태 모니터링을 수행한다.

*expr* 에 "SERVICE=ON"이 명시되면 구동 중인 브로커의 상태만 출력하며, "SERVICE=OFF"가 명시되면 멈춰있는 브로커의 이름만 출력한다.



cubrid broker status에서 사용하는 [options]는 다음과 같다. 이들 중 -b, -q, -c, -m, -S, -P, -f는 출력할 정보를 정의하는 모니터링 옵션이고, -s, -l, -t는 출력을 제어하는 옵션이다. 이 모든 옵션들은 상호 조합하여 사용할 수 있다. 이 모든 옵션들은 서로 조합하여 사용하는 것이 가능하다.

#### -b

브로커 응용 서버(CAS)에 관한 정보는 포함하지 않고, 브로커에 관한 상태 정보만 출력하며, QPS와 TPS는 브로커가 시작된 후부터의 전체 누적 값을 출력한다. 단, -s 옵션과 사용될 때는 -s에 주어진 시간을 기준으로 QPS와 TPS를 출력한다.

#### -q

작업 큐에 대기 중인 작업을 출력한다.

#### -c

cubrid\_broker.conf의 **SHARD**가 ON으로 설정된 경우 proxy에 접속한 클라이언트 정보를 출력한다.

#### -m

cubrid\_broker.conf의 **SHARD**가 ON으로 설정된 경우 SHARD 상태 및 통계 정보를 출력한다.

#### -S

shard db 별로 구분하여, -b 옵션의 항목 중 NAME, PID, PORT, JQ, #CONNECT를 제외한 나머지를 출력하며, ID, SHARD-Q, #REQUEST를 추가로 출력한다.

#### -p

proxy 별로 구분하여, -b 옵션의 항목 중 NAME, PID, PORT, JQ를 제외한 나머지를 출력하며, ID, SHARD-Q, #RESTART를 추가로 출력한다.

#### -f

브로커가 접속한 DB 및 호스트 정보를 출력한다.

**-b** 옵션과 함께 쓰이는 경우, CAS 정보를 추가로 출력한다. 하지만 -b 옵션에서 나타나는 SELECT, INSERT, UPDATE, DELETE, OTHERS 항목은 제외된다.

**-m** 옵션과 함께 쓰이는 경우, 보다 상세한 SHARD 통계 정보가 출력된다.

**-c** 옵션과 함께 쓰이는 경우, 각 shard proxy 별로 CLIENT-ID, CLIENT-IP, CONN-TIME, LAST-REQ-TIME, LAST-RES-TIME, LAST-REQ-CODE 항목을 추가로 출력한다.

**-P** 옵션과 함께 쓰이는 경우, STMT-POOL-RATIO 항목을 추가로 출력한다. 이 항목은 prepare statement 사용 시 pool에서 statement를 사용하는 비율을 나타낸다.

#### -l SECOND

**-l** 옵션은 **-f** 옵션과만 함께 쓰이며, 클라이언트 Waiting/Busy 상태인 CAS의 개수를 출력할 때 누적 주기(단위: 초)를 지정하기 위해 사용한다. **-l SECOND** 옵션을 생략하면 기본값은 1초이다.

#### -t

화면 출력시 tty mode 로 출력한다. 출력 내용을 리다이렉션하여 파일로 쓸 수 있다.

**-s SECOND**

브로커에 관한 상태 정보를 지정된 시간마다 주기적으로 출력한다. q를 입력하면 명령 프롬프트로 복귀한다.

옵션 및 인수를 입력하지 않으면 전체 브로커 상태 정보를 출력한다.

```
$ cubrid broker status
@ cubrid broker status
% query_editor

ID PID QPS LQS PSIZE STATUS

1 28434 0 0 50144 IDLE
2 28435 0 0 50144 BUSY
3 28436 0 0 50144 CLIENT_WAIT
4 28437 0 0 50140 CLIENT_WAIT
5 28438 0 0 50144 CLOSE_WAIT

% broker1 OFF
```

- % query\_editor: 브로커의 이름
- ID: 브로커 내에서 순차적으로 부여한 CAS의 일련 번호
- PID: 브로커 내 CAS 프로세스의 ID
- QPS: 초당 처리된 질의의 수
- LQS: 초당 처리되는 장기 실행 질의의 수
- PSIZE: CAS 프로세스 크기
- STATUS: CAS의 현재 상태로서, BUSY/IDLE/CLIENT\_WAIT/CLOSE\_WAIT가 있다.
  - IDLE: 데이터베이스와 연결되어 있지 않으며, 아무 작업도 수행 중이지 않은 상태이다.
  - BUSY: SQL을 수행 중이거나, 응용 프로그램에서 요청한 작업을 처리 중인 상태이다.
  - CLIENT\_WAIT: 요청한 작업은 완료되었으나 트랜잭션이 끝나지 않은 상태이다.
  - CLOSE\_WAIT: 트랜잭션이 완료되었지만 연결이 유지된 상태이다.
- % broker1 OFF: broker1의 SERVICE 파라미터가 OFF이다. 따라서, broker1은 구동되지 않는다.

 **참고**

SHARD 상태 확인 시 출력되는 ID 정보는 “(proxy 의 일련번호) - (shard DB의 일련번호) - (shard DB 에 접속하는 응용 서버(CAS)의 일련번호)”로 구성된 수의 조합을 나타낸다.

```
$ cubrid broker status
@ cubrid broker status
% shard1
```

```

 ID PID QPS LQS PSIZE STATUS

 1-1-1 2580 100 3 55968 IDLE
 1-2-1 2581 200 4 55968 IDLE
```

다음은 **-b** 옵션을 사용하여 브로커에 관해 5초 간격으로 상세한 상태 정보를 출력한다. 화면이 5초 간격마다 새로운 상태 정보로 갱신되며, 상태 정보 화면을 벗어나려면 <Q>를 누른다.

```
$ cubrid broker status -b -s 5
@ cubrid broker status
```

```
NAME PID PORT AS JQ TPS QPS
SELECT INSERT UPDATE DELETE OTHERS LONG-T LONG-Q
ERR-Q UNIQUE-ERR-Q #CONNECT #REJECT
=====
=====
=====
* query_editor 13200 30000 5 0 0 0
0 0 0 0 0 0/60.0 0/60.0
0 0 0 0 0
* broker1 13269 33000 5 0 70 60
10 20 10 10 10 0/60.0 0/60.0
30 10 213 1
```

- NAME: 브로커 이름
- PID: 브로커의 프로세스 ID
- PORT: 브로커의 포트 번호
- AS: CAS 개수
- JQ: 작업 큐에서 대기 중인 작업 개수
- TPS: 초당 처리된 트랜잭션의 수(옵션이 “-b -s <sec>”일 때만 해당 구간 계산)
- QPS: 초당 처리된 질의의 수(옵션이 “-b -s <sec>”일 때만 해당 구간 계산)

- SELECT: 브로커 시작 이후 SELECT 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 SELECT 개수로 매번 갱신됨.
- INSERT: 브로커 시작 이후 INSERT 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 INSERT 개수로 매번 갱신됨.
- UPDATE: 브로커 시작 이후 UPDATE 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 UPDATE 개수로 매번 갱신됨.
- DELETE: 브로커 시작 이후 DELETE 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 DELETE 개수로 매번 갱신됨.
- OTHERS: 브로커 시작 이후 SELECT, INSERT, UPDATE, DELETE를 제외한 CREATE, DROP 등의 질의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 질의 개수로 매번 갱신됨.
- LONG-T: LONG\_TRANSACTION\_TIME 시간을 초과한 트랜잭션 개수 / LONG\_TRANSACTION\_TIME 파라미터의 값. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 트랜잭션 개수로 매번 갱신됨.
- LONG-Q: LONG\_QUERY\_TIME 시간을 초과한 질의의 개수 / LONG\_QUERY\_TIME 파라미터의 값. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 질의 개수로 매번 갱신됨.
- ERR-Q: 에러가 발생한 질의의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 에러 개수로 매번 갱신됨. SHARD 파라미터가 ON으로 설정된 경우, proxy에서 에러가 발생하는 경우에도 ERR-Q의 값이 증가한다.
- UNIQUE-ERR-Q: 고유 키 에러가 발생한 질의의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 고유 키 에러 개수로 매번 갱신됨.
- #CONNECT: 브로커 시작 후 응용 클라이언트가 CAS에 접속한 회수
- #REJECT: 브로커 시작 후 ACL에 포함되지 않은 IP로부터 접속하는 응용 클라이언트가 CAS에 접속하는 것을 거부당한 회수. ACL 설정과 관련하여 [브로커 서버 접속 제한](#)를 참고한다.

다음은 **-q** 옵션을 이용하여, broker1을 포함하는 이름을 가진 브로커의 상태 정보를 확인하고 해당 브로커의 작업 큐에 대기 중인 작업 상태를 확인한다. 인자로 broker1을 입력하지 않으면 모든 브로커에 대하여 작업 큐에 대기 중인 작업 리스트가 출력된다.

```
% cubrid broker status -q broker1
@ cubrid broker status
% broker1

ID PID QPS LQS PSIZE STATUS

1 28444 0 0 50144 IDLE
2 28445 0 0 50140 IDLE
3 28446 0 0 50144 IDLE
```

```
4 28447 0 0 50144 IDLE
5 28448 0 0 50144 IDLE
```

다음은 **-s** 옵션을 이용하여 broker1을 포함하는 이름을 가진 브로커의 상태를 주기적으로 모니터링한다. 인자로 broker1을 입력하지 않으면 모든 브로커에 대하여 상태 모니터링이 주기적으로 수행된다. 또한, q를 입력하면 모니터링 화면에서 명령 프롬프트로 복귀한다.

```
% cubrid broker status -s 5 broker1
% broker1
```

```

ID PID QPS LQS PSIZE STATUS

```

```
1 28444 0 0 50144 IDLE
2 28445 0 0 50140 IDLE
3 28446 0 0 50144 IDLE
4 28447 0 0 50144 IDLE
5 28448 0 0 50144 IDLE
```

**-t** 옵션을 이용하여 TPS와 QPS 정보를 파일로 출력한다. 파일로 출력하는 것을 중단하려면 <Ctrl+C>를 눌러서 프로그램을 정지시킨다.

```
% cubrid broker status -b -t -s 1 > log_file
```

다음은 **-f** 옵션을 이용하여 브로커가 연결한 서버/데이터베이스 정보와 응용 클라이언트의 최근 접속 시각, CAS에 접속하는 클라이언트의 IP 주소와 드라이버의 버전 등을 출력한다.

```
$ cubrid broker status -f broker1
@ cubrid broker status
% broker1
```

```

ID PID QPS LQS PSIZE STATUS LAST ACCESS TIME
DB HOST LAST CONNECT TIME CLIENT IP CLIENT
VERSION SQL_LOG_MODE TRANSACTION STIME #CONNECT #RESTART

1 26946 0 0 51168 IDLE 2011/11/16 16:23:42 demodb
localhost 2011/11/16 16:23:40 10.0.1.101
9.2.0.0062 NONE 2011/11/16 16:23:42 0 0
2 26947 0 0 51172 IDLE 2011/11/16 16:23:34
- - - -
0.0.0.0 - -
0 0
3 26948 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
```

```

0.0.0.0 - -
0 0
 4 26949 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0
 5 26950 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0

```

각 칼럼에 대한 설명은 다음과 같다.

- LAST ACCESS TIME: CAS가 구동한 시각 또는 응용 클라이언트의 CAS에 최근 접속한 시각
- DB: CAS의 최근 접속 데이터베이스 이름
- HOST: CAS의 최근 접속 호스트 이름
- LAST CONNECT TIME: CAS의 DB 서버 최근 접속 시각
- CLIENT IP: 현재 CAS에 접속 중인 응용 클라이언트의 IP 주소. 현재 접속 중인 응용 클라이언트가 없으면 0.0.0.0으로 출력
- CLIENT VERSION: 현재 CAS에 접속 중인 응용 클라이언트의 드라이버 버전
- SQL\_LOG\_MODE: CAS의 SQL 로그 기록 모드. 브로커에 설정된 모드와 동일한 경우 “-”으로 출력, 자세한 내용은 **SQL\_LOG**을 참고한다.
- TRANSACTION STIME: 트랜잭션 시작 시간
- #CONNECT: 브로커 시작 후 응용 클라이언트가 CAS에 접속한 회수
- #RESTART: 브로커 시작 후 CAS의 재구동 회수

**-b** 옵션에 **-f** 옵션을 추가하여 AS(T W B Ns-W Ns-B), CANCELED 정보를 추가로 출력한다.

```

// 브로커 상태 정보 실행 시 -f 옵션 추가. -l 옵션으로 N초 동안의 Ns-W, Ns-B
를 출력하도록 초를 설정
% cubrid broker status -b -f -l 2
@ cubrid broker status
NAME PID PSIZE PORT AS(T W B 2s-W 2s-B) JQ TPS QPS LONG-
T LONG-Q ERR-Q UNIQUE-ERR-Q CANCELED ACCESS_MODE SQL_LOG #CONNECT
#REJECT
=====
=====
=====
query_editor 16784 56700 30000 5 0 0 0 0 0 16 29 0/

```

```
60.0 0/60.0 1 1 0 RW ALL
4 1
```

추가된 칼럼에 대한 설명은 다음과 같다.

- AS(T): 실행 중인 CAS의 전체 개수
- AS(W): 현재 클라이언트 대기(Waiting) 상태인 CAS의 개수
- AS(B): 현재 클라이언트 수행(Busy) 상태인 CAS의 개수
- AS(Ns-W): N초 동안 클라이언트 대기(Waiting) 상태였던 CAS의 개수
- AS(Ns-B): N초 동안 클라이언트 수행(Busy) 상태였던 CAS의 개수
- CANCELED: 브로커가 시작된 이후 사용자 인터럽트로 인해 취소된 질의의 개수 (-l N 옵션과 함께 사용하면 N초 동안 누적된 개수).
- ACCESS\_MODE: 브로커가 DB에 접속하는 모드. 브로커모드는 Read Write, Read Only, Standby Only 이렇게 세 가지로 구분된다. 자세한 내용은 [브로커 모드](#)를 참고한다.
- SQL\_LOG: CAS의 SQL 로그 기록 모드. SQL LOG는 ALL, OFF, ERROR, NOTICE, TIMEOUT 이렇게 다섯 가지로 구분된다. 자세한 내용은 [SQL\\_LOG](#)을 참고한다.

-m 옵션을 사용하여 SHARD 상태 및 통계 정보를 출력한다. cubrid\_broker.conf 의 파라미터에 대한 내용은 [브로커 설정](#) 참고한다.

```
$ cubrid broker status -m
@ cubrid broker status
% shard1
MODULAR : 256, LIBRARY_NAME : NOT DEFINED, FUNCTION_NAME : NOT
DEFINED
ACTIVE-PROXY : 1, NUM-NO-HINT-ERR-Q : 0

SHARD STATISTICS
ID NUM-KEY-Q NUM-ID-Q NUM-NO-HINT-Q SUM

 0 1281 0 0 1281
 1 1281 0 0 1281
 2 1281 0 0 1281
 3 1281 0 0 1281

NUM_SHARD_Q
PROXY_ID 1
SHARD_ID

0 1
1 0
```

|   |   |
|---|---|
| 2 | 0 |
| 3 | 2 |

각 칼럼에 대한 설명은 다음과 같다.

- shard1: 브로커의 이름
- MODULAR: **cubrid\_broker.conf**의 **SHARD\_KEY\_MODULR** 파라미터 값
- LIBRARY\_NAME: **cubrid\_broker.conf**의 **SHARD\_KEY\_LIBRARY\_NAME** 파라미터 값
- FUNCTION\_NAME: **cubrid\_broker.conf**의 **SHARD\_KEY\_FUNCTION\_NAME** 파라미터 값
- ACTIVE-PROXY: 실행 중인 proxy 프로세스 수
- NUM-NO-HINT-ERR-Q: 쿼리에 shard hint가 없어서 에러가 발생한 쿼리 개수
- SHARD STATISTICS: shard ID 질의 정보
  - ID: shard DB 일련번호(shard ID)
  - NUM-KEY-Q: shard key가 포함된 질의 요청 개수
  - NUM-ID-Q: shard ID가 포함된 질의 요청 개수
  - NUM-NO-HINT-Q: SHARD\_IGNORE\_HINT가 설정된 경우 hint 없이 load balancing되어 처리된 요청 개수
  - SUM: NUM-KEY-Q + NUM-ID-Q
- NUM\_SHARD\_Q: SHARD-Q 에 대기 중인 질의 실행 요청 개수
  - PROXY\_ID: proxy 의 일련번호
  - SHARD\_ID: shard DB 의 일련번호

SHARD-Q는 Shard Waiting Queue를 줄인 말이다. SHARD proxy 프로세스가 질의 실행을 요청했으나 이를 처리할 SHARD CAS 프로세스가 없는 경우 질의 실행 요청은 SHARD-Q에서 잠시 대기한다. 이 값이 커질수록 질의 실행을 바로 처리하지 못하고 대기중인 경우가 많아진다는 의미이므로, MAX\_NUM\_APPL\_SERVER의 개수를 더 크게 설정할 것을 고려한다.

다음은 **-m** 옵션과 **-f** 옵션을 이용하여 좀 더 상세한 SHARD 통계 정보를 출력한다. cubrid\_broker.conf 의 파라미터에 대한 내용은 **브로커 설정**을 참고한다.

```
$ cubrid broker status -m -f
@ cubrid broker status
% shard1
MODULAR : 256, LIBRARY_NAME : NOT DEFINED, FUNCTION_NAME : NOT
DEFINED
```



```
SHARD : 0 [HostA] [shard1], 1 [HostB] [shard1], 2 [HostC] [shard1],
3 [HostD] [shard1]
ACTIVE-PROXY : 1, NUM-NO-HINT-ERR-Q : 0
```

#### SHARD STATISTICS

| ID | NUM-KEY-Q | NUM-ID-Q | NUM-NO-HINT-Q | SUM  |
|----|-----------|----------|---------------|------|
| 0  | 2309      | 0        | 0             | 2309 |
| 1  | 2309      | 0        | 0             | 2309 |
| 2  | 2309      | 0        | 0             | 2309 |
| 3  | 2309      | 0        | 0             | 2309 |

```
NUM_SHARD_Q
```

```
PROXY_ID 1
```

```
SHARD_ID
```

|   |   |
|---|---|
| 0 | 1 |
| 1 | 0 |
| 2 | 0 |
| 3 | 2 |

#### RANGE STATISTICS : user\_no

| MIN ~ | MAX : | SHARD | NUM-Q |
|-------|-------|-------|-------|
| 0 ~   | 31 :  | 0     | 1157  |
| 32 ~  | 63 :  | 1     | 1157  |
| 64 ~  | 95 :  | 2     | 1157  |
| 96 ~  | 127 : | 3     | 1157  |
| 128 ~ | 159 : | 0     | 1152  |
| 160 ~ | 191 : | 1     | 1152  |
| 192 ~ | 223 : | 2     | 1152  |
| 224 ~ | 255 : | 3     | 1152  |

```
DB Alias : shard1 [USER : shard, PASSWD : shard123]
```

추가된 칼럼에 대한 설명은 다음과 같다.

- SHARD: proxy 내의 shard DB 정보
  - 0: shard DB 일련번호(shard ID)
  - [HostA]: shard 접속 정보
  - [shard1]: 실제 DB 이름
- RANGE STATISTICS: shard key 질의 정보
  - user\_no: shard key 이름
  - MIN: shard key 최소 범위

- MAX: shard key 최대 범위
- SHARD: shard DB 일련번호(shard ID)
- NUM-Q: shard key가 포함된 질의 요청 수

-c 옵션을 사용하여 proxy에 접속한 클라이언트 정보를 출력한다.

```
$ cubrid broker status -c
@ cubrid broker status
% shard1(0), MAX-CLIENT : 50, CUR-CLIENT : 0
% shard1(1), MAX-CLIENT : 50, CUR-CLIENT : 0
```

- MAX-CLIENT: 해당 proxy에 접속할 수 있는 최대 응용 클라이언트 개수
- CUR-CLIENT: 해당 proxy에 접속되어 있는 응용 클라이언트 개수

-c 옵션에 -f 옵션을 추가하면 좀더 상세한 클라이언트 정보가 출력된다.

```
$ cubrid broker status -c -f
@ cubrid broker status
% shardqa(0), MAX-CLIENT : 50, CUR-CLIENT : 0

CLIENT-ID CLIENT-IP CONN-TIME LAST-
REQ-TIME LAST-RES-TIME LAST-REQ-CODE

 2 127.0.0.1 2014/01/21 18:07:29 2014/01/21
18:07:56 2014/01/21 18:07:56 2
% shardqa(1), MAX-CLIENT : 50, CUR-CLIENT : 0

CLIENT-ID CLIENT-IP CONN-TIME LAST-
REQ-TIME LAST-RES-TIME LAST-REQ-CODE


```

추가된 칼럼에 대한 설명은 다음과 같다.

- CLIENT-ID: proxy 내에서 순차적으로 부여한 클라이언트 일련 번호
- CLIENT-IP: 클라이언트 IP 주소
- CONN-TIME: proxy에 접속한 시각
- LAST-REQ-TIME: proxy에 마지막으로 질의를 요청한 시각
- LAST-RES-TIME: proxy로부터 마지막으로 응답을 받은 시각

• LAST-REQ-CODE: 가장 마지막에 수행된 함수의 코드. 주요 코드에 해당하는 함수는 다음과 같다.

- 0: end\_tran(트랜잭션 종료)
- 1: prepare
- 2: execute
- 7: fetch

다음은 **-S** 옵션을 사용하여 shard DB별 정보를 출력한다.

```
$ cubrid broker status -S
@ cubrid broker status
% shard1
 SHARD_ID AS SHARD-Q TPS QPS SELECT INSERT
UPDATE DELETE OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-
Q #REQUEST
=====
=====
===
 0 2 0 3200 3772 956 960
928 928 0 0/60.0 0/60.0 700
0 6978
 1 2 0 3200 3776 960 960
928 928 0 0/60.0 0/60.0 704
0 6983
 2 2 0 3200 3762 960 960
928 914 0 0/60.0 0/60.0 690
0 6968
 3 2 0 3200 3776 960 960
928 928 0 0/60.0 0/60.0 704
0 6983
```

추가된 칼럼에 대한 설명은 다음과 같다.

- SHARD\_ID: shard의 index. 0부터 시작.
- SHARD-Q: 해당 shard의 SHARD-Q에 대기 중인 질의 실행 요청 개수(**SHARD-Q** 참고)
- #REQUEST: 해당 shard에 속한 CAS가 응용 클라이언트에게 받은 전체 요청 개수(요청은 질의 수행 뿐 아니라 연결 요청 등 모든 요청을 포함)

**-S** 옵션에 **-f** 옵션을 추가하면 AS 항목이 (T W B 1s-W 1s-B)으로 나뉘어져 상세히 출력된다. AS 항목에 대한 상세 설명은 **AS**를 참고한다.

다음은 **-P** 옵션을 사용하여 proxy별 정보를 출력한다.

```
$ cubrid broker status -P
% shard1
 PROXY_ID AS SHARD-Q TPS QPS SELECT INSERT
UPDATE DELETE OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-
Q #CONNECT #REJECT #RESTART
=====
=====
=====
 1 4 0 22174 26160 26160 0
0 0 0 0/60.0 0/60.0 5256
0 165 0 0
 2 4 0 35257 37903 23599 5152
4576 4576 0 0/60.0 0/60.0 4300
0 264 1 0
```

**-P** 옵션을 사용할 때 **-b** 옵션에 비해 추가로 출력되는 칼럼 정보는 다음과 같다.

- PROXY\_ID: proxy의 index. 1부터 시작.
- SHARD-Q: proxy의 SHARD-Q 에 대기 중인 질의 실행 요청 개수 (SHARD-Q 참고)
- #CONNECT: 응용 클라이언트가 proxy에 접속을 시도한 회수
- #REJECT: ACL에 포함되지 않은 IP로부터 접속하는 응용 클라이언트가 해당 proxy에 접속하는 것을 거부당한 회수. ACL 설정과 관련하여 **브로커 서버 접속 제한**를 참고한다.
- #RESTART: proxy가 재시작한 횟수

**-S** 옵션에 **-f** 옵션을 추가하면 AS 항목이 (T W B 1s-W 1s-B)으로 나뉘어져 상세히 출력되고, STMT-POOL-RATIO 항목이 추가된다. AS 항목에 대한 상세 설명은 **AS**를 참고한다.

```
$ cubrid broker status -P -f
% shard1
 PROXY_ID AS(T W B 1s-W 1s-B) SHARD-Q TPS QPS
LONG-T LONG-Q ERR-Q UNIQUE-ERR-Q #CONNECT #REJECT #RESTART
STMT-POOL-RATIO (%)
=====
=====
=====
 1 4 0 0 0 0 0 0 0
0/60.0 0/60.0 0 0 165 0
0 -
 2 4 0 0 0 0 0 0
0/60.0 0/60.0 0 0 264 1
0 -
```

추가된 칼럼에 대한 설명은 다음과 같다.

- STMT-POOL-RATIO: prepare statement 사용 시 pool에서 statement를 사용하는 비율

**-b, -S, -P** 옵션을 동시에 사용하면 다음과 같이 출력된다.

```
$ cubrid broker status -b -S -P
```

```
@ cubrid broker status
```

| NAME         | PID      | PORT    | AS     | JQ | TPS    | QPS    | SELECT |
|--------------|----------|---------|--------|----|--------|--------|--------|
| INSERT       | UPDATE   | DELETE  | OTHERS |    | LONG-T | LONG-Q | ERR-Q  |
| UNIQUE-ERR-Q | #CONNECT | #REJECT |        |    |        |        |        |

```
=====
=====
=====
```

|          |       |       |      |        |        |        |       |
|----------|-------|-------|------|--------|--------|--------|-------|
| * shard1 | 10204 | 56001 | 8    | 0      | 57431  | 64063  | 49759 |
| 5152     | 4576  | 4576  | 0    | 0/60.0 | 0/60.0 |        |       |
| 9556     |       | 0     | 429  | 0      |        |        |       |
| * shard2 | 10221 | 56002 | 8    | 0      | 51913  | 58979  |       |
| 49844    | 0     | 4687  | 4448 | 0      | 0/60.0 | 0/60.0 |       |
| 9862     |       | 0     | 429  | 2      |        |        |       |

```
% broker1 OFF
```

```
<SHARD INFO>
```

```
% shard1
```

| SHARD_ID | AS       | SHARD-Q | TPS    | QPS    | SELECT | INSERT  |
|----------|----------|---------|--------|--------|--------|---------|
| UPDATE   | DELETE   | OTHERS  | LONG-T | LONG-Q | ERR-Q  | UNIQUE- |
| ERR-Q    | #REQUEST |         |        |        |        |         |

```
=====
=====
=====
```

|      |      |   |        |        |       |       |      |
|------|------|---|--------|--------|-------|-------|------|
|      | 0    | 2 | 0      | 14464  | 16165 | 12613 | 1312 |
| 1120 | 1120 | 0 | 0/60.0 | 0/60.0 |       |       |      |
| 2437 |      | 0 | 30645  |        |       |       |      |
|      | 1    | 2 | 0      | 14464  | 15926 | 12310 | 1248 |
| 1184 | 1184 | 0 | 0/60.0 | 0/60.0 |       |       |      |
| 2198 |      | 0 | 30403  |        |       |       |      |
|      | 2    | 2 | 0      | 14464  | 16347 | 12795 | 1312 |
| 1120 | 1120 | 0 | 0/60.0 | 0/60.0 |       |       |      |
| 2619 |      | 0 | 30824  |        |       |       |      |
|      | 3    | 2 | 0      | 14039  | 15625 | 12041 | 1280 |
| 1152 | 1152 | 0 | 0/60.0 | 0/60.0 |       |       |      |
| 2302 |      | 0 | 29681  |        |       |       |      |

```
% shard2
```

| SHARD_ID | AS       | SHARD-Q | TPS    | QPS    | SELECT | INSERT  |
|----------|----------|---------|--------|--------|--------|---------|
| UPDATE   | DELETE   | OTHERS  | LONG-T | LONG-Q | ERR-Q  | UNIQUE- |
| ERR-Q    | #REQUEST |         |        |        |        |         |

```
=====
=====
=====
```

|  |   |   |   |       |       |       |   |
|--|---|---|---|-------|-------|-------|---|
|  | 0 | 2 | 0 | 13085 | 14884 | 12580 | 0 |
|--|---|---|---|-------|-------|-------|---|

```

1184 1120 0 0/60.0 0/60.0
2503 0 27985
 1 2 0 13056 14808 12507 0
1181 1120 0 0/60.0 0/60.0
2456 0 27878
 2 2 0 13056 14743 12453 0
1170 1120 0 0/60.0 0/60.0
2391 0 27812
 3 2 0 12716 14544 12304 0
1152 1088 0 0/60.0 0/60.0
2512 0 27273

% broker1 OFF

<PROXY INFO>
% shard1
 PROXY_ID AS SHARD-Q TPS QPS SELECT INSERT
UPDATE DELETE OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-Q
#CONNECT #REJECT #RESTART
=====
=====
=====
 1 4 0 22174 26160 26160 0
0 0 0 0/60.0 0/60.0 5256 0
165 0 0
 2 4 0 35257 37903 23599 5152
4576 4576 0 0/60.0 0/60.0 4300
0 264 1 0
% shard2
 PROXY_ID AS SHARD-Q TPS QPS SELECT INSERT
UPDATE DELETE OTHERS LONG-T LONG-Q ERR-Q UNIQUE-ERR-Q
#CONNECT #REJECT #RESTART
=====
=====
=====
 1 4 0 21590 25586 25586 0
0 0 0 0/60.0 0/60.0 5266 0
165 0 0
 2 4 0 30323 33393 24258 0
4687 4448 0 0/60.0 0/60.0 4596
0 264 1 0
% broker1 OFF

```

## 브로커 서버 접속 제한

브로커에 접속하는 응용 클라이언트를 제한하려면 **cubrid\_broker.conf**의 **ACCESS\_CONTROL** 파라미터 값을 ON으로 설정하고, **ACCESS\_CONTROL\_FILE** 파라미터 값에 접속을 허용하는 사용자와 데이터베이스 및 IP 목록을 작성한 파일 이름을 입력한다. ACCESS\_CONTROL\_FILE에 해당 브로커 이름이 없는 경우 그 브로커로의 모든 접속은 **ACCESS\_CONTROL\_DEFAULT\_POLICY** 파라미터에 의해 결정된

다. 이 파라미터의 값이 DENY 인 경우 접속이 제한되며, ALLOW 인 경우 접속이 허용된다.

**ACCESS\_CONTROL** 브로커 파라미터의 기본값은 **OFF**, **ACCESS\_CONTROL\_DEFAULT\_POLICY**의 기본값은 **DENY**이다. **ACCESS\_CONTROL**, **ACCESS\_CONTROL\_FILE**, **ACCESS\_CONTROL\_DEFAULT\_POLICY** 파라미터는 공통 파라미터가 지정되는 [broker] 섹션 아래에 작성해야 한다.

**ACCESS\_CONTROL\_FILE**의 형식은 다음과 같다.

```
[%<broker_name>]
<db_name>:<db_user>:<ip_list_file>
...
```

- <broker\_name>: 브로커 이름. **cubrid\_broker.conf**에서 지정한 브로커 이름 중 하나이다.
- <db\_name>: 데이터베이스 이름. \*로 지정하면 모든 데이터베이스를 허용한다.
- <db\_user>: 데이터베이스 사용자 ID. \*로 지정하면 모든 데이터베이스 사용자 ID를 허용한다.
- <ip\_list\_file>: 접속 가능한 IP 목록을 저장한 파일의 이름. ip\_list\_file1, ip\_list\_file2, ...와 같이 파일 여러 개를 쉼표(,)로 구분하여 지정할 수 있다.

#### 참고

Windows에서 절대 경로명 사용시 드라이브명을 반드시 명시해야 한다 (예, C:\CUBRID\CONF\CUBRID\_ACL.CONF).

브로커별로 [%<broker\_name>]과 <db\_name>:<db\_user>:<ip\_list\_file>을 추가 지정할 수 있으며, 같은 <db\_name>, 같은 <db\_user>에 대해 별도의 라인으로 추가 지정할 수 있다. IP 목록은 하나의 브로커 내에서 <db\_name>:<db\_user> 별로 최대 256 라인까지 작성될 수 있다.

ip\_list\_file의 작성 형식은 다음과 같다.

```
<ip_addr>
...
```

- <ip\_addr>: 접근을 허용할 IP 명. 뒷자리를 \*로 입력하면 뒷자리의 모든 IP를 허용한다.

**ACCESS\_CONTROL** 값이 ON인 경우에도, localhost 의 접속은 허용된다. (즉, ip\_list\_file에 localhost ip인 127.0.01를 지정해도 무시됨)

**ACCESS\_CONTROL\_FILE**에 지정되지 않은 브로커 접속 제한 방식.

- localhost에서만 접속 허용. (기본)
- **ACCESS\_CONTROL\_DEFAULT\_POLICY**를 ALLOW로 설정하면 모든 접속 허용.

브로커 구동 시 **ACCESS\_CONTROL\_FILE** 및 `ip_list_file` 분석에 실패하는 경우 브로커는 구동되지 않는다.

```
cubrid_broker.conf
[broker]
MASTER_SHM_ID =30001
ADMIN_LOG_FILE =log/broker/cubrid_broker.log
ACCESS_CONTROL =ON
ACCESS_CONTROL_FILE =/home1/cubrid/access_file.txt
[%QUERY_EDITOR]
SERVICE =ON
BROKER_PORT =30000
.....
```

다음은 **ACCESS\_CONTROL\_FILE**의 한 예이다. 파일 내에서 사용하는 \*은 모든 것을 나타내며, 데이터베이스 이름, 데이터베이스 사용자 ID, 접속을 허용하는 IP 리스트 파일 내의 IP에 대해 지정할 때 사용할 수 있다.

```
[%QUERY_EDITOR]
dbname1:dbuser1:READIP.txt
dbname1:dbuser2:WRITEIP1.txt,WRITEIP2.txt
*:dba:READIP.txt
*:dba:WRITEIP1.txt
*:dba:WRITEIP2.txt

[%BROKER2]
dbname:dbuser:iplist2.txt

[%BROKER3]
dbname:dbuser:iplist2.txt

[%BROKER4]
dbname:dbuser:iplist2.txt
```

위의 예에서 지정한 브로커는 QUERY\_EDITOR, BROKER2, BROKER3, BROKER4이다.

QUERY\_EDITOR 브로커는 다음과 같은 응용의 접속 요청만을 허용한다.

- `dbname1`에 `dbuser1`으로 로그인하는 사용자가 `READIP.txt`에 등록된 IP에서 접속
- `dbname1`에 `dbuser2`로 로그인하는 사용자가 `WRITEIP1.txt`나 `WRITEIP2.txt`에 등록된 IP에서 접속
- 모든 데이터베이스에 **DBA**로 로그인하는 사용자가 `READIP.txt`나 `WRITEIP1.txt` 또는 `WRITEIP2.txt`에 등록된 IP에서 접속

다음은 `ip_list_file`에서 허용하는 IP를 설정하는 예이다.



```
192.168.1.25
192.168.*
192.*
*
```

위의 예에서 지정한 IP를 보면 다음과 같다.

- 첫 번째 줄의 설정은 192.168.1.25을 허용한다.
- 두 번째 줄의 설정은 192.168 로 시작하는 모든 IP를 허용한다.
- 세 번째 줄의 설정은 192으로 시작하는 모든 IP를 허용한다.
- 네 번째 줄의 설정은 모든 IP를 허용한다.

이미 구동되어 있는 브로커에 대해서는 다음 명령어를 통해 설정 파일을 다시 적용하거나 현재 적용 상태를 확인할 수 있다.

브로커에서 허용하는 데이터베이스, 데이터베이스 사용자 ID, IP를 설정한 후 변경된 내용을 서버에 적용하려면 다음 명령어를 사용한다.

```
cubrid broker acl reload [<BR_NAME>]
```

- <BR\_NAME>: 브로커 이름. 이 값을 지정하면 특정 브로커만 변경 내용을 적용할 수 있으며, 생략하면 전체 브로커에 변경 내용을 적용한다.

#### 참고

현재 접속된 연결은 reload 명령을 이용한 access control list의 재구성에 영향을 받지 않는다. 즉, reload에 의한 재구성은 새로운 연결에만 적용되며, 재구성 전 접속된 연결에 변경된 access control list를 적용하기 위해서는 해당 브로커를 재시작해야 한다.

현재 구동 중인 브로커에서 허용하는 데이터베이스, 데이터베이스 사용자 ID, IP 목록, 최종 접속 시간을 화면에 출력하려면 다음 명령어를 사용한다.

```
cubrid broker acl status [<BR_NAME>]
```

- <BR\_NAME>: 브로커 이름. 이 값을 지정하면 특정 브로커의 설정을 출력할 수 있으며, 생략하면 전체 브로커의 설정을 출력한다.

다음은 출력 화면의 예이다.

```
$ cubrid broker acl status
ACCESS_CONTROL=ON
ACCESS_CONTROL_FILE=access_file.txt
ACCESS_CONTROL_DEFAULT_POLICY=ALLOW

[%broker1]
demodb:dba:iplist1.txt
 CLIENT IP LAST ACCESS TIME
=====
192.20.30.40
192.168.1.0 2023-11-07 15:19:14
192.168.2.0
192.168.3.0
192.169.* 2023-11-07 15:20:50

testdb:dba:iplist2.txt
 CLIENT IP LAST ACCESS TIME
=====
* 2013-11-08 10:10:12

[%broker2]
```

## 브로커 로그

허용되지 않는 IP에서 접근하면 다음과 같은 로그가 남는다.

- ACCESS\_LOG

```
1 192.10.10.10 - - 1288340944.198 1288340944.198 2010/10/29
17:29:04 ~ 2010/10/29 17:29:04 14942 - -1 db1 dba : rejected
```

- SQL LOG

```
10/29 10:28:57.591 (0) CLIENT IP 192.10.10.10 10/29 10:28:
57.592 (0) connect db db1 user dba url jdbc:cubrid:
192.10.10.10:30000:db1:: - rejected
```

## 참고

데이터베이스 서버에서의 접속 제한을 위해서는 **데이터베이스 서버 접속 제한** 을 참고한다.

## 패킷 암호화

개방형 네트워크에서 데이터베이스 서버와 클라이언트 간의 통신은 제 3자에게 유출될 수 있으며, 부정 사용될 수 있다. 안전하지 않은 통신 환경을 이용한 정보 접근 과정에서 정보 유출을 방지하기 위해서는 송수신되는 모든 정보가 **암호화** 되어야 한다. 큐브리드 브로커는 보안 모드로 설정이 가능하며, 이 경우 데이터베이스 서버와 클라이언트 간의 모든 데이터는 암호화되어 송수신된다.

큐브리드는 **TLS** (Transport Layer Security) 프로토콜을 이용한 암호화 기능을 제공한다. TLS는 암호화 기능 뿐만 아니라 데이터의 변조와 손실을 감지하는 기능을 포함하고 있어서 클라이언트와 서버에게 보안이 강화된 신뢰할 수 있는 통신 수단을 제공한다. 큐브리드는 이러한 기능을 제공하기 위해 **OpenSSL** 을 채택하였다.

큐브리드 브로커는 보안 모드 (**SSL = ON**) 또는 비보안 모드 (**SSL = OFF**)로 구성할 수 있으며, 이러한 모드는 **cubrid\_broker.conf** 의 **SSL** 파라미터의 값에 따라서 결정된다. 브로커의 SSL 파라미터의 값이 변경된 경우, 브로커를 재시작하여야 한다. 브로커가 보안 모드인 경우, **jdbc** 응용 프로그램과 같은 클라이언트들도 보안 모드로 설정되어 접속되어야 한다, 그렇지 않으면 연결 요청은 브로커에 의해서 거부된다. 반대의 경우도 마찬가지이다. 비보안 모드의 브로커에 보안 모드의 클라이언트 접속 요청도 거부된다.

**cubrid\_broker.conf** 에 SSL 파라미터가 정의되지 않은 경우, 브로커는 비보안 모드로 동작한다 (**SSL = OFF** 가 기본 모드). 아래의 예는 브로커 '**query\_editor**' 를 보안 모드로 설정한 예이다.

```
cubrid_broker.conf
[query_editor]
SERVICE =ON
SSL =ON
BROKER_PORT =30000
....
```

## 인증서 (Certificate) 와 개인키 (Private Key)

**SSL** 은 대칭형 (**symmetric**) 키를 이용하여 송수신 데이터를 암호화 한다 (클라이언트와 서버가 같은 **세션키** 를 공유하여 암호/복호함). 매 통신 세션에서 새로이 생성되는 세션키를 클라이언트와 서버가 암호화한 형태로 교환하기 위해서 비 대칭 (**asymmetric**) 암호화 알고리즘 을 사용하며, 이를 위해서 서버의 공개키와 개인키가 필요하다.

공개키는 인증서에 포함되어 있으며, 인증서와 개인키는 \$CUBRID/conf 디렉터리에 있으며 각각의 파일명은 '**cas\_ssl\_cert.crt**' 와 '**cas\_ssl\_cert.crt**' 이다. 이 인증서는 OpenSSL의 명령어 도구를 이용하여 생성된 것이며 'self-signed' 형태의 인증서이다.

사용자가 원하는 경우 **IdenTrust** 나 **DigiCert** 와 같은 공인 인증기관에서 발급받은 인증서로 대체도 가능하다. 또는 OpenSSL 명령어 도구를 이용하여 개인키/인증서를 새로 생성하여 대체하는 것도 가능하다. 아래의 예는 OpenSSL 명령어 도구를 이용하여 개인키, 인증서를 생성하는 것이다.

```
$ openssl genrsa -out my_cert.key
2048
```

```
2048 bit 크기의
```

```
RSA 개인키 생성
$ openssl req -new -key my_cert.key -out
my_cert.csr # 인증요청서 CSR
(Certificate Signing Request)
$ openssl x509 -req -days 365 -in my_cert.csr -signkey my_cert.key -
out my_cert.crt # 1년 유효한 인증서 생성
```

위에서 생성된 **my\_cert.key** 와 **my\_cert.crt** 를 각각 `$CUBRID/conf/cas_ssl_cert.key`와 `$CUBRID/conf/cas_ssl_cert.crt`로 대체하면 된다.

## 특정 브로커 관리

*broker1*만 구동하기 위하여 다음과 같이 입력한다. 단, *broker1*은 이미 공유 메모리에 설정된 브로커이다.

```
% cubrid broker on broker1
```

만약, *broker1*이 공유 메모리에 설정되지 않은 상태라면 다음과 같은 메시지가 출력된다.

```
% cubrid broker on broker1
Cannot open shared memory
```

*broker1*만 종료하기 위하여 다음과 같이 입력한다. 이 때, *broker1*의 서비스 풀을 함께 제거할 수 있다.

```
% cubrid broker off broker1
```

브로커 리셋 기능은 HA에서 failover 등으로 브로커 응용 서버(CAS)가 원하지 않는 데이터베이스 서버에 연결되었을 때, 기존 연결을 끊고 새로 연결할 수 있도록 한다. 예를 들어 Read Only 브로커가 액티브 서버와 연결된 후에는 스탠바이 서버가 연결이 가능한 상태가 되더라도 자동으로 스탠바이 서버와 재연결하지 않으며, **cubrid broker reset** 명령을 통해서만 기존 연결을 끊고 새로 스탠바이 서버와 연결할 수 있다.

*broker1*을 리셋하려면 다음과 같이 입력한다.

```
% cubrid broker reset broker1
```

## 브로커 파라미터의 동적 변경

브로커 구동과 관련된 파라미터는 브로커 환경 설정 파일( **cubrid\_broker.conf** )에서 설정할 수 있다. 그 밖에, **broker\_changer** 유틸리티를 이용하여 구동 중에만 한시적으로 일부 브로커 파라미터

를 동적으로 변경할 수 있다. 브로커 파라미터 설정 및 동적으로 변경 가능한 브로커 파라미터 등 기타 자세한 내용은 **브로커 설정**을 참조한다.

브로커 구동 중에 브로커 파라미터를 변경하기 위한 **broker\_changer** 유틸리티의 구문은 다음과 같다. *broker\_name*에는 구동 중인 브로커 이름을 입력하면 되고 *parameter*는 동적으로 변경할 수 있는 브로커 파라미터에 한정된다. 변경하고자 하는 파라미터에 따라 *value*가 지정되어야 한다. 브로커 응용 서버 식별자(*cas\_id*)를 지정하여 특정 브로커 응용 서버(CAS)에만 변경을 적용할 수도 있다.

*cas\_id*는 **cubrid broker status** 명령에서 출력되는 ID이다.

```
broker_changer <broker_name> [<cas_id>] <conf_name> <conf_value>
```

구동 중인 브로커에서 SQL 로그가 기록되도록 **SQL\_LOG** 파라미터를 ON으로 설정하기 위하여 다음과 같이 입력한다. 이와 같은 파라미터의 동적 변경은 브로커 구동 중일 때만 한시적으로 효력이 있다.

```
% broker_changer query_editor sql_log on
OK
```

HA 환경에서 브로커의 **ACCESS\_MODE**를 Read Only로 변경하고 해당 브로커를 자동으로 reset하기 위하여 다음과 같이 입력한다.

```
% broker_changer broker_m access_mode ro
OK
```

### 참고

Windows Vista 이상 버전에서 cubrid 유틸리티를 사용하여 서비스를 제어하려면 명령 프롬프트 창을 관리자 권한으로 구동한 후 사용하는 것을 권장한다. 자세한 내용은 **CUBRID 유틸리티**를 참고한다.

## 브로커 설정 정보 확인

**cubrid broker info**는 현재 “실행 중”인 브로커 파라미터의 설정 정보(cubrid\_broker.conf)를 출력한다. **broker\_changer** 명령에 의해 브로커 파라미터의 설정 정보가 동적으로 변경될 수 있는데, **cubrid broker info** 명령으로 동작 중인 브로커의 설정 정보를 확인할 수 있다.

```
% cubrid broker info
```

참고로 현재 “실행 중”인 시스템 파라미터의 설정 정보(cubrid.conf)를 확인하려면 **cubrid paramdump database\_name** 명령을 사용한다. **SET SYSTEM PARAMETERS** 구문에 의해 시스템 파라미터의 설정 정보가 동적으로 변경될 수 있는데, **cubrid broker info** 명령으로 동작 중인 시스템의 설정 정보를 확인할 수 있다.

## CUBRID SHARD ID 확인

**cubrid broker getid**는 특정 키가 어느 샤드 DB에 속하는지 알고 싶을 때 사용하는 명령으로, shard key에 대한 SHARD ID를 출력한다.

```
cubrid broker getid -b <broker-name> [-f] shard-key
```

- -b <broker-name>: 브로커 이름
- -f: 상세 정보 출력
- shard-key: shard key

다음은 shard1 브로커에서 키 1에 대한 SHARD ID를 출력하는 예이다.

```
$ cubrid broker getid -b shard1 1
@ cubrid broker getid
% shard1
SHARD_ID : 0, SHARD_KEY: 1
```

다음은 -f 옵션을 사용하여 상세 정보를 출력하는 예이다.

```
$ cubrid broker getid -b shard1 -f 1
@ cubrid broker getid
% shard1
SHARD_ID : 0, SHARD_KEY : 1, KEY_COLUMN : student_no
MODULAR : 256, LIBRARY_NAME : NOT DEFINED, FUNCTION_NAME : NOT
DEFINED
RANGE STATISTICS : student_no
 MIN ~ MAX : SHARD

 0 ~ 31 : 0

SHARD CONNECTION :
 SHARD_ID DB NAME CONNECTION_INFO

 0 shard1 192.168.10.1
```

## 브로커와 DB 간 연결 테스트

**cubrid broker test**는 지정한 브로커와 접속하는 DB에 사용자가 정의한 질의문을 수행해 보는 명령이다. 샤드 기능이 활성화되면 모든 SHARD DB에 질의를 수행해 본다. 질의 수행 후 트랜잭션은 롤백된다. 이 명령어를 통해 지정한 브로커에 접속하는 모든 SHARD DB에 질의를 수행하면 각 SHARD DB에 대한 질의 성공 여부를 확인할 수 있고, SHARD HASH 기능을 설정한 경우 입력한 질의가 어떤 SHARD DB에서 수행되었는지 확인할 수 있다.

```
cubrid broker test <broker_name> [-D <db_name>] [-u <db_user>] [-p <db_password>] {-c <query> | -i <input_file>} [-o <output_file>] [-s] [-v]
```

- db\_name: DB 이름
- db\_user: DB 사용자 계정
- db\_password: DB 사용자 암호
- query: 질의문
- input\_file: 입력할 질의문을 저장한 파일
- output\_file: 결과를 저장할 파일

**cubrid broker test**에서 사용하는 옵션은 다음과 같다.

### -D DB\_NAME

테스트 대상 DB 이름을 지정한다. 이 옵션이 생략될 때 cubrid\_broker.conf의 SHARD 파라미터 값이 ON이면 SHARD\_DB\_NAME 파라미터의 값이 사용된다. SHARD 파라미터의 값이 OFF이면 에러가 발생한다.

### -u DB\_USER

테스트 대상 DB 사용자 계정을 지정한다. 이 옵션이 생략될 때 cubrid\_broker.conf의 SHARD 파라미터 값이 ON이면 SHARD\_DB\_USER 파라미터의 값이 사용된다. SHARD 파라미터의 값이 OFF이면 "public"이 입력된다.

### -p DB\_PASSWORD

테스트 대상 DB 사용자 계정의 암호를 지정한다. cubrid\_broker.conf의 SHARD 파라미터 값이 ON일 때 이 옵션이 생략되면 SHARD\_DB\_PASSWORD 파라미터의 값이 사용된다. SHARD 파라미터의 값이 OFF이면 빈 문자열("")이 입력된다.

### -c QUERY

질의 문자열을 지정한다. 질의를 지정하기 위해 **-c** 또는 **-i** 옵션이 사용될 수 있다. **-c** 옵션과 **-i** 옵션이 생략되면 브로커와 DB 간 연결 여부만 출력한다.

### -i FILE\_NAME

입력할 질의들을 저장한 파일을 지정한다. 질의를 지정하기 위해 **-c** 또는 **-i** 옵션이 사용될 수 있다. **-c** 옵션과 **-i** 옵션이 생략되면 브로커와 DB 간 연결 여부만 출력한다.

#### **-o FILE\_NAME**

화면에 출력되는 수행 결과를 저장할 파일 이름을 지정한다. 이 옵션이 생략되면 수행 결과를 화면에만 출력한다.

#### **-s**

SHARD 힌트가 포함된 질의는 해당 SHARD DB에서만 수행된다. 이 옵션이 생략되면 모든 SHARD DB에 대해 질의를 수행한다.

SHARD 파라미터 값이 OFF이면 이 파라미터의 영향을 받지 않는다.

#### **-v**

아래의 정보와 함께 에러 메시지와 SELECT의 결과셋을 같이 출력한다.

- RESULT: 질의 수행 후 에러가 반환되었는지 여부. [OK | FAIL] 출력
- SHARD\_ID: 질의가 수행된 SHARD DB의 ID(SHARD 파라미터 값이 OFF이면 해당 항목은 출력되지 않음)
- ROW COUNT: DML에 의해 영향을 받는 행(affected rows)의 개수, 또는 SELECT 질의인 경우 행의 개수. 질의 수행 에러 시 -1을 출력
- EXECUTION TIME: 질의가 수행된 시간
- QUERY: 사용자가 입력한 질의

이 옵션이 생략되면 "RESULT, SHARD\_ID, ROW COUNT, EXECUTION TIME, QUERY"만 출력한다.

다음은 위의 옵션들을 사용한 예이다.

- DB에 질의

#### **cubrid\_broker.conf의 SHARD 파라미터 값이 OFF일 때**

DB 접속이 잘 되는지 확인한다.

```
$ cubrid broker test shard1 -D shard -u shard -p shard123 -c
"select 1 from db_root where charset = 3"
```

```
@ cubrid broker test
@ [OK] CONNECT broker1 DB [demodb] USER [shard]
```

```
@ SHARD OFF
```

```
RESULT ROW COUNT EXECUTION TIME QUERY
=====
```



```
OK 1 0.011341 sec select 1,'a' from db_root where charset = 3
@ [OK] QUERY TEST
```

### cubrid\_broker.conf의 SHARD 파라미터 값이 ON일 때

모든 SHARD DB에 대해 접속이 잘 되는지 확인한다.

```
$ cubrid broker test shard1 -D shard -u shard -p shard123 -c
"select 1 from db_root where charset = 3"

@ cubrid broker test
@ [OK] CONNECT shard1 DB [shard] USER [shard]

@ SHARD ON

RESULT SHARD_ID ROW COUNT EXECUTION TIME QUERY
=====
OK 0 1 0.003436 sec select 1 from db_root where charset = 3
OK 1 1 0.003010 sec select 1 from db_root where charset = 3
OK 2 1 0.003039 sec select 1 from db_root where charset = 3
OK 3 1 0.002916 sec select 1 from db_root where charset = 3
@ [OK] QUERY TEST
```

### · 사용자 권한 확인

#### SHARD DB 중 하나에 INSERT 권한이 없는 경우

INSERT 권한이 없으면 RESULT가 FAIL로 표시된다.

```
$ cubrid broker test shard1 -c "insert into foo values (1,'a') " -v

@ cubrid broker test
@ [OK] CONNECT shard1 DB [shard] USER [shard]

@ SHARD ON

RESULT SHARD_ID ROW COUNT EXECUTION TIME QUERY
=====
OK 0 1 0.001322 sec insert into foo values(1,'a')
FAIL(-494) -1 -1 0.001608 sec insert into foo values(1,'a')
<Error>
ERROR CODE : -494
Semantic: INSERT is not authorized on foo. insert into foo foo
(foo.a, foo.b) values (1, cast('a' as v...[CAS INFO -
127.0.0.1:52002, 1, 18145].

OK 2 1 0.001334 sec insert into foo values(1,'a')
```

```
OK 3 1 0.001325 sec insert into foo values(1,'a')
@ [FAIL] QUERY TEST
```

### 브로커에 접속하는 DB 중 하나에 UPDATE 권한이 없는 경우

UPDATE 권한이 없으면 RESULT가 FAIL로 표시된다.

```
$ vi dml.txt
```

```
#query
select a from foo
insert into foo(b) values(3)
update foo set c = 2 where b = 3
delete foo where b = 3
```

```
$ cubrid broker test broker1 -D demodb -u shard -p shard123 -i
dml.txt -v
```

```
@ cubrid broker test
@ [OK] CONNECT broker1 DB [demodb] USER [shard]
```

```
@ SHARD OFF
```

```
RESULT ROW COUNT EXECUTION TIME QUERY
```

```
=====
```

```
OK 1 0.001612 sec select a from foo
```

```
<Result of SELECT Command>
```

```
 a

 1
```

```
OK 1 0.001215 sec insert into foo(b) values(3)
```

```
FAIL(-494) -1 0.001291 sec update foo set c = 2 where b = 3
```

```
<Error>
```

```
ERROR CODE : -494
```

```
Semantic: UPDATE is not authorized on foo. update foo foo set
foo.c=2 where foo.b=3[CAS INFO - 127.0.0.1:52001, 1, 18139].
```

```
OK 0 0.001534 sec delete foo where b = 3
```

```
@ [FAIL] QUERY TEST
```

### • SHARD HASH 정상 동작 확인

특정 키에 대해 해싱이 잘 되는지 확인한다.

```
$ vi test_query.txt
```

```
#query
```

```
select number from demo_db where key = /** shard_key */ 14
select number from demo_db where key = /** shard_key */ 50
select number from demo_db where key = /** shard_key */ 80
select number from demo_db where key = /** shard_key */ 120
..
```

```
$ cubrid broker test shard1 -D shard -u shard -p shard123 -i
shard_key.txt -v -s

@ cubrid broker test
@ [OK] CONNECT shard1 DB [shard] USER [shard]

@ SHARD ON

RESULT SHARD_ID ROW COUNT EXECUTION TIME QUERY
=====
OK 0 1 0.002225 sec select * from foo where a = /** shard_key */ 10
<Result of SELECT Command>
 a b

 10 'aaaa'

OK 1 1 0.001870 sec select * from foo where a = /** shard_key */ 40
<Result of SELECT Command>
 a b

 40 'bbb'

OK 2 1 0.002004 sec select * from foo where a = /** shard_key */ 70
<Result of SELECT Command>
 a b

 70 'cccc'

OK 3 1 0.002025 sec select * from foo where a = /** shard_key */ 100
<Result of SELECT Command>
 a b

 100 'dddd'

@ [OK] QUERY TEST
```

· -v 옵션 사용 여부

#### **-v 옵션을 사용할 때**

SELECT 질의가 성공하면 결과셋을 출력하며, 실패 시 에러 메시지를 출력한다.

```
$ cubrid broker test broker1 -D demodb -u shard -p shard123 -i
dml.txt -v
@ cubrid broker test
@ [OK] CONNECT broker1 DB [demodb] USER [shard]

@ SHARD OFF

RESULT ROW COUNT EXECUTION TIME QUERY
OK 1 0.001311 sec select a from foo
<Result of SELECT Command>
 a

 1

OK 1 0.001083 sec insert into foo(b) values(3)
FAIL(-494) -1 0.001166 sec update foo set c = 2 where b = 3
<Error>
ERROR CODE : -494
Semantic: UPDATE is not authorized on foo. update foo foo set
foo.c=2 where foo.b=3[CAS INFO - 127.0.0.1:52001, 1, 18139].

OK 0 0.001399 sec delete foo where b = 3
@ [FAIL] QUERY TEST
```

#### **-v 옵션을 사용하지 않을 때**

질의 성공, 실패 여부만 출력한다.

```
$ cubrid broker test broker1 -D demodb -u shard -p shard123 -i
dml.txt

@ cubrid broker test
@ [OK] CONNECT broker1 DB [demodb] USER [shard]

@ SHARD OFF

RESULT ROW COUNT EXECUTION TIME QUERY
OK 1 0.001485 sec select a from foo
OK 1 0.001123 sec insert into foo(b) values(3)
FAIL(-494) -1 0.001180 sec update foo set c = 2 where b = 3
OK 0 0.001393 sec delete foo where b = 3
@ [FAIL] QUERY TEST
```

#### **• SHARD 키 값 확인**

-s 옵션과 함께 SHARD 키 힌트가 주어지면 해당 SHARD DB에 질의를 수행하며, 그 결과를 출력한다. SHARD\_ID로 어느 SHARD DB에서 질의가 수행되었는지 확인할 수 있다.

```

$ cubrid broker test shard1 -i shard_key.txt -s -v

@ cubrid broker test
@ [OK] CONNECT shard1 DB [shard1] USER [shard]

@ SHARD ON

RESULT SHARD_ID ROW COUNT EXECUTION TIME QUERY
OK 0 1 0.144730 sec select * from foo where a = /**+ shard_key */ 10
<Result of SELECT Command>
 a b

 10 'aaaa'

OK 1 1 0.001870 sec select * from foo where a = /**+ shard_key */ 40
<Result of SELECT Command>
 a b

 40 'bbb'

OK 2 1 0.002004 sec select * from foo where a = /**+ shard_key */ 70
<Result of SELECT Command>
 a b

 70 'cccc'

OK 3 1 0.002025 sec select * from foo where a = /**+ shard_key */ 100
<Result of SELECT Command>
 a b

 100 'dddd'

@ [OK] QUERY TEST

```

## 브로커 로그

브로커 구동과 관련된 로그에는 접속 로그, 에러 로그, SQL 로그가 있다. 각각의 로그는 설치 디렉터리의 log 디렉터리에서 확인할 수 있으며, 저장 디렉터리의 변경은 브로커 환경 설정 파일(**cubrid\_broker.conf**)의 **LOG\_DIR** 파라미터와 **ERROR\_LOG\_DIR** 파라미터를 통해 설정할 수 있다.

SHARD = ON 인 경우, CUBRID proxy의 로그 디렉터리는 **SHARD\_PROXY\_LOG\_DIR** 파라미터를 통해 설정할 수 있다.

## 접속 로그 확인

접속 로그 파일은 응용 클라이언트 접속에 관한 정보를 기록하며, **\$CUBRID/log/broker/<broker\_name>.access**에 저장된다. 또한, 브로커 환경 설정 파일에서 **ACCESS\_LOG** 파라미터가 ON으로 설정된 경우, 브로커의 구동이 정상적으로 종료되면 접속 로그 파일이 저장된다.

ACCESS\_LOG\_MAX\_SIZE 파라미터를 통해 ACCESS\_LOG 파일의 최대 크기 지정이 가능하다. ACCESS\_LOG 파일이 지정한 크기보다 커지면 broker\_name.access.YYYYMMDDHHMISS 형식의 이름으로 백업된 후 새 파일(broker\_name.access)에 로그가 기록된다.

접속 거부된 기록은 broker\_name.access.denied 에 기록된다. ACCESS\_LOG 파일과 동일한 규칙으로 백업된다.

다음은 log 디렉터리에 생성된 접속 로그 파일의 예제와 설명이다.

```
1 192.168.56.4 2020/11/10 14:41:55 testdb dba NEW 6
```

- 1: 브로커의 응용서버에 부여된 ID
- 192.168.56.4: 응용 클라이언트의 IP 주소
- 2020/11/10 14:41:55: 클라이언트 요청을 처리 시작한 시각
- testdb : 클라이언트가 접속 요청한 데이터베이스 이름
- dba : 클라이언트가 접속 요청한 데이터베이스 계정이름
- NEW : connection\_type
  - NEW : 새로운 접속
  - OLD : change client 또는 CAS 재시작으로 인한 기존 연결의 재접속
  - REJ : 접속 거부(access.denied 파일에만 기록됨)
- 6 : session-id(서버에서 할당한 세션 번호)

## 에러 로그 확인

에러 로그 파일은 응용 클라이언트의 요청을 처리하는 도중에 발생한 에러에 관한 정보를 기록하며, **\$CUBRID/log/broker/error\_log<broker\_name>\_<app\_server\_num>.err**에 저장된다. 에러 코드 및 에러 메시지는 **CAS 에러**를 참고한다.

다음은 에러 로그의 예제와 설명이다.

```
Time: 02/04/09 13:45:17.687 - SYNTAX ERROR *** ERROR CODE = -493,
Tran = 1, EID = 38
Syntax: Unknown class "unknown_tbl". select * from unknown_tbl
```

- Time: 02/04/09 13:45:17.687: 에러 발생 시각
- ◦ SYNTAX ERROR: 에러의 종류(SYNTAX ERROR, ERROR 등)
- \*\*\* ERROR CODE = -493: 에러 코드
- Tran = 1: 트랜잭션 ID. -1은 트랜잭션 ID를 할당 받지 못한 경우임.
- EID = 38: 에러 ID. SQL 문 처리 중 에러가 발생한 경우, 서버나 클라이언트 에러 로그와 관련이 있는 SQL 로그를 찾을 때 사용함.
- Syntax ...: 에러 메시지

## SQL 로그 관리

SQL 로그 파일은 응용 클라이언트가 요청하는 SQL을 기록하며, `<broker_name>_<app_server_num>.sql.log`라는 이름으로 저장된다. SQL 로그는 **SQL\_LOG** 파라미터 값이 ON인 경우에 설치 디렉터리의 `log/broker/sql_log` 디렉터리에 생성된다. 이 때, 생성되는 SQL 로그 파일의 크기는 **SQL\_LOG\_MAX\_SIZE** 파라미터의 설정값을 초과할 수 없으므로 주의한다. CUBRID는 SQL 로그를 관리하기 위한 유틸리티로서 **broker\_log\_top**, **cubrid\_replay**를 제공하며, 이 유틸리티는 SQL 로그가 존재하는 디렉터리에서 실행해야 한다.

다음은 SQL 로그 파일의 예제와 설명이다.

```
13-06-11 15:07:39.282 (0) STATE idle
13-06-11 15:07:44.832 (0) CLIENT IP 192.168.10.100
13-06-11 15:07:44.835 (0) CLIENT VERSION 11.4.0.1089
13-06-11 15:07:44.835 (0) session id for connection 0
13-06-11 15:07:44.836 (0) connect db demodb user dba url jdbc:cubrid:
192.168.10.200:30000:demodb:dba:*****: session id 12
13-06-11 15:07:44.836 (0) DEFAULT isolation_level 4, lock_timeout -1
13-06-11 15:07:44.840 (0) end_tran COMMIT
13-06-11 15:07:44.841 (0) end_tran 0 time 0.000
13-06-11 15:07:44.841 (0) *** elapsed time 0.004

13-06-11 15:07:44.844 (0) check_cas 0
13-06-11 15:07:44.848 (0) set_db_parameter lock_timeout 1000
13-06-11 15:09:36.299 (0) check_cas 0
13-06-11 15:09:36.303 (0) get_db_parameter isolation_level 4
13-06-11 15:09:36.375 (1) prepare 0 CREATE TABLE unique_tbl (a INT
PRIMARY key);
13-06-11 15:09:36.376 (1) prepare srv_h_id 1
13-06-11 15:09:36.419 (1) set query timeout to 0 (no limit)
```

```

13-06-11 15:09:36.419 (1) execute srv_h_id 1 CREATE TABLE unique_tbl
(a INT PRIMARY key);
13-06-11 15:09:38.247 (1) execute 0 tuple 0 time 1.827
13-06-11 15:09:38.247 (0) auto_commit
13-06-11 15:09:38.344 (0) auto_commit 0
13-06-11 15:09:38.344 (0) *** elapsed time 1.968

13-06-11 15:09:54.481 (0) get_db_parameter isolation_level 4
13-06-11 15:09:54.484 (0) close_req_handle srv_h_id 1
13-06-11 15:09:54.484 (2) prepare 0 INSERT INTO unique_tbl VALUES
(1);
13-06-11 15:09:54.485 (2) prepare srv_h_id 1
13-06-11 15:09:54.488 (2) set query timeout to 0 (no limit)
13-06-11 15:09:54.488 (2) execute srv_h_id 1 INSERT INTO unique_tbl
VALUES (1);
13-06-11 15:09:54.488 (2) execute 0 tuple 1 time 0.001
13-06-11 15:09:54.488 (0) auto_commit
13-06-11 15:09:54.505 (0) auto_commit 0
13-06-11 15:09:54.505 (0) *** elapsed time 0.021

...

13-06-11 15:19:04.593 (0) get_db_parameter isolation_level 4
13-06-11 15:19:04.597 (0) close_req_handle srv_h_id 2
13-06-11 15:19:04.597 (7) prepare 0 SELECT * FROM unique_tbl WHERE
ROWNUM BETWEEN 1 AND 5000;
13-06-11 15:19:04.598 (7) prepare srv_h_id 2 (PC)
13-06-11 15:19:04.602 (7) set query timeout to 0 (no limit)
13-06-11 15:19:04.602 (7) execute srv_h_id 2 SELECT * FROM
unique_tbl WHERE ROWNUM BETWEEN 1 AND 5000;
13-06-11 15:19:04.602 (7) execute 0 tuple 1 time 0.001
13-06-11 15:19:04.607 (0) end_tran COMMIT
13-06-11 15:19:04.607 (0) end_tran 0 time 0.000
13-06-11 15:19:04.607 (0) *** elapsed time 0.009

```

- 13-06-11 15:07:39.282: 응용 클라이언트의 요청 시각
- (1): SQL 문 그룹의 시퀀스 번호이며, prepared statement pooling을 사용하는 경우, 파일 내에서 SQL 문마다 고유(unique)하게 부여된다.
- CLIENT IP: 응용 클라이언트의 IP
- CLIENT VERSION: 응용 클라이언트 드라이버의 버전
- prepare 0: prepared statement인지 여부
- prepare srv\_h\_id 1: 해당 SQL 문을 srv\_h\_id 1로 prepare한다.
- (PC): 플랜 캐시에 저장되어 있는 내용을 사용하는 경우에 출력된다.
- Execute 0 tuple 1 time 0.001: 1개의 row가 실행되고, 소요 시간은 0.001초



- auto\_commit/auto\_rollback: 자동으로 커밋되거나, 롤백되는 것을 의미한다. 두 번째 auto\_commit/auto\_rollback은 에러 코드이며, 0은 에러가 없이 트랜잭션이 완료되었음을 뜻한다.

## broker\_log\_top

**broker\_log\_top** 유틸리티는 특정 기간 동안 생성된 SQL 로그를 분석하여 실행 시간이 긴 순서대로 각 SQL 문과 실행 시간에 관한 정보를 파일에 출력하며, 분석된 결과는 각각 log.top.q 및 log.top.res에 저장된다.

**broker\_log\_top** 유틸리티는 실행 시간이 긴 질의를 분석할 때 유용하며, 구문은 다음과 같다.

```
broker_log_top [options] sql_log_file_list
```

- sql\_log\_file\_list: 분석할 로그 파일 이름

**broker\_log\_top** 에서 사용하는 [options]는 다음과 같다.

**-t**

트랜잭션 단위로 결과를 출력한다.

**-F DATETIME**

분석 대상 SQL의 시작 날짜 및 시간을 지정한다. 입력 형식은 YY-MM-DD[ hh[:mm[:ss[:msec]]]]이며 []로 감싼 부분은 생략할 수 있다. 생략하면 hh, mm, ss, msec은 0을 입력한 것과 같다.

**-T DATETIME**

분석 대상 SQL의 끝 날짜 및 시간을 지정한다. 입력 형식은 **-F** 옵션의 *DATETIME*과 같다.

옵션을 모두 생략하면, 명시한 로그의 모든 SQL에 대해 SQL 문 단위로 결과를 출력한다.

다음은 밀리초까지 검색 범위를 설정하는 예제이다.

```
broker_log_top -F "13-01-19 15:00:25.000" -T "13-01-19 15:15:25.180"
log1.log
```

다음 예에서 시간 형식이 생략된 부분은 기본값 0으로 정해진다. 즉, -F "13-01-19 00:00:00.000" -T "13-01-20 00:00:00.000"을 입력한 것과 같다.

```
broker_log_top -F "13-01-19" -T "13-01-20" log1.log
```

다음 예는 2013년 11월 11일부터 11월 12일까지 생성된 SQL 로그에 대해 실행 시간이 긴 SQL문을 확인하기 위하여 **broker\_log\_top** 유틸리티를 실행한 화면이다. 기간을 지정할 때, 연, 월, 일은 하이픈(-)으로 구분한다. Windows에서는 "\*.sql.log"를 인식하지 않으므로 SQL 로그 파일들을 공백(space)으로 구분해서 나열해야 한다.

```
--Linux에서 broker_log_top 실행
% broker_log_top -F "13-11-11" -T "13-11-12" -t *.sql.log

query_editor_1.sql.log
query_editor_2.sql.log
query_editor_3.sql.log
query_editor_4.sql.log
query_editor_5.sql.log

--Windows에서 broker_log_top 실행
% broker_log_top -F "13-11-11" -T "13-11-12" -t
query_editor_1.sql.log query_editor_2.sql.log query_editor_3.sql.log
query_editor_4.sql.log query_editor_5.sql.log
```

위 예제를 실행하면 SQL 로그 분석 결과가 저장되는 **log.top.q** 및 **log.top.res** 파일이 동일한 디렉터리에 생성된다. **log.top.q** 에서 각 SQL 문 및 SQL 로그 상의 라인 번호를 확인할 수 있고, **log.top.res** 에서 각 SQL 문에 대한 최소 실행 시간, 최대 실행 시간, 평균 실행 시간, 쿼리 실행 수를 확인할 수 있다.

```
--log.top.q 파일의 내용
[Q1]-----
broker1_6.sql.log:137734
13-11-11 18:17:59.396 (27754) execute_all srv_h_id 34 select
a.int_col, b.var_col from dml_v_view_6 a, dml_v_view_6 b,
dml_v_view_6 c , dml_v_view_6 d, dml_v_view_6 e where
a.int_col=b.int_col and b.int_col=c.int_col and c.int_col=d.int_col
and d.int_col=e.int_col order by 1,2;
11/11 18:18:58.378 (27754) execute_all 0 tuple 497664 time 58.982
.
.
[Q4]-----
broker1_100.sql.log:142068
13-11-11 18:12:38.387 (27268) execute_all srv_h_id 798 drop table
list_test;
13-11-11 18:13:08.856 (27268) execute_all 0 tuple 0 time 30.469
```

```
--log.top.res 파일의 내용
```

|      | max    | min    | avg    | cnt(err) |
|------|--------|--------|--------|----------|
| [Q1] | 58.982 | 30.371 | 44.676 | 2 (0)    |
| [Q2] | 49.556 | 24.023 | 32.688 | 6 (0)    |
| [Q3] | 35.548 | 25.650 | 30.599 | 2 (0)    |
| [Q4] | 30.469 | 0.001  | 0.103  | 1050 (0) |

## cubrid\_replay

**cubrid\_replay** 유틸리티는 브로커의 SQL 로그를 재생하여, 기존의 수행 시간과 재생할 때의 수행 시간 차이를 비교하여 차이가 큰 것부터(기존보다 느린 것부터) 순서대로 정렬한 결과를 출력한다.

이 유틸리티는 SQL 로그에 기록된 질의들을 재생하되, 데이터의 변경이 발생하는 질의는 실행하지 않는다. 별도의 옵션을 주지 않으면 SELECT 문만 수행되며, -r 옵션을 부여하면 UPDATE, DELETE 문을 SELECT 문으로 변환하여 수행한다.

이 유틸리티는 서로 다른 두 장비 간 성능을 비교할 때 사용할 수 있는데, 예를 들어 하드웨어 스펙이 동일한 마스터와 슬레이브 사이에서도 같은 질의에 대해 성능 차이가 존재할 수 있다.

```
cubrid_replay -I <broker_host> -P <broker_port> -d <db_name>
[options] <sql_log_file> <output_file>
```

- *broker\_host*: CUBRID 브로커의 IP 주소 또는 호스트 이름
- *broker\_port*: CUBRID 브로커의 포트 번호
- *db\_name*: 질의를 실행할 데이터베이스
- *sql\_log\_file*: CUBRID 브로커의 SQL 로그 파일(\$CUBRID/log/broker/sql\_log/\*.log, \*.log.bak)
- *output\_file*: 수행 결과를 저장할 파일 이름

**cubrid\_replay** 에서 사용하는 [options]는 다음과 같다.

### -u DB\_USER

데이터베이스 사용자 이름 지정(기본값: public)

### -p DB\_PASSWORD

데이터베이스 암호 지정

### -r

UPDATE, DELETE 질의를 SELECT 질의로 변환

### -h SECOND

질의 실행 간격을 지정(기본값: 0.01초)

### -D SECOND

(재생된 질의 수행 시간 - 기존에 실행된 질의 수행 시간)이 이 설정 값보다 큰 경우만 해당 질의가 *output\_file*에 출력됨(기본값: 0.01초)

### -F DATETIME

재생 대상 SQL의 시작 날짜 및 시간을 지정한다. 입력 형식은 YY[-MM[-DD[ hh[:mm[:ss[:msec]]]]]]이며 []로 감싼 부분은 생략할 수 있다. 생략하면 MM, DD는 01을 입력한 것과 같고, hh, mm, ss, msec은 0을 입력한 것과 같다.

**-T DATETIME**

재생 대상 SQL의 끝 날짜 및 시간을 지정한다. 입력 형식은 **-F** 옵션의 *DATE* 와 같다.

```
$ cubrid_replay -I testhost -P 33000 -d testdb -u dba -r
testdb_1_11_1.sql.log.bak output.txt
```

위의 명령을 실행하면 실행 결과의 요약 정보가 화면에 출력된다.

```
----- Result Summary -----
* Total queries : 153103
* Skipped queries (see skip.sql) : 5127
* Error queries (see replay.err) : 30
* Slow queries (time diff > 0.000 secs) : 89987
* Max execution time diff : 0.016
* Avg execution time diff : -0.001

cubrid_replay run time : 245.308417 sec
```

- Total queries: 날짜 및 시간이 지정된 범위 안의 전체 질의 개수. DDL, DML을 포함
- Skipped queries: **-r** 옵션이 사용되었을 때 UPDATE/DELETE 문을 SELECT 문으로 변환할 수 없는 질의 개수. 이 질의는 skip.sql 파일에 기록됨
- Slow queries: **-D** 옵션의 설정 값보다 수행 시간의 차이가 더 큰(재생된 실행 시간이 기존 실행 시간에 설정한 값을 더한 것보다 느린) 질의 개수. **-D** 옵션을 생략하면 0.01초를 기본으로 설정함.
- Max execution time diff: 수행 시간의 차이 중 가장 큰 값(단위: 초)
- Avg execution time diff: 수행 시간의 차이의 평균 값(단위: 초)
- cubrid\_replay run time: 유틸리티의 수행 시간

“Skipped queries”는 내부 요인에 의해 UPDATE/DELETE 문에서 SELECT 문으로 질의 변환이 불가능한 경우로, skip.sql 파일에 기록된 질의문의 성능에 대해서는 별도로 확인해볼 필요가 있다.

또한, 변환된 질의문의 수행 시간은 데이터 변경 시간이 빠진 것임을 감안해야 한다.

output.txt 파일에는 SQL 로그의 수행 시간보다 재생된 SQL 수행 시간이 더 느린 SQL부터 정렬되어 기록된다. 즉, {(재생된 SQL 수행 시간) - {(SQL 로그의 수행 시간) + (-D 옵션 설정 시간)}}이 내림차순으로 정렬되어 기록된다. “-r” 옵션이 사용되었으므로 UPDATE/DELETE 문은 SELECT 문으로 재작성되어 실행된다.

```
EXEC TIME (REPLAY / SQL_LOG / DIFF): 0.003 / 0.001 / 0.002
SQL: UPDATE NDV_QUOTA_INFO SET last_mod_date = now() , used_quota =
(SELECT IFNULL(sum(file_size),0) FROM NDV_RECYCLED_FILE_INFO WHERE
user_id = ?) + (SELECT IFNULL(sum(file_size),0) FROM NDV_FILE_INFO
WHERE user_id = ?) WHERE user_id = ? /+shard_val(6900403)/ /* SQL :
```

```

NDVMUpdResetUsedQuota */
REWRITE SQL: select NDV_QUOTA_INFO, class NDV_QUOTA_INFO,
cast(SYS_DATETIME as datetime), cast((select
ifnull(sum(NDV_RECYCLED_FILE_INFO.file_size), 0) from
NDV_RECYCLED_FILE_INFO NDV_RECYCLED_FILE_INFO where
(NDV_RECYCLED_FILE_INFO.user_id= ?:0))+(select
ifnull(sum(NDV_FILE_INFO.file_size), 0) from NDV_FILE_INFO
NDV_FILE_INFO where (NDV_FILE_INFO.user_id= ?:1)) as bigint) from
NDV_QUOTA_INFO NDV_QUOTA_INFO where (NDV_QUOTA_INFO.user_id= ?:2)
BIND 1: 'babaemo'
BIND 2: 'babaemo'
BIND 3: 'babaemo'

```

- EXEC TIME: (재생 시간 / SQL 로그에서의 수행 시간 / 두 수행 시간의 차이)
- SQL: 브로커의 SQL 로그에 존재하는 원본 SQL
- REWRITE SQL: **-r** 옵션이 지정되어 UPDATE 또는 DELETE 문에서 변환된 SELECT 문

### 참고

broker\_log\_runner는 9.3 버전부터 제거될 예정(deprecated)이므로, cubrid\_replay를 대신 사용하도록 한다.

## CAS 에러

CAS 에러는 브로커 응용 서버(CAS) 프로세스에서 발생하는 에러로, 드라이버를 이용하여 CAS에 접속하는 모든 응용 프로그램에서 발생할 수 있다.

다음은 CAS에서 발생하는 에러 코드를 정리한 표이다. 같은 에러 번호에 대해 CCI와 JDBC의 에러 메시지가 서로 다르게 나타날 수 있다. 에러 메시지가 하나만 있으면 같은 것이며, 두 개가 있는 경우 앞에 있는 것이 CCI 에러 메시지, 뒤에 있는 것이 JDBC 에러 메시지이다.

| 에러 코드명(에러 번호)                 | 에러 메시지 (CCI / JDBC)     | 비고 |
|-------------------------------|-------------------------|----|
| CAS_ER_INTERNAL(-10001)       |                         |    |
| CAS_ER_NO_MORE_MEMORY(-10002) | Memory allocation error |    |

| 에러 코드명(에러 번호)                  | 에러 메시지 (CCI / JDBC)                                                | 비고                              |
|--------------------------------|--------------------------------------------------------------------|---------------------------------|
| CAS_ER_COMMUNICATION(-10003)   | Cannot receive data from client / Communication error              |                                 |
| CAS_ER_ARGS(-10004)            | Invalid argument                                                   |                                 |
| CAS_ER_TRAN_TYPE(-10005)       | Invalid transaction type argument / Unknown transaction type       |                                 |
| CAS_ER_SRV_HANDLE(-10006)      | Server handle not found / Internal server error                    |                                 |
| CAS_ER_NUM_BIND(-10007)        | Invalid parameter binding value argument / Parameter binding error | 바인딩할 데이터 수가 전송할 데이터 수와 일치하지 않음. |
| CAS_ER_UNKNOWN_U_TYPE(-10008)  | Invalid T_CCI_U_TYPE value / Parameter binding error               |                                 |
| CAS_ER_DB_VALUE(-10009)        | Cannot make DB_VALUE                                               |                                 |
| CAS_ER_TYPE_CONVERSION(-10010) | Type conversion error                                              |                                 |
| CAS_ER_PARAM_NAME(-10011)      | Invalid T_CCI_DB_PARAM value / Invalid database parameter name     | 시스템 파라미터 이름이 유효하지 않음            |
| CAS_ER_NO_MORE_DATA(-10012)    | Invalid cursor position / No more data                             |                                 |

| 에러 코드명(에러 번호)                        | 에러 메시지 (CCI / JDBC)                                                                 | 비고                                 |
|--------------------------------------|-------------------------------------------------------------------------------------|------------------------------------|
| CAS_ER_OBJECT(-10013)                | Invalid oid / Object is not valid                                                   |                                    |
| CAS_ER_OPEN_FILE(-10014)             | Cannot open file / File open error                                                  |                                    |
| CAS_ER_SCHEMA_TYPE(-10015)           | Invalid T_CCI_SCH_TYPE value / Invalid schema type                                  |                                    |
| CAS_ER_VERSION(-10016)               | Version mismatch                                                                    | DB 서버 버전과 클라이언트 (CAS) 버전이 호환되지 않음. |
| CAS_ER_FREE_SERVER(-10017)           | Cannot process the request. Try again later                                         | 응용 프로그램의 연결요청을 처리할 CAS를 할당할 수 없음.  |
| CAS_ER_NOT_AUTHORIZED_CLIENT(-10018) | Authorization error                                                                 | 접근을 불허함.                           |
| CAS_ER_QUERY_CANCEL(-10019)          | Cannot cancel the query                                                             |                                    |
| CAS_ER_NOT_COLLECTION(-10020)        | The attribute domain must be the set type                                           |                                    |
| CAS_ER_COLLECTION_DOMAIN(-10021)     | Heterogeneous set is not supported / The domain of a set must be the same data type |                                    |
| CAS_ER_NO_MORE_RESULT_SET(-10022)    | No More Result                                                                      |                                    |

| 에러 코드명(에러 번호)                                    | 에러 메시지 (CCI / JDBC)                                           | 비고 |
|--------------------------------------------------|---------------------------------------------------------------|----|
| CAS_ER_INVALID_CALL_STMT(-10023)                 | Illegal CALL statement                                        |    |
| CAS_ER_STMT_POOLING(-10024)                      | Invalid plan                                                  |    |
| CAS_ER_DBSERVER_DISCONNECTED(-10025)             | Cannot communicate with DB Server                             |    |
| CAS_ER_MAX_PREPARED_STMT_COUNT_EXCEEDED(-10026)  | Cannot prepare more than MAX_PREPARED_STMT_COUNT statements   |    |
| CAS_ER_HOLDABLE_NOT_ALLOWED(-10027)              | Holdable results may not be updatable or sensitive            |    |
| CAS_ER_HOLDABLE_NOT_ALLOWED_KEEP_CON_OFF(-10028) | Holdable results are not allowed while KEEP_CONNECTION is off |    |
| CAS_ER_NOT_IMPLEMENTED(-10100)                   | None / Attempt to use a not supported service                 |    |
| CAS_ER_SSL_TYPE_NOT_ALLOWED(-10103)              | None / The requested SSL mode is not permitted                |    |
| CAS_ER_IS(-10200)                                | None / Authentication failure                                 |    |



## 게이트웨이

### 게이트웨이 구동

게이트웨이를 구동하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway start
@ cubrid gateway start
++ cubrid gateway start: success
```

이미 게이트웨이가 구동 중이라면 다음과 같은 메시지가 출력된다.

```
cubrid gateway start
@ cubrid gateway start
++ cubrid gateway is running.
```

### 게이트웨이 종료

게이트웨이를 종료하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway stop
@ cubrid gateway stop
++ cubrid gateway stop: success
```

이미 게이트웨이가 종료되었다면 다음과 같은 메시지가 출력된다.

```
$ cubrid gateway stop
@ cubrid gateway stop
++ cubrid gateway is not running.
```

### 게이트웨이 재시작

전체 게이트웨이를 재시작하기 위하여 다음과 같이 입력한다.

```
$ cubrid gateway restart
```

### 게이트웨이 상태 확인

**cubrid gateway status** 는 여러 옵션을 제공하여, 각 게이트웨이의 처리 완료된 작업 수, 처리 대기중인 작업 수를 포함한 게이트웨이 상태 정보를 확인할 수 있도록 한다.

## cubrid gateway status [options] [expr]

· *expr*: 게이트웨이 이름의 일부 또는 "SERVICE=ON|OFF"

*expr* 이 명시되면 이름이 *expr* 을 포함하는 게이트웨이에 대한 상태 모니터링을 수행하고, 생략되면 CUBRID 게이트웨이 환경 설정 파일( **cubrid\_gateway.conf** )에 등록된 전체 게이트웨이에 대해 상태 모니터링을 수행한다.

*expr* 에 "SERVICE=ON"이 명시되면 구동 중인 게이트웨이의 상태만 출력하며, "SERVICE=OFF"가 명시되면 멈춰있는 게이트웨이의 이름만 출력한다.

cubrid gateway status에서 사용하는 [options]는 다음과 같다. 이들 중 -b, -q, -c, -m, -S, -P, -f는 출력할 정보를 정의하는 모니터링 옵션이고, -s, -l, -t는 출력을 제어하는 옵션이다. 이 모든 옵션들은 상호 조합하여 사용할 수 있다.

### -b

게이트웨이 응용 서버(CAS)에 관한 정보는 포함하지 않고, 게이트웨이에 관한 상태 정보만 출력한다.

### -q

작업 큐에 대기 중인 작업을 출력한다.

### -f

게이트웨이가 접속한 DB 및 호스트 정보를 출력한다.

**-b** 옵션과 함께 쓰이는 경우, CAS 정보를 추가로 출력한다. 하지만 -b 옵션에서 나타나는 SELECT, INSERT, UPDATE, DELETE, OTHERS 항목은 제외된다.

**-P** 옵션과 함께 쓰이는 경우, STMT-POOL-RATIO 항목을 추가로 출력한다. 이 항목은 prepare statement 사용 시 pool에서 statement를 사용하는 비율을 나타낸다.

### -l SECOND

**-l** 옵션은 **-f** 옵션과만 함께 쓰이며, 클라이언트 Waiting/Busy 상태인 CAS의 개수를 출력할 때 누적 주기(단위: 초)를 지정하기 위해 사용한다. **-l SECOND** 옵션을 생략하면 기본값은 1초이다.

### -t

화면 출력시 tty mode 로 출력한다. 출력 내용을 리다이렉션하여 파일로 쓸 수 있다.

### -s SECOND

게이트웨이에 관한 상태 정보를 지정된 시간마다 주기적으로 출력한다. q를 입력하면 명령 프롬프트로 복귀한다.

옵션 및 인수를 입력하지 않으면 전체 게이트웨이 상태 정보를 출력한다.

```
$ cubrid gateway status
@ cubrid gateway status
```

```
% oracle_gateway

ID PID QPS LQS PSIZE STATUS

1 28434 0 0 50144 IDLE
2 28435 0 0 50144 IDLE
3 28436 0 0 50144 IDLE
4 28437 0 0 50140 IDLE
5 28438 0 0 50144 IDLE
```

- % oracle\_gateway: 브로커의 이름
- ID: 게이트웨이 내에서 순차적으로 부여한 CAS의 일련 번호
- PID: 게이트웨이 내 CAS 프로세스의 ID
- QPS: 초당 처리된 질의의 수
- LQS: 초당 처리되는 장기 실행 질의의 수
- PSIZE: CAS 프로세스 크기
- STATUS: CAS의 현재 상태로서, BUSY/IDLE/CLIENT\_WAIT/CLOSE\_WAIT가 있다.

다음은 **-b** 옵션을 사용하여 브로커에 관해 5초 간격으로 상세한 상태 정보를 출력한다. 화면이 5초 간격마다 새로운 상태 정보로 갱신되며, 상태 정보 화면을 벗어나려면 <Q>를 누른다.

```
$ cubrid gateway status -b -s 5
@ cubrid gateway status

NAME PID PORT AS JQ TPS QPS
SELECT INSERT UPDATE DELETE OTHERS LONG-T LONG-Q
ERR-Q UNIQUE-ERR-Q #CONNECT #REJECT
=====
=====
=====
* oracle_gateway 13200 30000 5 0 0 0
0 0 0 0 0 0/60.0 0/60.0
0 0 0 0 0
```

- NAME: 게이트웨이 이름
- PID: 게이트웨이의 프로세스 ID
- PORT: 게이트웨이의 포트 번호
- AS: CAS 개수
- JQ: 작업 큐에서 대기 중인 작업 개수

- TPS: 초당 처리된 트랜잭션의 수(옵션이 “-b -s <sec>”일 때만 해당 구간 계산)
- QPS: 초당 처리된 질의의 수(옵션이 “-b -s <sec>”일 때만 해당 구간 계산)
- SELECT: 게이트웨이 시작 이후 SELECT 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 SELECT 개수로 매번 갱신됨.
- INSERT: 게이트웨이 시작 이후 INSERT 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 INSERT 개수로 매번 갱신됨.
- UPDATE: 게이트웨이 시작 이후 UPDATE 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 UPDATE 개수로 매번 갱신됨.
- DELETE: 게이트웨이 시작 이후 DELETE 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 DELETE 개수로 매번 갱신됨.
- OTHERS: 게이트웨이 시작 이후 SELECT, INSERT, UPDATE, DELETE를 제외한 CREATE, DROP 등의 질의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 질의 개수로 매번 갱신됨.
- LONG-T: LONG\_TRANSACTION\_TIME 시간을 초과한 트랜잭션 개수 / LONG\_TRANSACTION\_TIME 파라미터의 값. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 트랜잭션 개수로 매번 갱신됨.
- LONG-Q: LONG\_QUERY\_TIME 시간을 초과한 질의의 개수 / LONG\_QUERY\_TIME 파라미터의 값. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 질의 개수로 매번 갱신됨.
- ERR-Q: 에러가 발생한 질의의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 에러 개수로 매번 갱신됨.
- UNIQUE-ERR-Q: 고유 키 에러가 발생한 질의의 개수. 옵션이 “-b -s <sec>”인 경우 -s 옵션으로 지정한 초 동안의 고유 키 에러 개수로 매번 갱신됨.
- #CONNECT: 게이트웨이 시작 후 응용 클라이언트가 CAS에 접속한 회수
- #REJECT: 게이트웨이 시작 후 ACL에 포함되지 않은 IP로부터 접속하는 응용 클라이언트가 CAS에 접속하는 것을 거부당한 회수. ACL 설정과 관련하여 **브로커 서버 접속 제한**을 참고한다.

다음은 -q 옵션을 이용하여, oracle\_gateway을 포함하는 이름을 가진 게이트웨이의 상태 정보를 확인하고 해당 게이트웨이의 작업 큐에 대기 중인 작업 상태를 확인한다. 인자로 oracle\_gateway을 입력하지 않으면 모든 게이트웨이에 대하여 작업 큐에 대기 중인 작업 리스트가 출력된다.

```
% cubrid gateway status -q oracle_gateway
@ cubrid gateway status
% oracle_gateway

ID PID QPS LQS PSIZE STATUS

1 28444 0 0 50144 IDLE
```

```

2 28445 0 0 50140 IDLE
3 28446 0 0 50144 IDLE
4 28447 0 0 50144 IDLE
5 28448 0 0 50144 IDLE

```

다음은 **-s** 옵션을 이용하여 oracle\_gateway을 포함하는 이름을 가진 게이트웨이의 상태를 주기적으로 모니터링한다. 인자로 oracle\_gateway을 입력하지 않으면 모든 게이트웨이에 대하여 상태 모니터링이 주기적으로 수행된다. 또한, q를 입력하면 모니터링 화면에서 명령 프롬프트로 복귀한다.

```

% cubrid gateway status -s 5 oracle_gateway
% oracle_gateway

```

```

ID PID QPS LQS PSIZE STATUS

1 28444 0 0 50144 IDLE
2 28445 0 0 50140 IDLE
3 28446 0 0 50144 IDLE
4 28447 0 0 50144 IDLE
5 28448 0 0 50144 IDLE

```

**-t** 옵션을 이용하여 TPS와 QPS 정보를 파일로 출력한다. 파일로 출력하는 것을 중단하려면 <Ctrl+C>를 눌러서 프로그램을 정지시킨다.

```

% cubrid gateway status -b -t -s 1 > log_file

```

다음은 **-f** 옵션을 이용하여 게이트웨이가 연결한 서버/데이터베이스 정보와 응용 클라이언트의 최근 접속 시각, CAS에 접속하는 클라이언트의 IP 주소와 드라이버의 버전 등을 출력한다.

```

$ cubrid gateway status -f oracle_gateway
@ cubrid gateway status
% oracle_gateway

ID PID QPS LQS PSIZE STATUS LAST ACCESS TIME
DB HOST LAST CONNECT TIME CLIENT IP CLIENT
VERSION SQL_LOG_MODE TRANSACTION STIME #CONNECT #RESTART

1 26946 0 0 51168 IDLE 2011/11/16 16:23:42 demodb
localhost 2011/11/16 16:23:40 10.0.1.101
9.2.0.0062 NONE 2011/11/16 16:23:42 0 0
2 26947 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0

```

```

3 26948 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0
4 26949 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0
5 26950 0 0 51172 IDLE 2011/11/16 16:23:34
- - -
0.0.0.0 - -
0 0

```

각 칼럼에 대한 설명은 다음과 같다.

- LAST ACCESS TIME: CAS가 구동한 시각 또는 응용 클라이언트의 CAS에 최근 접속한 시각
- DB: CAS의 최근 접속 데이터베이스 이름
- HOST: CAS의 최근 접속 호스트 이름
- LAST CONNECT TIME: CAS의 DB 서버 최근 접속 시각
- CLIENT IP: 현재 CAS에 접속 중인 응용 클라이언트의 IP 주소. 현재 접속 중인 응용 클라이언트가 없으면 0.0.0.0으로 출력
- CLIENT VERSION: 현재 CAS에 접속 중인 응용 클라이언트의 드라이버 버전
- SQL\_LOG\_MODE: CAS의 SQL 로그 기록 모드. 게이트웨이에 설정된 모드와 동일한 경우 “-”으로 출력
- TRANSACTION STIME: 트랜잭션 시작 시간
- #CONNECT: 게이트웨이 시작 후 응용 클라이언트가 CAS에 접속한 회수
- #RESTART: 게이트웨이 시작 후 CAS의 재구동 회수

**-b** 옵션에 **-f** 옵션을 추가하여 AS(T W B Ns-W Ns-B), CANCELED 정보를 추가로 출력한다.

```

// 게이트웨이 상태 정보 실행 시 -f 옵션 추가. -l 옵션으로 N초 동안의 Ns-W,
Ns-B를 출력하도록 초를 설정
% cubrid gateway status -b -f -l 2
@ cubrid gateway status
NAME PID PSIZE PORT AS(T W B 2s-W 2s-B) JQ TPS QPS LONG-
T LONG-Q ERR-Q UNIQUE-ERR-Q CANCELED ACCESS_MODE SQL_LOG #CONNECT
#REJECT
=====
=====
=====
oracle_gateway 16784 56700 30000 5 0 0 0 0 0 16 29 0/

```

```
60.0 0/60.0 1 1 0 RW ALL
4 1
```

추가된 칼럼에 대한 설명은 다음과 같다.

- AS(T): 실행 중인 CAS의 전체 개수
- AS(W): 현재 클라이언트 대기(Waiting) 상태인 CAS의 개수
- AS(B): 현재 클라이언트 수행(Busy) 상태인 CAS의 개수
- AS(Ns-W): N초 동안 클라이언트 대기(Waiting) 상태였던 CAS의 개수
- AS(Ns-B): N초 동안 클라이언트 수행(Busy) 상태였던 CAS의 개수
- CANCELED: 게이트웨이가 시작된 이후 사용자 인터럽트로 인해 취소된 질의의 개수 (-l N 옵션과 함께 사용하면 N초 동안 누적된 개수).

#### 참고

게이트웨이는 브로커와 거의 유사 하므로, 게이트웨이의 더 많은 정보는 아래의 내용을 참조한다.

- [데이터베이스 서버 접속 제한](#)
- [패킷 암호화](#)
- [특정 브로커 관리](#)
- [브로커 설정 정보 확인](#)
- [브로커 로그](#)

## CUBRID 매니저 서버

### CUBRID 매니저 서버 구동

CUBRID 매니저 서버를 구동하기 위하여 다음과 같이 입력한다.

```
% cubrid manager start
```

이미 CUBRID 매니저 서버가 구동 중에 있다면 다음과 같은 메시지가 출력된다.

```
% cubrid manager start
@ cubrid manager server start
++ cubrid manager server is running.
```

## CUBRID 매니저 서버 종료

CUBRID 매니저 서버를 종료하기 위하여 다음과 같이 입력한다.

```
% cubrid manager stop
@ cubrid manager server stop
++ cubrid manager server stop: success
```

## CUBRID 매니저 서버 로그

CUBRID 매니저 서버와 관련된 로그는 설치 디렉터리의 log/manager 디렉터리에 저장되며, 매니저 서버 프로세스에 따라 다음과 같이 네 종류의 로그 파일로 저장된다.

- auto\_backupdb.log: 매니저 클라이언트에서 예약된 백업 자동화 작업에 관한 로그
- auto\_execquery.log: 매니저 클라이언트에서 예약된 쿼리 자동화 작업에 관한 로그
- cub\_js.access.log: 매니저 서버에 성공한 로그인과 작업에 대한 로그
- cub\_js.error.log: 매니저 서버에 실패한 로그인과 작업에 관한 로그

## CUBRID 매니저 서버 환경 설정

CUBRID 매니저 서버의 환경 설정 파일은 **\$CUBRID/conf**에 위치하며, 파일 이름은 **cm.conf**이다. CUBRID 매니저 서버의 환경 설정 파일에서 주석은 “#”으로 처리되며, 매개 변수 이름과 값이 저장된다. 매개 변수 이름과 값 사이에는 공백 또는 등호 부호(=)로 구분한다.

**cm.conf**에서 설정할 수 있는 매개 변수는 다음과 같다.

### cm\_port

CUBRID 매니저 서버와 클라이언트 사이의 통신 포트를 설정하는 매개 변수로, 기본값은 **8001**로 설정된다.

### monitor\_interval

**cub\_auto**의 모니터링 주기를 초 단위로 설정하는 매개 변수로, 기본값은 **5**이다.

### allow\_user\_multi\_connection



CUBRID 매니저 서버에 클라이언트가 중복 접속하는 것을 허용하기 위한 매개 변수로, 기본값은 **YES**이다. 따라서 CUBRID 매니저 서버에는 두 개 이상의 CUBRID 매니저 클라이언트가 접속할 수 있으며, 같은 사용자 이름으로 접속할 수도 있다.

### server\_long\_query\_time

서버의 진단 항목 중 **slow\_query** 항목을 설정할 경우 몇 초 이상을 늦은 질의로 판별할 지 결정하는 매개 변수로, 기본 값은 **10**이다. 서버에서 수행된 질의 수행 시간이 매개 변수 설정 값보다 큰 경우, **slow\_query**의 개수가 증가한다.

### auto\_job\_timeout

**auto\_job\_timeout** 는 작업 자동화(cub\_auto)의 작업이 유지되기 위한 최대 시간이다. 기본값은 43,200 (12 시간)이다.

### mon\_cub\_auto

**mon\_cub\_auto**는 cub\_auto가 종료되면 자동으로 재시작할 것인지 설정한다. 기본값은 NO 이다.

### token\_active\_time

**token\_active\_time**는 로그인된 세션의 최대 유지 시간을 설정한다. 기본값은 7200 (2 시간)이다.

### support\_mon\_statistic

**support\_mon\_statistic**는 누적 모니터링을 사용할 것인지 설정한다. 기본값은 NO 이다.

### cm\_process\_monitor\_interval

**cm\_process\_monitor\_interval**는 모니터링 정보 수집 주기이다. 기본값과 최소값은 5 (5 분)이다.

## CUBRID 매니저 사용자 관리 콘솔

CUBRID 매니저 사용자의 계정과 비밀번호는 CUBRID 매니저 클라이언트 구동을 시작할 때 CUBRID 매니저 서버에 접속하기 위해 사용하는 것이며, 데이터베이스 사용자와는 다른 개념이다. CUBRID 매니저 관리자(cm\_admin)는 사용자 정보를 관리하는 CLI 도구로, 콘솔 창에서 명령어를 실행하여 사용자를 관리한다. 이 유틸리티는 Linux OS만 지원한다.

다음은 CUBRID 매니저(이하 CM) 관리자 유틸리티 구문 사용법이다. 아래 기능은 CUBRID 매니저 클라이언트에서 GUI를 통해 사용할 수도 있다.

```
cm_admin <utility_name>
<utility_name>:
 adduser [<option>] <cmuser-name> <cmuser-password> --- CM 사용자 추가
```

```
deluser <cmuser-name> --- CM 사용자 삭제
viewuser [<cmuser-name>] --- CM 사용자 정보 출력
changeuserauth [<option>] <cmuser-name> --- CM 사용자 권한 변경
changeuserpwd [<option>] <cmuser-name> --- CM 사용자 비밀번호 변경
adddbinfo [<option>] <cmuser-name> <database-name> --- CM 사용자
의 데이터베이스 정보 추가
deldbinfo <cmuser-name> <database-name> --- CM 사용자의 데이터베이
스 정보 삭제
changedbinfo [<option>] <database-name> number-of-pages --- CM 사
용자의 데이터베이스 정보 변경
```

## CM 사용자

CM 사용자 정보는 다음과 같은 정보로 구성된다.

- CM 사용자 권한: 다음과 같은 권한 정보를 포함한다.
  - 브로커 권한
  - 데이터베이스 생성 권한. 현재는 **admin** 사용자만 이 권한을 가질 수 있다.
  - 상태 모니터링 권한
- 데이터베이스 정보: CM 사용자가 사용할 수 있는 데이터베이스
- CM 사용자 비밀번호

CUBRID 매니저의 기본 사용자는 모든 관리 권한을 가진 **admin** 사용자이며 기본 비밀번호는 admin이다.

## CM 사용자 추가

**cm\_admin adduser** 유틸리티는 특정 권한과 데이터베이스 정보를 갖는 CM 사용자를 생성한다. 브로커 권한, 데이터베이스 생성 권한 및 상태 모니터링 권한 등을 CM 사용자에게 부여할 수 있다.

```
cm_admin adduser [options] cmuser-name cmuser-password
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **adduser**: 새 CM 사용자를 생성하는 명령어
- *cmuser-name*: 생성할 CM 사용자의 고유한 이름을 지정한다. 사용 가능한 문자는 0~9, A~Z, a~z, \_이며 최소 4자, 최대 32자이다. 지정한 *cmuser-name*이 기존 *cmuser-name*과 같으면 **cm\_admin**은 CM 사용자 생성을 중지한다.
- *cmuser-password*: 생성할 CM 사용자의 비밀번호이다. 사용 가능한 문자는 0~9, A~Z, a~z, \_이며 최소 4자, 최대 32자이다.

다음은 **cm\_admin adduser**에 대한 [options]이다.

#### **-b, --broker AUTHORITY**

생성할 CM 사용자의 브로커 권한을 지정한다. 사용할 수 있는 값은 **admin**, **none**, **monitor**이며, 기본값은 **none**이다.

다음은 이름이 *testcm*이고 비밀번호가 *testcmpwd*인 CM 사용자를 생성하고 브로커 권한을 **monitor**로 설정하는 예이다.

```
cm_admin adduser -b monitor testcm testcmpwd
```

#### **-c, --dbcreate AUTHORITY**

생성할 CM 사용자의 데이터베이스 생성 권한을 지정한다. 사용할 수 있는 값은 **none**, **admin**이며, 기본값은 **none**이다.

다음은 이름이 *testcm*이고 비밀번호가 *testcmpwd*인 CM 사용자를 생성하고 데이터베이스 생성 권한을 **admin**으로 설정하는 예이다.

```
cm_admin adduser -c admin testcm testcmpwd
```

#### **-m, --monitor AUTHORITY**

생성할 CM 사용자의 모니터링 권한을 지정한다. 사용할 수 있는 값은 **admin**, **none**, **monitor**이며, 기본값은 **none**이다.

다음은 이름이 *testcm*이고 비밀번호가 *testcmpwd*인 CM 사용자를 생성하고 상태 모니터링 권한을 **admin**으로 설정하는 예이다.

```
cm_admin adduser -m admin testcm testcmpwd
```

#### **-d, --dbinfo INFO\_STRING**

생성할 CM 사용자의 데이터베이스 정보를 지정한다. *INFO\_STRING*은 "<dbname>;<uid>;<broker\_ip>;<broker\_port>"의 형식으로 지정해야 한다.

다음은 이름이 *testcm*인 CM 사용자에게 "testdb;dba;localhost,30000"이라는 데이터베이스 정보를 추가하는 예이다.

```
cm_admin adduser -d "testdb;dba;localhost,30000" testcm testcmpwd
```

## CM 사용자 삭제

**cm\_admin deluser** 유틸리티는 지정한 CM 사용자 이름을 기준으로 CM 사용자를 삭제한다.

```
cm_admin deluser cmuser-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **deluser**: 기존 CM 사용자를 삭제하는 명령어
- *cmuser-name*: 삭제할 CM 사용자 이름

다음은 이름이 *testcm*인 CM 사용자를 삭제하는 예이다.

```
cm_admin deluser testcm
```

## CM 사용자 정보 출력

**cm\_admin viewuser** 유틸리티는 지정한 CM 사용자 이름을 기준으로 CM 사용자를 삭제한다.

```
cm_admin viewuser cmuser-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **viewuser**: CM 사용자의 권한 및 데이터베이스 정보를 출력하는 명령어
- *cmuser-name*: CM 사용자 이름. 이 값을 입력하면 해당 사용자의 정보만 출력하고, 생략하면 모든 기존 CM 사용자 정보를 표시한다.

다음은 이름이 *testcm*인 CM 사용자의 정보를 출력하는 예이다.

```
cm_admin viewuser testcm
```

다음과 같이 정보가 출력된다.

```
CM USER: testcm
Auth info:
 broker: none
 dbcreate: none
 statusmonitorauth: none
DB info:

=====
=====
 DBNAME
UID BROKER INFO
=====
```

```
=====
testdb
dba localhost,30000
```

## CM 사용자 권한 변경

**cm\_admin changeuserauth** 유틸리티는 CM 사용자의 권한을 변경한다.

```
cm_admin changeuserauth [options] cmuser-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **changeuserauth**: CM 사용자의 권한 정보를 변경하는 명령어
- *cmuser-name*: 권한을 변경할 CM 사용자의 이름

**cm\_admin changeuserauth**에서 사용하는 [options]는 다음과 같다.

### -b, --broker

CM 사용자의 브로커 권한을 지정한다. 사용할 수 있는 값은 **admin**, **none**, **monitor**이다.

다음은 이름이 *testcm*인 CM 사용자의 브로커 권한을 **monitor**로 변경하는 예이다.

```
cm_admin changeuserauth -b monitor testcm
```

### -c, --dbcreate

CM 사용자의 데이터베이스 생성 권한을 지정한다. 사용할 수 있는 값은 **none**, **admin**이다.

다음은 이름이 *testcm*인 CM 사용자의 데이터베이스 생성 권한을 **admin**으로 변경하는 예이다.

```
cm_admin changeuserauth -c admin testcm
```

### -m, --monitor

CM 사용자의 모니터링 권한을 지정한다. 사용할 수 있는 값은 **admin**, **none**, **monitor**이다.

다음은 이름이 *testcm*인 CM 사용자의 상태 모니터링 권한을 **admin**으로 변경하는 예이다.

```
cm_admin changeuserauth -m admin testcm
```

## CM 사용자 비밀번호 변경

**cm\_admin changeuserpwd** 유틸리티는 CM 사용자의 비밀번호를 변경한다.

```
cm_admin changeuserpwd [options] cmuser-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **changeuserpwd**: CM 사용자의 비밀번호를 변경하는 명령어
- *cmuser-name*: 비밀번호를 변경할 CM 사용자의 이름

**cm\_admin changeuserpwd**에서 사용하는 [options]는 다음과 같다.

### -o, --oldpass PASSWORD

CM 사용자의 예전 비밀번호를 지정한다.

다음은 이름이 *testcm*인 CM 사용자의 비밀번호를 변경하는 예이다.

```
cm_admin changeuserpwd -o old_password -n new_password
testcm
```

### --adminpass PASSWORD

사용자의 예전 비밀번호를 모를 때 **admin**의 비밀번호를 대신 지정할 수 있다.

다음은 **admin** 비밀번호를 사용하여 이름이 *testcm*인 CM 사용자의 비밀번호를 변경하는 예이다.

```
cm_admin changeuserauth --adminpass admin_password -n
new_password testcm
```

### -n, --newpass PASSWORD

CM 사용자의 새 비밀번호를 지정한다.

## CM 사용자의 데이터베이스 정보 추가

**cm\_admin adddbinfo** 유틸리티는 데이터베이스 정보(데이터베이스 이름, UID, 브로커 IP 및 브로커 포트)를 CM 사용자에게 추가한다.

```
cm_admin adddbinfo [options] cmuser-name database-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **adddbinfo**: CM 사용자에게 데이터베이스 정보를 추가하는 명령어

- `cmuser-name`: CM 사용자 이름
- `database-name`: 추가할 데이터베이스 이름

다음은 이름이 `testcm`인 CM 사용자에게 이름이 `testdb`이고 기본값을 사용하는 데이터베이스를 추가하는 예이다.

```
cm_admin adddbinfo testcm testdb
```

다음은 `cm_admin adddbinfo`에서 사용하는 [options]이다.

#### **-u, --uid ID**

추가할 데이터베이스의 사용자 ID를 지정한다. 기본값은 **dba**이다.

다음은 이름이 `testcm`인 CM 사용자에게 이름이 `testdb`이고 사용자 ID가 `cubriduser`인 데이터베이스를 추가하는 예이다.

```
cm_admin adddbinfo -u cubriduser testcm testdb
```

#### **-h, --host IP**

클라이언트가 데이터베이스에 접속할 때 사용하는 브로커의 호스트 IP를 지정한다. 기본값은 **localhost**이다.

다음은 이름이 `testcm`인 CM 사용자에게 이름이 `testdb`이고 호스트 IP가 `127.0.0.1`인 데이터베이스를 추가하는 예이다.

```
cm_admin adddbinfo -h 127.0.0.1 testcm testdb
```

#### **-p, --port NUMBER**

클라이언트가 데이터베이스에 접속할 때 사용하는 브로커의 포트 번호를 지정한다. 기본값은 **30000**이다.

### CM 사용자의 데이터베이스 정보 삭제

`cm_admin deldbinfo` 유틸리티는 지정한 CM 사용자의 데이터베이스 정보를 삭제한다.

```
cm_admin deldbinfo cmuser-name database-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **deldbinfo**: CM 사용자의 데이터베이스 정보를 삭제하는 명령어
- `cmuser-name`: CM 사용자 이름

- `database-name`: 삭제할 데이터베이스 이름

다음은 이름이 *testcm*인 CM 사용자에게서 이름이 *testdb*인 데이터베이스 정보를 삭제하는 예이다.

```
cm_admin deldbinfo testcm testdb
```

## CM 사용자의 데이터베이스 정보 변경

**cm\_admin changedbinfo** 유틸리티는 지정한 CM 사용자의 데이터베이스 정보를 변경한다.

```
cm_admin changedbinfo [options] cmuser-name database-name
```

- **cm\_admin**: CUBRID 매니저를 관리하는 통합 유틸리티
- **changedbinfo**: CM 사용자의 데이터베이스 정보를 변경하는 명령어
- `<cmuser-name>`: CM 사용자 이름
- `<database-name>`: 변경할 데이터베이스 이름

다음은 **cm\_admin changedbinfo**에서 사용하는 [options]이다.

### -u, --uid ID

데이터베이스의 사용자 ID를 지정한다.

다음은 이름이 *testcm*인 CM 사용자의 *testdb* 데이터베이스에서 사용자 ID 정보를 *uid*로 업데이트하는 예이다.

```
cm_admin changedbinfo -u uid testcm testdb
```

### -h, --host IP

클라이언트가 데이터베이스에 접속할 때 사용하는 브로커의 호스트를 지정한다.

다음은 이름이 *testcm*인 CM 사용자의 *testdb* 데이터베이스에서 호스트 IP 정보를 10.34.63.132로 업데이트하는 예이다.

```
cm_admin changedbinfo -h 10.34.63.132 testcm testdb
```

### -p, --port NUMBER

클라이언트가 데이터베이스에 접속할 때 사용하는 브로커의 포트 번호를 지정한다.



다음은 이름이 *testcm*인 CM 사용자의 *testdb* 데이터베이스에서 브로커 포트 정보를 33000로 업데이트하는 예이다.

```
cm_admin changedbinfo -p 33000 testcm testdb
```

## CUBRID 프로시저 언어 서버

### CUBRID 프로시저 언어 서버 구동

CUBRID 프로시저 언어 서버는 데이터베이스 서버 구동 시 자동으로 시작되며, 데이터베이스 서버 종료 시 프로시저 언어 서버를 종료한다.

프로시저 언어 서버를 데이터베이스 서버와 함께 구동하고 싶지 않으면 CUBRID 설정 파일 (**cubrid.conf**)에서 해당하는 데이터베이스에 대해 **stored\_procedure** 값을 **no**로 설정한다.

```
% cubrid server start demodb
@ cubrid pl status
```

CUBRID 프로시저 언어 서버 시작 시 발생할 수 있는 다른 유형의 오류에 대한 자세한 내용은 **CUBRID 프로시저 언어 에러** 를 참고한다.

### CUBRID 프로시저 언어 서버 재시작

다음은 *demodb* 용 프로시저 언어 서버를 재시작할 수 있는 방법이다. 프로시저 언어 서버의 재시작 명령 시 프로시저 언어 서버가 종료되고 데이터베이스 서버에 의해 다시 시작된다. 데이터베이스 서버가 구동 중이지 않으면 재시작 명령은 동작하지 않는다.

```
% cubrid pl restart demodb
@ cubrid pl restart: demodb
++ cubrid pl restart: success
```

..note:

데이터베이스 서버가 자동으로 프로시저 언어 서버를 구동하므로 재시작 명령어는 자주 사용되지 않는다.  
프로시저를 사용한 대량의 입출력 작업이나 분석 작업을 수행한 후 프로시저 언어 서버의 VM에 의한 GC가 느린 경우 서버의 안정화를 위해 사용할 수 있다.

**⚠ 경고**

프로시저 언어 서버를 재시작하면 현재 실행 중인 트랜잭션이 인터럽트되며 롤백된다.

## CUBRID 프로시저 언어 서버 상태 확인

다음은 *demodb* 용 CUBRID 프로시저 언어 서버의 상태를 확인하는 예시이다. 프로시저 언어 서버가 현재 실행 중인 대상 데이터베이스의 이름, *demodb* 가 출력된다. 또한 서버의 PID, 포트 번호와 적용된 JVM 옵션이 함께 표시된다.

```
% cubrid pl status demodb

@ cubrid pl status: demodb
Procedural Language Server (demodb, pid 9220, UDS)
VM arguments :

...

```

## 프로시저 언어 서버 설정

### 프로시저 언어 환경: 번들 JDK 사용

CUBRID에서 프로시저 언어 기능을 사용하기 위해 JVM 환경을 사용한다. CUBRID 서버는 설치 시 `$CUBRID/vm` 디렉토리에 Eclipse Temurin JDK 8 버전을 포함하므로 사용자의 환경에 별도로 JDK 환경을 구성할 필요가 없다. 이 번들 JDK는 CUBRID에서 프로시저 언어 기능의 사용에 대해서 검증되어 있으며 일반적으로 좋은 성능과 안정성을 제공할 수 있으므로 사용을 권장한다. 번들 JDK 대신 사용자의 환경에 구성된 JDK를 사용하기 위해 다음의 **CUBRID\_JAVA\_HOME** 환경 변수를 설정할 수 있다.

### CUBRID\_JAVA\_HOME

**CUBRID\_JAVA\_HOME** 환경 변수는 JDK 설치 디렉토리를 지정한다. 사용자가 **CUBRID\_JAVA\_HOME**을 환경 변수를 지정하지 않으면 CUBRID 설치 시 포함된 번들 JDK를 사용한다. 만약 올바르게 않은 **CUBRID\_JAVA\_HOME**을 지정하면 프로시저 언어 서버가 제대로 시작되지 않으며 오류를 반환할 수 있다. **CUBRID\_JAVA\_HOME**의 경로에는 JDK 환경과 함께 `libjvm` 라이브러리가 함께 포함되어 있어야 한다.

다음은 **CUBRID\_JAVA\_HOME** 환경 변수를 설정하는 예시이다.

```
Windows 환경에서 설정 예시
```

```
set CUBRID_JAVA_HOME=C:\Program Files\Java\jdk1.8.0_421
```

**\*\*Linux 환경에서 설정 예시\*\***

```
export CUBRID_JAVA_HOME=/usr/java/jdk1.8.0_421
```

## 프로시저 언어 서버 시스템 파라미터

다음 표는 설정 파일 (**cubrid.conf**)에서 설정할 수 있는 프로시저 언어 서버 관련 서버 파라미터이다.

| 파라미터 이름                     | 타입     | 기본값 | 최소값 | 최대값   |
|-----------------------------|--------|-----|-----|-------|
| stored_procedure            | bool   | yes |     |       |
| stored_procedure_uds        | bool   | yes |     |       |
| stored_procedure_port       | int    | 0   | 0   | 65535 |
| stored_procedure_vm_options | string |     |     |       |

이 파라미터에 대한 자세한 사항은 **cubrid.conf 설정 파일과 기본 제공 파라미터** 를 참고한다.

## CUBRID 프로시저 언어 서버 로그

CUBRID 프로시저 언어 서버의 로그는 설치 디렉터리의 **log/** 에 저장된다. 각 데이터베이스 별로 다음과 같은 로그 파일이 생성된다.

- 에러 로그 (\$CUBRID/log/[db\_name]\_pl.err)
- 자바 로그 (\$CUBRID/log/[db\_name]\_pl.log)

### 에러 로그

각 데이터베이스 별 프로시저 언어 서버의 에러 로그는 **\$CUBRID/log** 디렉터리에 저장되며, 파일 이름은 **<db\_name>\_pl.err** 형식으로 저장된다. 확장자는 **.err** 이다.

```
demodb_pl.err
```

프로시저 언어 서버를 시작하는 동안 에러가 발생하면 에러 메시지가 에러 로그 파일에 저장된다.

```
Time: 11/11/20 18:17:15.438 - ERROR *** file ../../src/jsp/jsp_sr.c,
line 501 ERROR CODE = -900, Tran = -1, EID = 1
Java 가상 머신 라이브러리를 찾을 수 없습니다:
Failed to get 'JVM_PATH' environment variable.
Failed to load libjvm from 'JAVA_HOME' environment variable:
/jre/lib/amd64/server/libjvm.so: cannot open shared object
file: No such file or directory
/lib/server/libjvm.so: cannot open shared object file: No
such file or directory.
```

### 참고

For more details on what errors can be occurred, see [CUBRID 프로시저 언어 에러](#).

## 예외 로그

프로시저 실행 중 발생하는 VM 예외 메시지를 저장한다. 각 데이터베이스 별 프로시저 언어 서버의 예외 로그는 **\$CUBRID/log** 디렉터리에 저장되며, 파일 이름은 **<db\_name>\_pl.log** 형식으로 저장된다. 확장자는 **.log** 이다.

```
demodb_pl.log
```

VM에서 저장 프로시저/함수를 수행하는 동안 예외가 발생하면 예외 문자열이 로그에 저장된다.

```
SEVERE:
java.lang.NullPointerException
at Test.testFunction(Test.java:50)
...
at com.cubrid.jsp.StoredProcedure.invoke(StoredProcedure.java:263)
at com.cubrid.jsp.ExecuteThread.run(ExecuteThread.java:197)
```

## CUBRID 프로시저 언어 에러

다음은 CUBRID 프로시저 언어 서버 시작 시 발생할 수 있는 에러에 대한 에러 메시지이다. 에러 메시지는 **\$CUBRID/log/<db\_name>\_pl.err** 에 저장된다.

| 에러 코드 | 에러 메시지                         | 설명                                                                                     | 조치사항                                                                                                              |
|-------|--------------------------------|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| -900  | 가상 머신 라이브러리를 찾을 수 없습니다: %1\$s. | CUBRID 가 CUBRID_JAVA_HOME 환경 변수에서 JVM 라이브러리를 찾을 수 없음                                   | CUBRID_JAVA_HOME 환경 변수가 올바르게 설정 되었는지 확인한다. <b>프로시저 언어 환경: 번들 JDK 사용</b> 를 참고한다.                                   |
| -901  | PL 서버를 시작할 수 없습니다: %1\$s.      | JVM 라이브러리 내에서 예상치 못한 에러가 발생 JVM 라이브러리 또는 \$CUBRID/java/jspserver.jar 에서 문제가 발생할 가능성 있음 | JRE 재설치를 시도해보고 만약 동일한 에러가 발생하면 다른 버전의 JRE를 설치를 시도한다. 그리고 \$CUBRID/java/jspserver.jar 파일을 동일한 CUBRID 버전의 것으로 교체한다. |

다음은 CUBRID 프로시저 언어 서버가 시작되지 않은 경우를 포함하여 연결에 문제가 있을 때 발생할 수 있는 에러에 대한 에러 메시지이다. 에러 메시지는 **\$CUBRID/log/broker/error\_log/<broker\_name>\_<app\_server\_num>.err** 에 저장된다.

| 에러 코드 | 에러 메시지                   | 설명                                                                    | 조치사항                                                                                                   |
|-------|--------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| -902  | PL 서버가 실행되지 않았습니다.       | 프로시저 언어 서버가 시작되지 않음                                                   | <b>cubrid pl start &lt;db_name&gt;</b> 명령어로 프로시저 언어 서버를 시작한다. 자세한 설명은 <b>CUBRID 프로시저 언어 서버</b> 를 참고한다. |
| -903  | PL 서버에 접속할 수 없습니다: %1\$s | 프로시저 언어 서버가 CAS로부터 연결할 수 없음 이 에러는 여러가지 이유로 발생할 수 있다. 예를 들어 프로시저 언어 서버 | 프로시저 언어 서버를 재시작한다. 만약 재시작을 실패하면 <b>cub_pl &lt;db_name&gt;</b> 프로세스를 리눅스 <b>kill</b> 명령어로 강제로           |

| 에러 코드 | 에러 메시지 | 설명                                                                                         | 조치사항                                                                                                                                                                                                                                                                         |
|-------|--------|--------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|       |        | 버가 불안정하거나 CAS에서 프로시저 언어 서버에 연결할 수 없는 경우, 또는 프로시저 언어가 예기치 않게 종료(kill)된 경우 이러한 에러 메시지를 출력한다. | <p>종료 한다. 그리고 다시 프로시저 언어 서버를 재시작한다.</p> <p><b>cubrid pl status &lt;db_name&gt;</b> 명령어를 통해 프로시저 언어 서버의 포트로는 CAS에서 접근 가능한지 확인한다. 방화벽에 의해 해당 포트가 막혀있을 수 있으므로 방화벽에서 포트를 열어준다. 필요한 경우 <b>stored_procedure_port</b> 파라미터를 설정하고 프로시저 언어 서버를 재시작한다. 자세한 사항은 <b>포트 설정</b>을 참고한다.</p> |

-905 PL 서버와 통신 중 오류가 발생하였습니다:  
%1\$d | CAS가 프로시저 언어 서버로부터 잘못된 패킷을 받음

## 데이터베이스 관리

### 데이터베이스 사용자

CUBRID 데이터베이스 사용자는 동일한 권한을 갖는 멤버를 가질 수 있다. 사용자에게 권한 **A**가 부여되면, 상기 사용자에게 속하는 모든 멤버에게도 권한 **A**가 동일하게 부여된다. 이와 같이 데이터베이스 사용자와 그에 속한 멤버를 '그룹'이라 하고, 멤버가 없는 사용자를 '사용자'라 한다.

CUBRID는 **DBA**와 **PUBLIC**이라는 사용자를 기본으로 제공한다.

- **DBA**는 모든 사용자의 멤버가 되며 데이터베이스의 모든 객체에 접근할 수 있는 최고 권한 사용자이다. 또한, **DBA**만이 데이터베이스 사용자를 추가, 편집, 삭제할 수 있는 권한을 갖는다.

- **DBA**를 포함한 모든 사용자는 **PUBLIC**의 멤버가 되므로 모든 데이터베이스 사용자는 **PUBLIC**에 부여된 권한을 가진다. 예를 들어, **PUBLIC** 사용자에게 권한 **B**를 추가하면 데이터베이스의 모든 사용자에게 일괄적으로 권한 **B**가 부여된다. 사용자의 추가, 권한 부여에 대한 자세한 내용은 [사용자 관리](#)를 참고한다.

## databases.txt 파일

CUBRID에 존재하는 모든 데이터베이스의 위치 정보는 **databases.txt** 파일에 저장하는데, 이를 데이터베이스 위치 정보 파일이라 한다. 이러한 데이터베이스 위치 정보 파일은 데이터베이스의 생성, 이름 변경, 삭제 및 복사에 관한 유틸리티를 수행하거나 각 데이터베이스를 구동할 때에 사용되며, 기본으로는 설치 디렉터리의 **databases** 디렉터리에 위치하고, **CUBRID\_DATABASES** 환경 변수로 디렉터리 위치를 지정할 수 있다.

```
db_name db_directory server_host logfile_directory
```

데이터베이스 위치 정보 파일의 라인별 형식은 구문에 정의된 바와 같으며, 데이터베이스 이름, 데이터베이스 경로, 서버 호스트 및 로그 파일의 경로에 관한 정보를 저장한다. 다음은 데이터베이스 위치 정보 파일의 내용을 확인한 예이다.

```
% more databases.txt

dist_testdb /home1/user/CUBRID/bin d85007 /home1/user/CUBRID/bin
dist_demodb /home1/user/CUBRID/bin d85007 /home1/user/CUBRID/bin
testdb /home1/user/CUBRID/databases/testdb d85007 /home1/user/CUBRID/
databases/testdb
demodb /home1/user/CUBRID/databases/demodb d85007 /home1/user/CUBRID/
databases/demodb
```

데이터베이스 위치 정보 파일의 저장 디렉터리는 기본적으로 설치 디렉터리의 **databases** 디렉터리로 지정되며, 시스템 환경 변수 **CUBRID\_DATABASES**의 설정을 변경하여 기본 디렉터리를 변경할 수 있다. 데이터베이스 위치 정보 파일의 저장 디렉터리 경로가 유효해야 데이터베이스 관리를 위한 **cubrid** 유틸리티가 데이터베이스 위치 정보 파일에 접근할 수 있게 된다. 이를 위해서 사용자는 디렉터리 경로를 정확하게 입력해야 하고, 해당 디렉터리 경로에 대해 쓰기 권한을 가지는지 확인해야 한다. 다음은 **CUBRID\_DATABASES** 환경 변수에 설정된 값을 확인하는 예이다.

```
% set | grep CUBRID_DATABASES
CUBRID_DATABASES=/home1/user/CUBRID/databases
```

만약 **CUBRID\_DATABASES** 환경 변수에서 유효하지 않은 디렉터리 경로가 설정되는 경우에는 에러가 발생하며, 설정된 디렉터리 경로는 유효하나 데이터베이스 위치 정보 파일이 존재하지 않는 경우에는 새로운 위치 정보 파일을 생성한다. 또한, **CUBRID\_DATABASES** 환경 변수가 아예 설정되지 않은 경우에는 현재 작업 디렉터리에서 위치 정보 파일을 검색한다.

## 데이터베이스 볼륨

CUBRID 데이터베이스의 볼륨은 크게 영구적 볼륨, 일시적 볼륨, 백업 볼륨으로 분류한다.

- 영구적 볼륨 중
  - 영구적 데이터를 저장하지만 일시적 데이터도 저장할 수 있는 데이터 볼륨이 있다.
  - 데이터베이스의 최근 변경 사항을 포함하고 있는 활성 로그(active log) 볼륨이 있다.
- 일시적 볼륨 중
  - 임시 데이터를 저장할 수 있는 일시적 볼륨(temporary temp volume)이 있다.
  - 활성 로그에서 밀려난 로그를 저장하기 위한 보관 로그(archive log) 볼륨이 있다.
  - 백그라운드에서 로그 보관 작업을 수행할 때 사용되는 백그라운드 보관 로그(background archive log) 볼륨이 있다.

볼륨에 대한 자세한 내용은 [데이터베이스 볼륨 구조](#)를 참고한다.

다음은 *testdb* 데이터베이스를 운영할 때 발생하는 데이터베이스 관련 파일의 예이다.

| 파일 이름       | 크기    | 종류             | 분류              | 설명                                                                                                                                                                                  |
|-------------|-------|----------------|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| testdb      | 512MB | permanent data | Database volume | DB 생성시 최초로 생성되는 볼륨<br>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)<br>이 볼륨은 데이터베이스 메타 정보를 포함한다.<br><b>cubrid createdb</b> 는 <b>cubrid.conf</b> 의 <b>db_volume_size</b> 에 명시된 디폴트 크기를 사용한다. |
| testdb_perm | 512MB | permanent data |                 | <b>cubrid addvoldb</b> 유틸                                                                                                                                                           |



| 파일 이름       | 크기    | 종류             | 분류 | 설명                                                                                                                |
|-------------|-------|----------------|----|-------------------------------------------------------------------------------------------------------------------|
|             |       |                |    | 리터를 사용해 수동으로 생성된 볼륨<br>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)                                                      |
| testdb_temp | 512MB | temporary data |    | <b>cubrid</b><br><b>addvoldb</b> 유틸리터를 사용해 수동으로 생성된 볼륨<br>이 볼륨은 임시 데이터를 저장한다. (질의 결과, 리스트 파일들, 정렬 파일들, 결합 객체 해시들) |
| testdb_x003 | 512MB | permanent data |    | 데이터베이스의 공간이 더 필요할 때 자동으로 생성<br>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)                                              |
| testdb_x004 | 512MB | permanent data |    | 데이터베이스의 공간이 더 필요할 때 자동으로 생성<br>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)                                              |
| testdb_x005 | 512MB | permanent data |    | 데이터베이스의 공간이 더 필요                                                                                                  |

| 파일 이름          | 크기    | 종류                   | 분류               | 설명                                                                                                           |
|----------------|-------|----------------------|------------------|--------------------------------------------------------------------------------------------------------------|
|                |       |                      |                  | <p>할 때 자동으로 생성</p> <p>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)</p>                                              |
| testdb_x006    | 64MB  | permanent data       |                  | <p>데이터베이스의 공간이 더 필요할 때 자동으로 생성</p> <p>이 볼륨은 영구 데이터를 저장한다. (시스템, 힙과 인덱스 파일들)</p> <p>이 볼륨의 크기는 아직 최대화되지 않음</p> |
| testdb_t32766  | 512MB | temporary data       | Temporary Volume | <p>데이터베이스의 공간이 더 필요할 때 자동으로 생성</p> <p>이 볼륨은 임시 데이터를 저장한다. (질의 결과, 리스트 파일들, 정렬 파일들, 결합 객체 해시들)</p>            |
| testdb_lgar_t  | 512MB | background archiving | Log volume       | <p>백그라운드 보관 (background archiving) 기능과 관련된 로그 파일</p> <p>보관 로그를 저장할 때 사용된다.</p>                               |
| testdb_lgar224 | 512MB | archive              |                  |                                                                                                              |

| 파일 이름       | 크기    | 종류     | 분류 | 설명                                                                                                                                                                                                                                                                   |
|-------------|-------|--------|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|             |       |        |    | <p>보관 로그 (archiving log)가 계속 쌓이면서 세 자리 숫자로 끝나는 파일들이 생성된다. cubrid backupdb -r 옵션 또는 cubrid.conf의 log_max_archives 파라미터의 설정으로 인해 001~223까지의 보관 로그들은 정상적으로 삭제된 것으로 보인다. 보관 로그가 삭제되는 경우, lginf 파일의 REMOVE 섹션에서 삭제된 보관 로그 번호를 확인할 수 있다.</p> <p><b>보관 로그 관리</b>를 참고한다.</p> |
| testdb_lgat | 512MB | active |    | <p>활성로그(Active log) 파일</p>                                                                                                                                                                                                                                           |

| 파일 이름      | 크기  | 종류             | 분류                  | 설명                                                                |
|------------|-----|----------------|---------------------|-------------------------------------------------------------------|
| testdb_dwb | 1MB | temporary data | Double write buffer | 플러시 (flush) 대상 페이지를 먼저 쓰기 위한 이중 쓰기 버퍼 (Double Write Buffer) 저장 파일 |

#### · 데이터베이스 볼륨 파일

- 위의 표에서 *testdb*, *testdb\_perm*, *testdb\_temp*, *testdb\_x003* ~ *testdb\_x006* 은 데이터베이스 볼륨 파일로 분류된다.
- 파일 크기는 **cubrid createdb** 및 **cubrid addvoldb** 의 **--db-volume-size** 옵션과 **cubrid.conf** 의 **db\_volume\_size** 에 의해 결정된다.
- 데이터베이스에 저장 공간이 부족해지면 새 볼륨이 자동 생성된다.

#### · 일시적 볼륨

- 일시적 볼륨은 일반적으로 일시적 데이터를 저장하는 데 사용된다. 이 볼륨은 데이터베이스 별로 자동 생성되고 삭제된다.
- 파일 크기는 **cubrid.conf** 의 **db\_volume\_size** 에 의해 결정된다.

#### · 로그 볼륨 파일

- 위의 표에서 *testdb\_lgar\_t*, *testdb\_lgar224* 및 *testdb\_lgat* 는 로그 볼륨 파일로 분류된다.
- 파일 크기는 **cubrid.conf** 의 **log\_volume\_size** 또는 **cubrid createdb** 의 **--log-volume-size** 옵션에 의해 결정된다.

#### · 이중 쓰기 버퍼 (Double Write Buffer, DWB) 파일

- DWB 파일은 부분 쓰기 (Partial Write)로 인한 I/O 에러를 방지하기 위한 저장공간이다.
- 모든 데이터 페이지는 DWB 에 먼저 쓰여지고 난 후에 영구 데이터 볼륨에 있는 데이터 위치에 쓰여진다.
- 데이터베이스가 재시작될 때, 부분적으로 쓰여진 페이지들이 탐지되고 DWB 에서 대응되는 페이지로 대체된다.

- 파일 크기는 **cubrid.conf** 의 **double\_write\_buffer\_size** 에 의해 결정된다. 만약 0으로 설정되었다면, DWB 는 사용되지 않고 DWB 파일도 생성되지 않는다.

## 참고

데이터베이스 재시작과 비정상 종료 시에도 보존해야 하는 데이터는 영구적 데이터 용도로 생성된 데이터베이스 볼륨에 저장된다. 이 볼륨은 테이블 행(힙 파일), 인덱스(b-tree 파일) 및 여러 시스템 파일을 저장한다.

질의 처리 및 정렬의 중간 결과와 최종 결과의 경우 일시적 저장소만 필요하다. 요구되는 일시적 데이터 크기에 따라 우선적으로 메모리에 저장된다(공간 크기는 **cubrid.conf** 에 지정된 시스템 파라미터 **temp\_file\_memory\_size\_in\_pages** 에 의해 결정됨). 이를 초과하는 데이터는 디스크에 저장한다.

데이터베이스는 일시적 데이터를 위한 디스크 공간을 할당하기 위해 일반적으로 일시적 볼륨을 생성해 사용한다. 그러나 관리자는 **cubrid addvoldb -p temp** 명령을 사용해 일시적 데이터를 저장하기 위한 용도로 영구적 데이터베이스 볼륨을 할당할 수 있다. 이러한 영구적 데이터베이스 볼륨이 있는 경우 임시 데이터를 디스크 공간에 저장할 때 일시적 볼륨 보다 우선 사용한다.

일시적 데이터를 사용할 수 있는 질의의 예는 다음과 같다.

- **SELECT** 등의 결과 집합이 생성되는 질의
- **GROUP BY** 나 **ORDER BY** 가 포함된 질의
- 부질의(subquery)가 포함된 질의
- 정렬 병합(sort-merge) 조인이 수행되는 질의
- **CREATE INDEX** 질의문이 포함된 질의

일시적 데이터에 의해 시스템의 디스크 공간이 소진되는 것을 방지하려면 다음과 같이 조치할 것을 권장한다.

- 영구적 데이터베이스 볼륨을 미리 생성해 일시적 데이터에 필요한 저장 공간을 확보한다.
- **cubrid.conf** 에서 **temp\_file\_max\_size\_in\_pages** 파라미터를 설정해 질의를 수행할 때 일시적 볼륨에 사용되는 공간의 크기를 제한한다(기본적으로는 제한 없음).

## cubrid 유틸리티

cubrid 유틸리티의 사용법(구문)은 다음과 같다.

```
cubrid utility_name
utility_name :
 createdb [option] <database_name> <locale_name> --- 데이터베이스 생성
 deletedb [option] <database_name> --- 데이터베이스 삭제
 installdb [option] <database_name> --- 데이터베이스 설치
 renamedb [option] <source-database-name> <target-database-name>
--- 데이터베이스 이름 변경
 copydb [option] <source-database-name> <target-database-name>
--- 데이터베이스 복사
 backupdb [option] <database-name> --- 데이터베이스 백업
 restoredb [option] <database-name> --- 데이터베이스 복구
 addvolddb [option] <database-name> --- 데이터베이스 볼륨 파일 추가
 spacedb [option] <database-name> --- 데이터베이스 공간 정보 출력
 lockdb [option] <database-name> --- 데이터베이스의 lock 정보 출력
 tranlist [option] <database-name> --- 트랜잭션 확인
 killtran [option] <database-name> --- 트랜잭션 제거
 optimizedb [option] <database-name> --- 데이터베이스 통계 정보 갱신
 statdump [option] <database-name> --- 데이터베이스 서버 실행 통계 정보 출력
 compactdb [option] <database-name> --- 사용되지 않는 영역을 해제, 공간 최적화
 diagdb [option] <database-name> --- 내부 정보 출력
 checkdb [option] <database-name> --- 데이터베이스 일관성 검사
 alterdbhost [option] <database-name> --- 데이터베이스 호스트 변경
 plandump [option] <database-name> --- 쿼리 플랜 캐시 정보 출력
 loaddb [option] <database-name> --- 데이터 및 스키마 가져오기(로드)
 unloaddb [option] <database-name> --- 데이터 및 스키마 내보내기(언로드)
 paramdump [option] <database-name> --- 데이터베이스의 설정된 파라미터 값 확인
 changemode [option] <database-name> --- 서버의 HA 모드 출력 또는 변경
 applyinfo [option] <database-name> --- HA 환경에서 트랜잭션 로그 반영 정보를 확인하는 도구
 synccolldb [option] <database-name> --- DB 콜레이션을 시스템 콜레이션에 맞게 변경하는 도구
 genlocale [option] <database-name> --- 사용하고자 하는 로캘 정보를 컴파일하는 도구
 dumplocale [option] <database-name> --- 컴파일된 바이너리 로캘 정보를 사람이 읽을 수 있는 텍스트로 출력하는 도구
 gen_tz [option] [<database-name>] --- 공유 라이브러리로 컴파일할 수 있는 타임존 데이터가 포함된 C 소스 파일 생성
 dump_tz [option] --- 타임존 관련 정보 출력
 tde <operation> [option] <database-name> --- TDE 암호화 관리 도구
```

```
vacuumdb [option] <database-name> --- 데이터베이스의 삭제된 레코드 또는 불필요한 mvcc 관련 정보를 정리 및 관련 정보 확인하는 도구
flashback [option] <database-name> <owner_name.class_name> --- 커밋된 특정 트랜잭션을 되돌릴 수 있도록 SQL 구문을 제공하는 도구
memmon [option] <database-name> --- 현재 서버의 힙 메모리 사용량에 대한 정보를 확인하는 도구
```

## cubrid 유틸리티 로깅

CUBRID는 cubrid 유틸리티의 수행 결과에 대한 로깅 기능을 제공하며, 자세한 내용은 [cubrid 유틸리티 로깅](#) 을 참고한다.

## createdb

**cubrid createdb** 유틸리티는 CUBRID 시스템에서 사용할 데이터베이스를 생성하고 미리 만들어진 CUBRID 시스템 테이블을 초기화한다. 데이터베이스에 권한이 주어진 초기 사용자를 정의할 수도 있다. 일반적으로 데이터베이스 관리자만이 **cubrid createdb** 유틸리티를 사용한다. 로그와 데이터베이스의 위치도 지정할 수 있다.

### ⚠ 경고

데이터베이스를 생성할 때 데이터베이스 이름 뒤에 로캘 이름과 문자셋(예: ko\_KR.utf8)을 반드시 지정해야 한다. 문자셋에 따라 문자열 타입의 크기, 문자열 비교 연산 등에 영향을 끼친다. 데이터베이스 생성 시 지정된 문자셋은 변경할 수 없으므로 지정에 주의해야 한다.

문자셋, 로캘 및 콜레이션 설정과 관련된 자세한 내용은 [다국어 개요](#) 을 참고한다.

```
cubrid createdb [options] database_name locale_name.charset
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **createdb**: 새로운 데이터베이스를 생성하기 위한 명령이다.
- *database\_name*: 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않고, 생성하고자 하는 데이터베이스의 이름을 고유하게 부여한다. 이 때, 지정한 데이터베이스 이름이 이미 존재하는 데이터베이스 이름과 중복되는 경우, CUBRID는 기존 파일을 보호하기 위하여 데이터베이스 생성을 더 이상 진행하지 않는다.
- *locale\_name*: 데이터베이스에서 사용할 로캘 이름을 입력한다. CUBRID에서 사용 가능한 로캘 이름은 [1단계: 로캘 선택](#) 을 참고한다.

• *charset*: 데이터베이스에서 사용할 문자셋을 입력한다. CUBRID에서 사용 가능한 문자셋은 iso88591, euckr, utf8이다.

- *locale\_name* 이 en\_US이고 *charset* 을 생략하면 문자셋은 iso88591이 된다.
- *locale\_name* 이 ko\_KR이고 *charset* 을 생략하면 문자셋은 utf8이 된다.
- 나머지 *locale\_name* 은 *charset* 을 생략할 수 없으며, utf8만 지정 가능하다.

데이터베이스 이름의 최대 길이는 영문 17자이다.

다음은 **cubrid createdb** 에 대한 [options]이다.

|                                              |                                      |
|----------------------------------------------|--------------------------------------|
| <code>--db-volume-size=SIZE</code>           | 처음 생성되는 볼륨(generic)의 크기              |
| <code>--db-page-size=SIZE</code>             | 바이트 단위의 데이터베이스 페이지 크기                |
| <code>--log-volume-size=SIZE</code>          | 로그 볼륨의 크기                            |
| <code>--log-page-size=SIZE</code>            | 바이트 단위의 로그 페이지 크기                    |
| <code>--comment=COMMENT</code>               | 데이터베이스에 관한 설명                        |
| <code>-F, --file-path=PATH</code>            | 데이터베이스가 생성되는 디렉터리 경로                 |
| <code>-L, --log-path=PATH</code>             | 로그 볼륨이 저장되는 디렉터리 경로                  |
| <code>-B, --lob-base-path=PATH</code>        | LOB 파일이 저장되는 디렉터리 경로                 |
| <code>--server-name=HOST</code>              | 서버 호스트 이름                            |
| <code>-r, --replace</code>                   | 같은 이름의 데이터베이스가 존재하면 덮어쓰기             |
| <code>--more-volume-file=FILE</code>         | 추가로 확장될 볼륨에 대한 제어 파일                 |
| <code>--user-definition-file=FILE</code>     | 데이터베이스 사용자를 정의하는 파일                  |
| <code>--csql-initialization-file=FILE</code> | 데이터베이스 생성 후 CSQL에서 FILE에 저장된 질의문을 실행 |
| <code>-o, --output-file=FILE</code>          | 출력 메시지를 파일에 저장                       |
| <code>-v, --verbose</code>                   | 상세한 상태 정보를 화면 출력                     |

### **--db-volume-size=SIZE**

생성될 데이터베이스 볼륨의 크기를 지정한다. 기본값은 시스템 파라미터 **db\_volume\_size** 에 지정된 값이다. 단위는 K, M, G 및 T로 설정할 수 있으며, 각각 킬로바이트(KB), 메가바이트(MB), 기가바이트(GB) 및 테라바이트(TB)를 나타낸다. 단위를 생략하면 바이트 단위가 적용된다. 데이터베이스의 크기는 항상 64개 디스크 섹터의 배수로 올림된다. 이 값은 페이지의 크기에 따라 달라질 수 있으며, 페이지 크기가 각각 4k, 8k 및 16k인 경우 16M, 32M 또는 64M이 될 수 있다.

다음은 첫 번째로 생성되는 testdb의 볼륨 크기를 512MB로 지정하는 구문이다.

```
cubrid createdb --db-volume-size=512M testdb en_US
```

### **--db-page-size=SIZE**

데이터베이스 페이지 크기를 지정하는 옵션으로서, 최소값은 4K, 최대값은 16K(기본값)이다. K는 KB(kilobytes)를 의미한다. 데이터베이스 페이지 크기는 4K, 8K, 16K 중 하나의 값이 된다. 4K



와 16K 사이의 값을 지정할 경우 지정한 값의 올림값으로 설정되며, 4K보다 작으면 4K로 설정되고 16K보다 크면 16K로 설정된다.

다음은 *testdb*를 생성하고, *testdb*의 데이터베이스 페이지 크기를 16K로 지정하는 구문이다.

```
cubrid createdb --db-page-size=16K testdb en_US
```

### **--log-volume-size=SIZE**

데이터베이스 로그 볼륨의 크기를 지정한다. 기본값은 데이터베이스 볼륨 크기와 같으며, 최소값은 20M이다. 단위는 K, M, G 및 T로 설정할 수 있으며, 각각 킬로바이트(KB), 메가바이트(MB), 기가바이트(GB) 및 테라바이트(TB)를 나타낸다. 단위를 생략하면 바이트 단위가 적용된다.

다음은 *testdb*를 생성하고, *testdb*의 로그 볼륨 크기를 256M로 지정하는 구문이다.

```
cubrid createdb --log-volume-size=256M testdb en_US
```

### **--log-page-size=SIZE**

생성되는 데이터베이스의 로그 볼륨 페이지 크기를 지정하는 옵션으로, 기본값은 데이터 페이지 크기와 같다. 최소값은 4K, 최대값은 16K이다. K는 KB(kilobytes)를 의미한다. 데이터베이스 페이지 크기는 4K, 8K, 16K 중 하나의 값이 된다. 4K와 16K 사이의 값을 지정할 경우 지정한 값의 올림값으로 설정되며, 4K보다 작으면 4K로 설정되고 16K보다 크면 16K로 설정된다.

다음은 *testdb*를 생성하고, *testdb*의 로그 볼륨 페이지 크기를 8kbyte로 지정하는 구문이다.

```
cubrid createdb --log-page-size=8K testdb en_US
```

### **--comment=COMMENT**

데이터베이스의 볼륨 헤더에 지정된 주석을 포함하는 옵션으로, 문자열에 공백이 포함되면 큰 따옴표로 감싸주어야 한다.

다음은 *testdb*를 생성하고, 데이터베이스 볼륨에 이에 대한 주석을 추가하는 구문이다.

```
cubrid createdb --comment "a new database for study" testdb en_US
```

### **-F, --file-path=PATH**

새로운 데이터베이스가 생성되는 디렉터리의 절대 경로를 지정하는 옵션으로, **-F** 옵션을 지정하지 않으면 현재 작업 디렉터리에 새로운 데이터베이스가 생성된다.

다음은 *testdb*라는 이름의 데이터베이스를 /dbtemp/new\_db라는 디렉터리에 생성하는 구문이다.

```
cubrid createdb -F "/dbtemp/new_db/" testdb en_US
```

### -L, --log-path=PATH

데이터베이스의 로그 파일이 생성되는 디렉터리의 절대 경로를 지정하는 옵션으로, **-L** 옵션을 지정하지 않으면 **-F** 옵션에서 지정한 디렉터리에 생성된다. **-F** 옵션과 **-L** 옵션을 둘 다 지정하지 않으면 데이터베이스와 로그 파일이 현재 작업 디렉터리에 생성된다.

다음은 *testdb* 라는 이름의 데이터베이스를 */dbtemp/newdb*라는 디렉터리에 생성하고, 로그 파일을 */dbtemp/db\_log* 디렉터리에 생성하는 구문이다.

```
cubrid createdb -F "/dbtemp/new_db/" -L "/dbtemp/db_log/" testdb en_US
```

### -B, --lob-base-path=PATH

**BLOB/CLOB** 데이터를 사용하는 경우 **LOB** 데이터 파일이 저장되는 디렉터리의 경로를 지정하는 옵션으로, 이 옵션을 지정하지 않으면 <데이터베이스 볼륨이 생성되는 디렉터리> **/lob** 디렉터리에 **LOB** 데이터 파일이 저장된다.

다음은 *testdb* 를 현재 작업 디렉터리에 생성하고, **LOB** 데이터 파일이 저장될 디렉터리를 로컬 파일 시스템의 *"/home/data1"*으로 지정하는 구문이다.

```
cubrid createdb --lob-base-path "file:/home/data1" testdb en_US
```

### --server-name=HOST

CUBRID의 클라이언트/서버 버전을 사용할 때 특정 데이터베이스에 대한 서버가 지정한 호스트 상에 구동되도록 하는 옵션이다. 이 옵션으로 지정된 서버 호스트의 정보는 데이터베이스 위치 정보 파일( **databases.txt** )에 기록된다. 이 옵션이 지정되지 않으면 기본값은 현재 로컬 호스트이다.

다음은 *testdb* 를 *aa\_host* 호스트 상에 생성 및 등록하는 구문이다.

```
cubrid createdb --server-name aa_host testdb en_US
```

### -r, --replace

**-r** 은 지정된 데이터베이스 이름이 이미 존재하는 데이터베이스 이름과 중복되더라도 새로운 데이터베이스를 생성하고, 기존의 데이터베이스를 덮어쓰도록 하는 옵션이다.

다음은 *testdb* 라는 이름의 데이터베이스가 이미 존재하더라도 기존의 *testdb* 를 덮어쓰고 새로운 *testdb* 를 생성하는 구문이다.

```
cubrid createdb -r testdb en_US
```

**--more-volume-file=FILE**

데이터베이스가 생성되는 디렉터리에 추가 볼륨을 생성하는 옵션으로 지정된 파일에 저장된 명세에 따라 추가 볼륨을 생성한다. 이 옵션을 이용하지 않더라도, **cubrid addvoldb** 유틸리티를 이용하여 볼륨을 추가할 수 있다.

다음은 *testdb* 를 생성함과 동시에 *vol\_info.txt*에 저장된 명세를 기반으로 볼륨을 추가 생성하는 구문이다.

```
cubrid createdb --more-volume-file vol_info.txt testdb en_US
```

다음은 위 구문으로 *vol\_info.txt*에 저장된 추가 볼륨에 관한 명세이다. 각 볼륨에 관한 명세는 라인 단위로 작성되어야 한다.

```
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
NAME volname COMMENTS volcmnts PURPOSE volpurp NPAGES volnpgs
NAME data_v1 COMMENTS "data information volume" PURPOSE data
NPAGES 1000
NAME data_v2 COMMENTS "data information volume" NPAGES 1000
NAME temp_v1 COMMENTS "temporary information volume" PURPOSE
temp NPAGES 500
#XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXX
```

예제 파일에서와 같이 각 볼륨에 관한 명세는 다음과 같이 구성된다.

```
[NAME volname] [COMMENTS volcmnts] [PURPOSE volpurp] NPAGES
volnpgs
```

- *volname*: 추가 생성될 볼륨의 이름으로 Unix 파일 이름 규약을 따라야 하고, 디렉터리 경로를 포함하지 않는 단순한 이름이어야 한다. 볼륨명에 관한 명세는 생략할 수 있으며, 이 경우 시스템에 의해 “생성될 데이터베이스 이름\_볼륨 식별자”로 볼륨명이 생성된다.
- *volcmnts*: 볼륨 헤더에 기록되는 주석 문장으로, 추가 생성되는 볼륨에 관한 정보를 임의로 부여할 수 있다. 볼륨 주석에 관한 명세 역시 생략할 수 있다.
- *volpurp*: 볼륨이 사용되는 용도를 나타낸다. 영구적 데이터(기본 옵션) 볼륨 또는 일시적 데이터 볼륨 중에 하나를 사용할 수 있다.

 **참고**

이전 버전과의 호환성을 위해 **data**, **index**, **temp** 또는 **generic** 등 기존의 모든 키워드를 사용할 수 있다. **temp** 는 일시적 데이터 볼륨이고, 나머지는 영구적 데이터 볼륨을 나타낸다.

- *volnpgs*: 생성될 추가 볼륨의 페이지 수를 나타낸다. 볼륨의 페이지 수 지정은 생략할 수 없으며, 반드시 지정해야 한다. 실제 볼륨 크기는 **64 sectors** 의 배수로 올림된다.

### --user-definition-file=FILE

생성하고자 하는 데이터베이스에 대해 권한이 있는 사용자를 추가하는 옵션으로, 파라미터로 지정된 사용자 정보 파일에 저장된 명세에 따라 사용자를 추가한다. **--user-definition-file** 옵션을 이용하지 않더라도 **CREATE USER** 구문을 이용하여 사용자를 추가할 수 있다.

다음은 *testdb* 를 생성함과 동시에 *user\_info.txt*에 정의된 사용자 정보를 기반으로 *testdb* 에 대한 사용자를 추가하는 구문이다.

```
cubrid createdb --user-definition-file=user_info.txt testdb en_US
```

사용자 정보 파일의 구문은 아래와 같다.

```
USER user_name [<groups_clause> | <members_clause>]
<groups_clause>:
 [GROUPS <group_name> [{ <group_name> }...]]
<members_clause>:
 [MEMBERS <member_name> [{ <member_name> }...]]
```

- *user\_name*: 데이터베이스에 대해 권한을 가지는 사용자 이름이며, 공백이 포함되지 않아야 한다.
- **GROUPS** 절: 옵션이며, <group\_name>은 지정된 <user\_name>을 포함하는 상위 그룹의 이름이다. 이 때, <group\_name>은 하나 이상이 지정될 수 있으며, **USER** 로 미리 정의되어야 한다.
- **MEMBERS** 절: 옵션이며, <member\_name> 은 지정된 <user\_name>에 포함되는 하위 멤버의 이름이다. 이 때, <member\_name>은 하나 이상이 지정될 수 있으며, **USER** 로 미리 정의되어야 한다.

사용자 정보 파일에서는 주석을 사용할 수 있으며, 주석 라인은 연속된 하이픈(--)으로 시작된다. 공백 라인은 무시된다.

다음 예제는 그룹 *sedan* 에 *grandeur* 와 *sonata* 가, 그룹 *suv* 에 *tuscan* 이, 그룹 *hatchback* 에 *i30* 가 포함되는 것을 정의하는 사용자 정보 파일이다. 사용자 정보 파일명은 *user\_info.txt*로 예시한다.

```
--
-- 사용자 정보 파일의 예 1
--
USER sedan
USER suv
USER hatchback
USER grandeur GROUPS sedan
USER sonata GROUPS sedan
USER tuscan GROUPS suv
USER i30 GROUPS hatchback
```

위 예제와 동일한 사용자 관계를 정의하는 파일이다. 다만, 아래 예제에서는 **MEMBERS** 절을 이용하였다.

```
--
-- 사용자 정보 파일의 예 2
--
USER grandeur
USER sonata
USER tuscan
USER i30
USER sedan MEMBERS sonata grandeur
USER suv MEMBERS tuscan
USER hatchback MEMBERS i30
```

### --csql-initialization-file=FILE

생성하고자 하는 데이터베이스에 대해 CSQL 인터프리터에서 구문을 실행하는 옵션으로, 파라미터로 지정된 파일에 저장된 SQL 구문에 따라 스키마를 생성할 수 있다.

다음은 *testdb* 를 생성함과 동시에 *table\_schema.sql*에 정의된 SQL 구문을 CSQL 인터프리터에서 실행시키는 구문이다.

```
cubrid createdb --csql-initialization-file table_schema.sql
testdb en_US
```

### -o, --output-file=FILE

데이터베이스 생성에 관한 메시지를 파라미터로 지정된 파일에 저장하는 옵션이며, 파일은 데이터베이스와 동일한 디렉터리에 생성된다. **-o** 옵션이 지정되지 않으면 메시지는 콘솔 화면에 출력된다. **-o** 옵션은 데이터베이스가 생성되는 중에 출력되는 메시지를 지정된 파일에 저장함으로써 특정 데이터베이스의 생성 과정에 관한 정보를 활용할 수 있게 한다.

다음은 *testdb* 를 생성하면서 이에 관한 유틸리티의 출력을 콘솔 화면이 아닌 *db\_output* 파일에 저장하는 구문이다.

```
cubrid createdb -o db_output testdb en_US
```

### -v, --verbose

데이터베이스 생성 연산에 관한 모든 정보를 화면에 출력하는 옵션으로서, **-o** 옵션과 마찬가지로 특정 데이터베이스 생성 과정에 관한 정보를 확인하는데 유용하다. 따라서, **-v** 옵션과 **-o** 옵션을 함께 지정하면, **-o** 옵션의 파라미터로 지정된 출력 파일에 **cubrid createdb** 유틸리티의 연산 정보와 생성 과정에 관한 출력 메시지를 저장할 수 있다.

다음은 *testdb* 를 생성하면서 이에 관한 상세한 연산 정보를 화면에 출력하는 구문이다.

```
cubrid createdb -v testdb en_US
```

### 참고

- **temp\_file\_max\_size\_in\_pages** 는 복잡한 질의문이나 정렬 수행에 사용되는 일시적 볼륨을 디스크에 저장하는 데에 할당되는 최대 페이지 개수를 설정하는 파라미터이다. 기본값은 **-1** 로, **temp\_volume\_path** 파라미터가 지정한 디스크의 여유 공간까지 일시적 볼륨이 커질 수 있다. 0이면 일시적 볼륨이 생성되지 않으므로 **cubrid addvoldb** 유틸리티를 이용하여 영구적 임시 볼륨을 충분히 추가해야 한다. 볼륨을 효율적으로 관리하기 위해서는 후자의 방법을 사용할 것을 권장한다.
- **cubrid spacedb** 유틸리티를 사용하면 각 볼륨의 남은 공간을 확인할 수 있다. **cubrid addvoldb** 유틸리티를 사용하면 데이터베이스를 관리하면서 필요에 따라 볼륨을 더 추가할 수 있다. 시스템 부하가 적을 때 볼륨을 추가하는 것이 좋다. 사전 할당된 볼륨이 모두 사용 중이면 데이터베이스 시스템에서는 자동으로 새 볼륨을 생성한다.

다음 예제에서는 영구적 임시 볼륨을 포함한 여러 볼륨을 추가하는 데이터베이스 생성 방법을 보여준다.

```
cubrid createdb --db-volume-size=512M --log-volume-size=256M
cubrid en_US
cubrid addvoldb -S -n cubridb_DATA01 --db-volume-size=512M cubridb
cubrid addvoldb -S -p temp -n cubridb_TEMP01 --db-volume-size=512M
cubridb
```

### 참고

기존 키 파일을 사용하는 데이터베이스 생성

데이터베이스가 생성될 때 기본적으로 키 파일이 함께 생성된다. 만약 데이터베이스 생성 시 기존 키 파일을 사용하고 싶다면, 먼저 키 파일을 <database-name>\_keys 이름으로 복사해 둔다. 이후 복사한 키 파일의 디렉토리를 **tde\_keys\_file\_path** 시스템 파라미터에 지정하고 **createdb** 유틸리티를 통해 데이터베이스를 생성한다. TDE 키 파일에 관한 자세한 내용은 **파일 기반의 마스터 키 관리**를 참고한다.

## addvoldb

CUBRID 볼륨을 세부적으로 관리하려면 addvoldb 를 사용하면 된다. 각 볼륨 파일명, 경로, 용도 및 크기를 세분해서 관리할 수 있다. 데이터베이스 시스템은 모든 볼륨을 직접 관리할 수 있으나, 옵션을 생략하면 각각의 새 볼륨을 구성할 때는 기본값을 사용한다

데이터베이스 볼륨을 수동으로 추가하기 위한 명령은 다음과 같다.

```
cubrid addvoldb [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **addvoldb**: 지정된 데이터베이스에 지정된 페이지 수만큼 새로운 볼륨을 추가하기 위한 명령이다.
- **database\_name**: 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않고, 볼륨을 추가하고자 하는 데이터베이스의 이름을 지정한다.

다음은 데이터베이스를 생성하고 볼륨 용도를 구분하여 데이터(**data**), 인덱스(**index**), 임시(**temp**) 볼륨을 추가하는 예이다.

```
cubrid createdb --db-volume-size=512M --log-volume-size=256M
cubrid en_US
cubrid addvoldb -S -n cubridb_DATA01 --db-volume-size=512M cubridb
cubrid addvoldb -S -p temp -n cubridb_TEMP01 --db-volume-size=512M
cubridb
```

다음은 **cubrid addvoldb** 에 대한 [options]이다.

|                                    |                    |
|------------------------------------|--------------------|
| <b>--db-volume-size=SIZE</b>       | 추가될 볼륨의 크기         |
| <b>-n, --volume-name=NAME</b>      | 볼륨 이름              |
| <b>-F, --file-path=PATH</b>        | 데이터베이스 파일에 대한 볼륨경로 |
| <b>--comment=COMMENT</b>           | 볼륨 주석              |
| <b>-p, --purpose=PURPOSE</b>       | 볼륨 사용 용도           |
| <b>-S, --SA-mode</b>               | 독립 모드 실행           |
| <b>-C, --CS-mode</b>               | 클라이언트 서버 모드 실행     |
| <b>--max-writesize-in-sec=SIZE</b> | 초당 추가될 볼륨의 크기      |

**--db-volume-size=SIZE**

**--db-volume-size** 지정한 데이터베이스에 추가될 볼륨 크기를 지정하는 옵션이다. **--db-volume-size** 옵션을 생략하면 시스템 파라미터 **db\_volume\_size** 의 값이 기본값으로 사용된다. 단위는 K, M, G 및 T로 설정할 수 있으며, 각각 킬로바이트(KB), 메가바이트(MB), 기가바이트(GB) 및 테라바이트(TB)를 나타낸다. 단위를 생략하면 바이트 단위가 적용된다. 데이터베이스의 크기는 항상 64개 디스크 섹터의 배수로 올림된다. 이 값은 페이지의 크기에 따라 달라질 수 있으며, 페이지 크기가 각각 4k, 8k 및 16k인 경우 16M, 32M 또는 64M이 될 수 있다

다음은 *testdb* 에 데이터 볼륨을 추가하며 볼륨 크기를 256MB로 지정하는 구문이다.

```
cubrid addvolddb --db-volume-size=256M testdb
```

**-n, --volume-name=NAME**

지정된 데이터베이스에 대하여 추가될 볼륨의 이름을 지정하는 옵션이다. 볼륨명은 운영체제의 파일 이름 규약을 따라야 하고, 디렉터리 경로나 공백을 포함하지 않는 단순한 이름이어야 한다. **-n** 옵션을 생략하면 추가되는 볼륨의 이름은 시스템에 의해 “데이터베이스 이름\_볼륨 식별자”로 자동 부여된다. 예를 들어, 데이터베이스 이름이 *testdb* 이면 자동 부여된 볼륨명은 *testdb\_x001* 이 된다.

다음은 *testdb*라는 데이터베이스에 볼륨을 추가하는 예이며, 추가되는 볼륨명은 *testdb\_v1* 이 된다.

```
cubrid addvolddb -n testdb_v1 testdb
```

**-F, --file-path=PATH**

지정된 데이터베이스에 대하여 추가될 볼륨이 저장되는 디렉터리 경로를 지정하는 옵션이다. **-F** 옵션을 생략하면, 시스템 파라미터인 **volume\_extension\_path** 의 값이 기본값으로 사용된다.

다음은 *testdb*라는 데이터베이스에 볼륨을 추가하는 구문이며, 추가 볼륨은 */dbtemp/addvol* 디렉터리에 생성된다. 볼륨명에 관한 **-n** 옵션을 지정하지 않았으므로, 볼륨명은 *testdb\_x001* 으로 만들어진다.

```
cubrid addvolddb -F /dbtemp/addvol/ testdb
```

**--comment=COMMENT**

추가된 볼륨에 관한 정보 검색을 쉽게 하기 위하여 볼륨에 관한 정보를 주석으로 처리하는 옵션이다. 이때 주석의 내용은 볼륨을 추가하는 **DBA** 의 이름이나 볼륨 추가의 목적을 포함하는 것이 바람직하며, 큰따옴표로 감싸야 한다.

다음은 볼륨을 추가하고 추가 정보로 주석을 삽입하는 구문이다.



```
cubrid addvolddb --comment "Data volume added by cheolsoo kim
because permanent data space was almost depleted." testdb
```

### **-p, --purpose=PURPOSE**

추가될 볼륨의 용도를 지정한다. 지정된 용도는 추가된 볼륨의 파일 타입들을 정의한다.

- **PERMANENT DATA** : 테이블 행, 인덱스 및 시스템 파일 저장.
- **TEMPORARY DATA** : 질의 처리 및 정렬을 수행할 때 중간 결과와 최종 결과 저장.

이 옵션이 지정되지 않은 경우 해당 볼륨의 용도는 기본적으로 **PERMANENT DATA** 로 간주한다. 다음은 임시 데이터 용도로 볼륨을 추가하는 구문이다.

```
cubrid addvolddb -p temp testdb
```

### **i 참고**

예전 버전에는 PERMANENT DATA 볼륨이 generic, data 및 index로 구분되었으나, 이번 버전부터는 볼륨 구조에 대한 설계가 변경되어 볼륨의 구분이 없어졌다. 그러나 기존에 사용 하던 스크립트의 오류를 방지하기 위해 예전에 사용하였던 keyword(generic, data, index) 는 유지하였고, 기존 버전의 temp는 동일한 용도로 유지하였다.

각 용도에 대한 자세한 내용은 [데이터베이스 볼륨 구조](#) 를 참고한다.

### **-S, --SA-mode**

서버 프로세스를 구동하지 않고 데이터베이스에 접근하는 독립 모드(standalone)로 작업하기 위해 지정되며, 인수는 없다. **-S** 옵션을 지정하지 않으면, 시스템은 클라이언트/서버 모드로 인식한다.

```
cubrid addvolddb -S --db-volume-size=256M testdb
```

### **-C, --CS-mode**

서버 프로세스와 클라이언트 프로세스를 각각 구동하여 데이터베이스에 접근하는 클라이언트/서버 모드로 작업하기 위한 옵션이며, 인수는 없다. **-C** 옵션을 지정하지 않더라도 시스템은 기본적으로 클라이언트/서버 모드로 인식한다.

```
cubrid addvolddb -C --db-volume-size=256M testdb
```

### **--max-writesize-in-sec=SIZE**

데이터베이스에 볼륨을 추가할 때 디스크 출력량을 제한하여 시스템 운영 영향을 줄이도록 하는 옵션이다. 이 옵션을 통해 1초당 쓸 수 있는 최대 크기를 지정할 수 있으며, 단위는 K(kilobytes), M(megabytes)이다. 최소값은 160K이며, 이보다 작게 값을 설정하면 160K로 바뀐다. 단, 클라이언트/서버 모드(-C)에서만 사용 가능하다.

다음은 2GB 볼륨을 초당 1MB씩 쓰도록 하는 예이다. 소요 시간은 35분( = (2048MB / 1MB) / 60 초 ) 정도가 예상된다.

```
cubrid addvoldb -C --db-volume-size=2G --max-writesize-in-sec=1M
testdb
```

## deletedb

**cubrid deletedb** 는 데이터베이스를 삭제하는 유틸리티이다. 데이터베이스가 몇 개의 상호 의존적 파일들로 만들어지기 때문에, 데이터베이스를 제거하기 위해 운영체제 파일 삭제 명령이 아닌 **cubrid deletedb** 유틸리티를 사용해야 한다.

**cubrid deletedb** 유틸리티는 데이터베이스 위치 파일( **databases.txt** )에 지정된 데이터베이스에 대한 정보도 같이 삭제한다. **cubrid deletedb** 유틸리티는 오프라인 상에서 즉, 아무도 데이터베이스를 사용하지 않는 상태에서 독립 모드로 사용해야 한다.

```
cubrid deletedb [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **deletedb**: 데이터베이스 및 관련 데이터, 로그, 백업 파일을 전부 삭제하기 위한 명령으로, 데이터베이스 서버가 구동 정지 상태인 경우에만 정상적으로 수행된다.
- *database\_name*: 디렉터리 경로명을 포함하지 않고, 삭제하고자 하는 데이터베이스의 이름을 지정한다

다음은 **cubrid deletedb** 에 대한 [options]이다.

|                               |                |
|-------------------------------|----------------|
| <b>-o, --output-file=FILE</b> | 출력 메시지가 저장될 파일 |
| <b>-d, --delete-backup</b>    | 백업 볼륨을 지움      |

### **-o, --output-file=FILE**

데이터베이스를 삭제하면서 출력되는 메시지를 인자로 지정한 파일에 기록하는 명령이다. **cubrid deletedb** 유틸리티를 사용하면 데이터베이스 위치 정보 파일( **databases.txt** )에 기록된 데이터베이스 정보가 함께 삭제된다.

```
cubrid deletedb -o deleted_db.out testdb
```

만약, 존재하지 않는 데이터베이스를 삭제하는 명령을 입력하면 다음과 같은 메시지가 출력된다.

```
cubrid deletedb testdb
Database "testdb" is unknown, or the file "databases.txt" cannot
be accessed.
```

### -d, --delete-backup

데이터베이스를 삭제하면서 백업 볼륨 및 백업 정보 파일도 함께 삭제할 수 있다. -d 옵션을 지정하지 않으면 백업 볼륨 및 백업 정보 파일은 삭제되지 않는다.

```
cubrid deletedb -d testdb
```

## renamedb

**cubrid renamedb** 유틸리티는 존재하는 데이터베이스의 현재 이름을 변경한다. 정보 볼륨, 로그 볼륨, 제어 파일들이 새로운 이름과 일치되게 이름을 변경한다.

이에 비해 **cubrid alterdbhost** 유틸리티는 지정된 데이터베이스의 호스트 이름을 설정하거나 변경한다. 즉, **databases.txt**에 있는 호스트 이름을 변경한다.

```
cubrid renamedb [options] src_database_name dest_database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **renamedb**: 현재 존재하는 데이터베이스의 이름을 새로운 이름으로 변경하기 위한 명령으로, 데이터베이스가 구동 정지 상태인 경우에만 정상적으로 수행된다. 관련된 정보 볼륨, 로그 볼륨, 제어 파일도 함께 새로 지정된 이름으로 변경된다.
- *src\_database\_name*: 이름을 바꾸고자 하는 현재 존재하는 데이터베이스의 이름이며, 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않는다.
- *dest\_database\_name*: 새로 부여하고자 하는 데이터베이스의 이름이며, 현재 존재하는 데이터베이스 이름과 중복되어서는 안 된다. 이 역시, 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않는다.

다음은 **cubrid renamedb**에 대한 [options]이다.

```
-E, --extended-volume-path=PATH 확장 볼륨이 저장되는 디렉터리 경로
-i, --control-file=FILE 여러 개의 볼륨들이 각각 저장되는 디렉터리 경
로를 지정하는 제어 파일
-d, --delete-backup 백업 볼륨을 지움
```

**-E, --extended-volume-path=PATH**

확장 볼륨의 이름을 변경한 후 새 디렉터리 경로로 이동하는 명령으로서, **-E** 옵션을 이용하여 변경된 이름을 가지는 확장 볼륨을 이동시킬 새로운 디렉터리 경로(예: `/dbtemp/newaddvols/`)를 지정한다.

**-E** 옵션을 주지 않으면, 확장 볼륨은 기존 위치에서 이름만 변경된다. 이때, 기존 데이터베이스 볼륨의 디스크 파티션 외부에 있는 디렉터리 경로 또는 유효하지 않은 디렉터리 경로가 지정되는 경우 데이터베이스 이름 변경 작업은 수행되지 않으며, **-i** 옵션과 병행될 수 없다.

```
cubrid renamedb -E /dbtemp/newaddvols/ testdb testdb_1
```

**-i, --control-file FILE**

각 볼륨 또는 파일에 대하여 일괄적으로 데이터베이스 이름을 변경하면서 디렉터리 경로를 상이하게 지정하기 위해 디렉터리 정보가 저장된 입력 파일을 지정하는 명령으로서, **-i** 옵션을 이용한다. 이때, **-i** 옵션은 **-E** 옵션과 병행될 수 없다.

```
cubrid renamedb -i rename_path testdb testdb_1
```

다음은 개별적 볼륨들의 이름과 현재 디렉터리 경로, 그리고 변경된 이름의 볼륨들이 저장될 디렉터리 경로를 포함하는 파일의 구문 및 예시이다.

```
valid source_fullvolname dest_fullvolname
```

- *valid*: 각 볼륨을 식별하기 위한 정수이며, 데이터베이스 볼륨 정보 제어 파일 (`database_name_vinf`)를 통해 확인할 수 있다.
- *source\_fullvolname*: 각 볼륨에 대한 현재 디렉터리 경로이다.
- *dest\_fullvolname*: 이름이 변경된 새로운 볼륨이 이동될 목적지 디렉터리 경로이다. 만약, 목적지 디렉터리가 유효하지 않은 경우 데이터베이스 이름 변경 작업은 수행되지 않는다.

```
-5 /home1/user/testdb_vinf /home1/CUBRID/databases/
testdb_1_vinf
-4 /home1/user/testdb_lginf /home1/CUBRID/databases/
testdb_1_lginf
-3 /home1/user/testdb_bkvinf /home1/CUBRID/databases/
testdb_1_bkvinf
-2 /home1/user/testdb_lgat /home1/CUBRID/databases/
testdb_1_lgat
0 /home1/user/testdb /home1/CUBRID/databases/testdb_1
1 /home1/user/backup/testdb_x001 /home1/CUBRID/databases/
backup/testdb_1_x001
```

**-d, --delete-backup**

데이터베이스의 이름을 변경하면서 데이터베이스와 동일 위치에 있는 모든 백업 볼륨 및 백업 정보 파일을 함께 강제 삭제하는 명령이다. 일단, 데이터베이스 이름이 변경되면 이보다 앞선 이름의 백업 파일은 이용할 수 없으므로 주의해야 한다. 만약, **-d** 옵션을 지정하지 않으면 백업 볼륨 및 백업 정보 파일은 삭제되지 않는다.

```
cubrid renamedb -d testdb testdb_1
```

## alterdbhost

**cubrid alterdbhost** 유틸리티는 지정된 데이터베이스의 호스트 이름을 설정하거나 변경한다. 즉, **databases.txt** 에 있는 호스트 이름을 변경한다.

```
cubrid alterdbhost [<option>] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **alterdbhost**: 현 데이터베이스의 호스트 이름을 새로운 이름으로 변경하기 위한 명령이다.

**cubrid alterdbhost** 에서 사용하는 옵션은 다음과 같다.

### -h, --host=HOST

뒤에 변경할 호스트 이름을 지정하며, 옵션을 생략하면 호스트 이름으로 localhost를 지정한다.

## copydb

**cubrid copydb** 유틸리티는 데이터베이스를 한 위치에서 다른 곳으로 복사 또는 이동하며, 인자로 원본 데이터베이스 이름과 새로운 데이터베이스 이름이 지정되어야 한다. 이때, 새로운 데이터베이스 이름은 원본 데이터베이스 이름과 다른 이름으로 지정되어야 하고, 새로운 데이터베이스에 대한 위치 정보는 **databases.txt** 에 등록된다.

**cubrid copydb** 유틸리티는 원본 데이터베이스가 정지 상태일 때(오프라인)에만 실행할 수 있다.

```
cubrid copydb [<options>] src-database-name dest-database-name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **copydb**: 원본 데이터베이스를 새로운 위치로 이동 또는 복사하는 명령이다.
- *src-database-name*: 복사 또는 이동하고자 하는 원본 데이터베이스 이름이다.
- *dest-database-name*: 새로운 데이터베이스 이름이다.

[options]를 생략하면 원본 데이터베이스를 현재 작업 디렉터리에 복사한다.

**cubrid copydb** 에 대한 [options]는 다음과 같다.

|                                              |                                        |
|----------------------------------------------|----------------------------------------|
| <code>--server-name=HOST</code>              | 서버 호스트 이름                              |
| <code>-F, --file-path=PATH</code>            | 데이터베이스가 저장되는 디렉터리 경로                   |
| <code>-L, --log-path=PATH</code>             | 로그 볼륨이 저장되는 디렉터리 경로                    |
| <code>-E, --extended-volume-path=PATH</code> | 확장 볼륨이 저장되는 디렉터리 경로                    |
| <code>-i, --control-file=FILE</code>         | 여러 개의 볼륨들이 각각 저장되는 디렉터리 경로를 지정하는 제어 파일 |
| <code>-r, --replace</code>                   | 같은 이름의 데이터베이스가 존재하면 덮어쓰기               |
| <code>-d, --delete-source</code>             | 복사 후 원본 데이터베이스 삭제                      |
| <code>--copy-lob-path</code>                 | 원본 데이터베이스의 LOB 디렉터리 경로를 복사             |
| <code>-B</code> 옵션과 함께 사용 불가. 기본값: 복사하지 않음   |                                        |
| <code>-B, --lob-base-path=PATH</code>        | LOB 파일이 저장되는 디렉터리 경로                   |

#### **--server-name=HOST**

새로운 데이터베이스의 서버 호스트 이름을 명시하며, 이는 **databases.txt** 의 **db-host** 항목에 등록된다. 이 옵션을 생략하면, 로컬 호스트가 등록된다.

```
cubrid copydb --server-name=cub_server1 demodb new_demodb
```

#### **-F, --file-path=PATH**

새로운 데이터베이스 볼륨이 저장되는 특정 디렉터리 경로를 지정할 수 있다. 절대 경로로 지정해야 하며, 존재하지 않는 디렉터리를 지정하면 에러를 출력한다. 이 옵션을 생략하면 현재 작업 디렉터리에 새로운 데이터베이스의 볼륨이 생성된다. 이 경로는 **databases.txt** 의 **vol-path** 항목에 등록된다.

```
cubrid copydb -F /home/usr/CUBRID/databases demodb new_demodb
```

#### **-L, --log-path=PATH**

새로운 데이터베이스 로그 볼륨이 저장되는 특정 디렉터리 경로를 지정할 수 있다. 절대 경로로 지정해야 하며, 존재하지 않는 디렉터리를 지정하면 에러를 출력한다. 이 옵션을 생략하면 새로운 데이터베이스 볼륨이 저장되는 경로에 로그 볼륨도 함께 생성된다. 이 경로는 **databases.txt** 의 **log-path** 항목에 등록된다.

```
cubrid copydb -L /home/usr/CUBRID/databases/logs demodb new_demodb
```

#### **-E, --extended-volume-path=PATH**

새로운 데이터베이스의 확장 정보 볼륨이 저장되는 특정 디렉터리 경로를 지정할 수 있다. 이 옵션을 생략하면 새로운 데이터베이스 볼륨이 저장되는 경로 또는 제어 파일에 등록된 경로에 확장 정보 볼륨이 저장된다. **-i** 옵션과 병행될 수 없다.

```
cubrid copydb -E /home/usr/CUBRID/databases/extvols demodb
new_demodb
```

### **-i, --control-file=FILE**

대상 데이터베이스에 대한 복수 개의 볼륨들을 각각 다른 디렉터리에 복사 또는 이동하기 위해서, 원본 볼륨의 경로 및 새로운 디렉터리 경로 정보를 포함하는 입력 파일을 지정할 수 있다. 이때, **-i** 옵션은 **-E** 옵션과 병행될 수 없다. 아래 예제에서는 `copy_path`라는 입력 파일을 예로 사용했다.

```
cubrid copydb -i copy_path demodb new_demodb
```

다음은 각 볼륨들의 이름과 현재 디렉터리 경로, 그리고 새로 복사할 디렉터리 및 새로운 볼륨 이름을 포함하는 입력 파일의 예시이다.

```
valid source_fullvolname dest_fullvolname
0 /usr/databases/demodb /drive1/usr/databases/new_demodb
1 /usr/databases/demodb_data1 /drive1/usr/databases/
new_demodb_data1
2 /usr/databases/ext/demodb_ext1 /drive2//usr/databases/
new_demodb_ext1
3 /usr/databases/ext/demodb_ext2 /drive2/usr/databases/
new_demodb_ext2
```

- **valid** : 각 볼륨을 식별하기 위한 정수이며, 데이터베이스 볼륨 정보 제어 파일(**database\_name\_vinf**)를 통해 확인할 수 있다.
- **source\_fullvolname** : 원본 데이터베이스의 각 볼륨이 존재하는 현재 디렉터리 경로이다.
- **dest\_fullvolname** : 새로운 데이터베이스의 각 볼륨이 저장될 디렉터리 경로이며, 유효한 디렉터리를 지정해야 한다.

### **-r, --replace**

새로운 데이터베이스 이름이 기존 데이터베이스 이름과 중복되더라도 에러를 출력하지 않고 덮어쓴다.

```
cubrid copydb -r -F /home/usr/CUBRID/databases demodb new_demodb
```

### **-d, --delete-source**

새로운 데이터베이스로 복사한 후, 원본 데이터베이스를 제거한다. 이 옵션이 주어진다면 데이터베이스 복사 후 **cubrid deletedb** 를 수행하는 것과 동일하다. 단, 원본 데이터베이스에 **LOB** 데이터를 포함하는 경우, 원본 데이터베이스의 **LOB** 파일 디렉터리 경로가 새로운 데이터베이스로 복사되어 **databases.txt** 의 **-lob-base-path** 항목에 등록된다.

```
cubrid copydb -d -F /home/usr/CUBRID/databases demodb new_demodb
```

### --copy-lob-path

이 옵션은 대상 데이터베이스의 LOB 디렉토리의 위치를 원본 데이터베이스의 LOB 디렉토리의 위치와 동일하게 설정한다. 이 옵션을 생략하면 기본 위치는 <대상 데이터베이스 디렉토리>/lob 이다. 이 옵션은 LOB 파일들을 복사하는 기능은 아니며, -B 옵션과 병행할 수 없다.

```
cubrid copydb --copy-lob-path demodb new_demodb
```

### -B, --lob-base-path=PATH

-B 옵션을 사용하여 특정 디렉토리를 새로운 데이터베이스에 대한 **LOB** 파일 디렉터리 경로를 지정한다. 원본 데이터베이스에 있던 **LOB** 파일들은 사용자가 직접 이 디렉터리에 복사해야 한다. 이 옵션은 --copy-lob-path 옵션과 병행할 수 없다.

```
cubrid copydb -B /home/usr/CUBRID/databases/new_lob demodb
new_demodb
```

## installdb

**cubrid installdb** 유틸리티는 데이터베이스 위치 정보를 저장하는 **databases.txt** 에 새로 설치된 데이터베이스 정보를 등록한다. 이 유틸리티의 실행은 등록 대상 데이터베이스의 동작에 영향을 끼치지 않는다.

```
cubrid installdb [<options>] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **installdb**: 이동 또는 복사된 데이터베이스의 정보를 **databases.txt** 에 등록하는 명령이다.
- *database\_name*: **databases.txt** 에 등록하고자 하는 데이터베이스의 이름이다.

[options]를 생략하는 경우, 해당 데이터베이스가 존재하는 디렉터리에서 명령을 수행해야 한다.

**cubrid installdb** 에 대한 [options]는 다음과 같다.

|                      |                       |
|----------------------|-----------------------|
| --server-name=HOST   | 서버 호스트 이름             |
| -F, --file-path=PATH | 데이터베이스 볼륨을 위한 디렉터리 경로 |
| -L, --log-path=PATH  | 로그 볼륨을 위한 디렉터리 경로     |

### --server-name=HOST



대상 데이터베이스의 서버 호스트 정보를 지정된 호스트 명으로 **databases.txt**에 등록한다. 이 옵션을 생략하면, 현재의 호스트 정보가 등록된다.

```
cubrid installldb --server-name=cub_server1 testdb
```

#### **--file-path=PATH**

대상 데이터베이스의 데이터베이스 볼륨을 위한 디렉터리 경로를 **databases.txt**에 등록한다. 이 옵션을 생략하면 현재 작업 디렉터리 경로가 등록된다.

```
cubrid installldb -F /home/cubrid/CUBRID/databases/testdb testdb
```

#### **-L, --log-path=PATH**

대상 데이터베이스 로그 볼륨 디렉터리의 절대 경로를 **databases.txt**에 등록한다. 이 옵션을 생략하면 데이터베이스 볼륨의 디렉터리 경로가 등록된다.

```
cubrid installldb -L /home/cubrid/CUBRID/databases/logs/testdb testdb
```

## backupdb

데이터베이스 백업은 CUBRID 데이터베이스 볼륨, 제어 파일, 로그 파일을 저장하는 작업으로 **cubrid backupdb** 유틸리티 또는 CUBRID 매니저를 이용하여 수행된다. DBA는 저장 매체의 오류 또는 파일 오류 등의 장애에 대비하여 데이터베이스를 정상적으로 복구할 수 있도록 주기적으로 데이터베이스를 백업해야 한다. 이 때 복구 환경은 백업 환경과 동일한 운영체제 및 동일한 버전의 CUBRID가 설치된 환경이어야 한다. 이러한 이유로 데이터베이스를 새로운 버전으로 마이그레이션한 후에는 즉시 새로운 버전의 환경에서 백업을 수행해야 한다. 다만, CUBRID에서 사용하는 LOB(Large Object) 데이터는 데이터베이스 볼륨과 별도의 파일로 관리되므로, backupdb 유틸리티에 의해 자동으로 백업되지 않는다. 따라서 DBA는 데이터베이스 백업 시 LOB 파일이 저장된 디렉터리를 별도로 백업하여야 하며, 복구 시에도 데이터베이스 복구와 함께 해당 LOB 파일을 원래 위치에 복원해야 한다.

**cubrid backupdb** 유틸리티는 모든 데이터베이스 페이지들, 제어 파일들, 데이터베이스를 백업 시와 일치된 상태로 복구하기 위해 필요한 로그 레코드들을 복사한다.

```
cubrid backupdb [options] database_name[@hostname]
```

- @hostname: 독립(Standalone) 모드 DB를 백업하는 경우 생략되며, HA 환경에서 백업하는 경우 데이터베이스 이름 뒤에 "@ hostname"을 지정한다. hostname은 \$CUBRID\_DATABASES/databases.txt에 지정된 이름이다. 로컬 서버를 설정하려면 "@localhost"를 지정하면 된다.

다음은 **cubrid backupdb** 유틸리티와 결합할 수 있는 옵션이다. 대소문자가 구분됨을 주의한다.

```

-D, --destination-path=<백업> 볼륨이 저장될 디렉터리 경로
-r, --remove-archive 불필요한 로그 파일을 지움
-l, --level=LEVEL 백업 수준
-o, --output-file=<파일> 자세한 백업 메시지를 파일에 출력합니다.
-S, --SA-mode 독립 모드 실행
-C, --CS-mode 클라이언트 서버 모드 실행
 --no-check 데이터베이스의 정합성 확인 작업을 수행하지 않습
니다
-t, --thread-count=COUNT 백업을 수행하는 스레드 개수
 --no-compress 백업 볼륨을 압축하지 않습니다
 --sleep-msecs=N 매 1M 읽기 마다 N 밀리초를 쉬니다
-k, --separate-keys TDE에 사용되는 Key File (_keys)을 백업볼륨
에 포함하지 않고 별개의 파일로 분리합니다.

```

### **-D, --destination-path=PATH**

지정된 디렉터리에 백업 파일이 저장되도록 하며, 현재 존재하는 디렉터리가 지정되어야 한다. 그렇지 않으면 지정한 이름의 백업 파일이 생성된다. **-D** 옵션이 지정되지 않으면 백업 파일은 해당 데이터베이스의 위치 정보를 저장하는 파일인 **databases.txt**에 명시된 로그 디렉터리에 생성된다.

```
cubrid backupdb -D /home/cubrid/backup demodb
```

**-D** 옵션을 이용하여 현재 디렉터리에 백업 파일이 저장되도록 한다. **-D** 옵션의 인수로 **"."**을 입력하면 현재 디렉터리가 지정된다.

```
cubrid backupdb -D . demodb
```

### **-r, --remove-archive**

활성 로그(active log)가 꼭 차면 활성 로그를 새로운 보관 로그 파일에 기록한다. 이때 백업을 수행하여 백업 볼륨이 생성되면, 백업 시점보다 앞의 보관 로그는 추후 복구 작업에 불필요하다. **-r** 옵션은 백업을 수행한 후에, 추후 복구 작업에 더 이상 사용되지 않을 보관 로그 파일을 제거하는 옵션이다. **-r** 옵션은 백업 시점 이전의 불필요한 보관 로그만 제거하므로 복구 작업에는 영향을 끼치지 않지만, 관리자가 백업 시점 이후의 보관 로그까지 제거하는 경우 전체 복구가 불가능할 수도 있다. 따라서 보관 로그를 제거할 때에는 추후 복구 작업에 필요한 것인지 반드시 검토해야 한다.

**-r** 옵션을 사용하여 증분 백업(백업 수준 1 또는 2)을 수행하는 경우, 추후 데이터베이스의 정상 복구가 불가능할 수도 있으므로 **-r** 옵션은 전체 백업 수행 시에만 사용하는 것을 권장한다.

```
cubrid backupdb -r demodb
```

### **-l, --level=LEVEL**

지정된 백업 수준으로 증분 백업을 수행한다. **-l** 옵션이 지정되지 않으면 전체 백업이 수행된다. 백업 수준에 대한 자세한 내용은 **증분 백업** 을 참조한다.

```
cubrid backupdb -l 1 demodb
```

### **-o, --output-file=FILE**

대상 데이터베이스의 백업에 관한 진행 정보를 info\_backup이라는 파일에 기록한다.

```
cubrid backupdb -o info_backup demodb
```

다음은 info\_backup 파일 내용의 예시로서, 스레드 개수, 압축 방법, 백업 시작 시간, 영구 볼륨의 개수, 백업 진행 정보, 백업 완료 시간 등의 정보를 확인할 수 있다.

```
[Database(demodb) Full Backup start]
- num-threads: 1
- compression method: NONE
- backup start time: Mon Jul 21 16:51:51 2008
- number of permanent volumes: 1
- backup progress status

volume name | # of pages | backup progress
status | done

demodb_keys | 1 |
| done
demodb_vinf | 1 |
| done
demodb | 25000 |
| done
demodb_lginf | 1 |
| done
demodb_lgat | 25000 |
| done

backup end time: Mon Jul 21 16:51:53 2008
[Database(demodb) Full Backup end]
```

### **-S, --SA-mode**

독립 모드, 즉 오프라인으로 백업을 수행한다. **-S** 옵션이 생략되면 클라이언트/서버 모드에서 백업이 수행된다.

```
cubrid backupdb -S demodb
```

**-C, --CS-mode**

클라이언트/서버 모드에서 백업을 수행하며, demodb를 온라인 백업한다. **-C** 옵션이 생략되면 클라이언트/서버 모드에서 백업이 수행된다.

```
cubrid backupdb -C demodb
```

**--no-check**

대상 데이터베이스의 일관성을 체크하지 않고 백업을 수행한다.

```
cubrid backupdb --no-check demodb
```

**-t, --thread-count=COUNT**

관리자가 임의로 스레드의 개수를 지정함으로써 병렬 백업을 수행한다. **-t** 옵션의 인수를 지정하지 않더라도 시스템의 CPU 개수만큼 스레드를 자동 부여하여 병렬 백업을 수행한다.

```
cubrid backupdb -t 4 demodb
```

**--no-compress**

기본적으로 큐브리드는 백업 볼륨의 크기를 줄이기 위해서, 대상 데이터베이스를 압축한 뒤 저장한다. 백업 파일 압축 실행 시 발생하는 CPU 사용량을 없애고 싶다면, **--no-compress** 옵션을 통해 백업 볼륨을 압축하지 않을 수 있다.

```
cubrid backupdb --no-compress demodb
```

**--sleep-msecs=NUMBER**

대상 데이터베이스를 백업하는 도중 쉬는 시간을 설정한다. 단위는 밀리초이며, 기본값은 0 이다. 1MB의 파일을 읽을 때마다 설정한 시간만큼 쉰다. 백업 작업이 과도한 디스크 I/O를 유발하기 때문에, 운영 중인 서비스에 백업 작업으로 인한 영향을 줄이고자 할 때 이 옵션이 사용된다.

```
cubrid backupdb --sleep-msecs=5 demodb
```

**-k, --separate-keys**

백업 볼륨에 키 파일을 포함하지 않는다. 포함되지 않은 키는 `<database_name>_bk<backup_level>_keys` 라는 이름을 가진 별도의 파일로 분리된다. 옵션을 주지 않을 경우 기본적으로 키 파일은 백업 볼륨에 포함된다. 키 파일 분리에 대한 자세한 설명은 **백업 볼륨 암호화** 을 참고한다.

```
cubrid backupdb -k demodb
```

### 참고

키 파일은 DB 내부에 저장되어 있는 TDE의 master key를 암호/복호화하기 위한 보조 키를 관리하는 파일로 TDE 기능을 사용할 경우 백업시 배업 볼륨에 함께 백업하거나 동일한 저장 매체에 보관하지 말고, 별도의 안전한 방법으로 분리하여 관리를 권장한다. 키 파일이 유실되거나 노출될 경우 TDE 기능 사용한 데이터를 사용할 수 없거나 보안 사고로 이어질 수 있으므로 보관 정책을 명확히 수립하는 것을 권장한다.

## 백업 정책 및 방식

백업을 진행할 때 고려해야 할 사항은 다음과 같다.

### · 백업할 대상 데이터 선별

- 보존 가치가 있는 유효한 데이터인지 판단한다.
- 데이터베이스 전체를 백업할 것인지, 일부만 백업할 것인지 결정한다.
- 데이터베이스와 함께 백업해야 할 다른 파일이 있는지 확인한다.

### · 백업 방식 결정

- 증분 백업, 온라인 백업 방식을 결정한다. 부가적으로 압축 백업, 병렬 백업 모드 사용 여부를 결정한다.
- 사용 가능한 백업 도구 및 백업 장비를 준비한다.

### · 백업 시기 판단

- 데이터베이스 사용이 가장 적은 시간을 파악한다.
- 보관 로그의 양을 파악한다.
- 백업할 데이터베이스를 이용하는 클라이언트 수를 파악한다.

## 온라인 백업

온라인 백업(또는 핫 백업)은 운영 중인 데이터베이스에 대해 백업을 수행하는 방식으로, 특정 시점의 데이터베이스 이미지의 스냅샷을 제공한다. 운영 중인 데이터베이스를 대상으로 백업을 수행하기 때문에 커밋되지 않은 데이터가 저장될 우려가 있고, 다른 데이터베이스 운영에도 영향을 줄 수 있다.

온라인 백업을 하려면 **cubrid backupdb -C** 명령어를 사용한다.

## 오프라인 백업

오프라인 백업(또는 콜드 백업)은 정지 상태인 데이터베이스에 대해 백업을 수행하는 방식으로 특정 시점의 데이터베이스 이미지의 스냅샷을 제공한다.

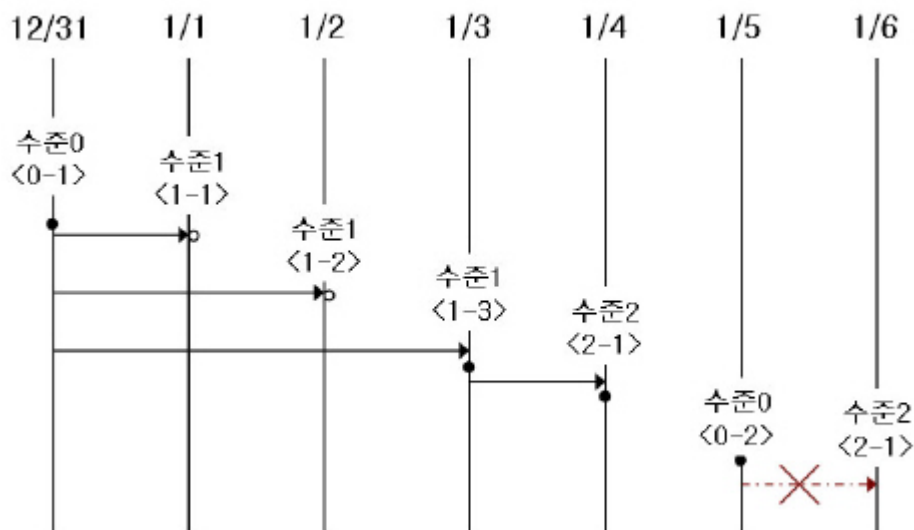
오프라인 백업을 하려면 **cubrid backupdb -S** 명령어를 사용한다.

## 증분 백업

증분 백업(incremental backup)은 전체 백업에 종속적으로 수행되는 백업으로 먼저 수행된 백업 이후의 변경된 사항만을 선택적으로 백업하는 방식이다. 이는 전체 백업보다 백업 볼륨이 적고, 백업 소요 시간이 짧다는 장점이 있다. CUBRID는 0, 1, 2의 백업 수준을 제공하며, 낮은 백업 수준으로 백업을 수행한 이후에만 순차적으로 다음 수준의 백업을 수행할 수 있다.

증분 백업을 하려면 **cubrid backupdb -l LEVEL** 명령어를 사용한다.

다음은 증분 백업에 관한 예시로서, 이를 참조하여 백업 수준에 관해 상세하게 살펴보기로 한다.



- **전체 백업(백업 수준 0):** 백업 수준 0은 모든 데이터베이스 페이지를 포함하는 전체 백업이다.

데이터베이스에 최초 시도되는 백업 수준은 당연히 수준 0이 된다. **DBA**는 복구 상황을 대비하여 정기적으로 전체 백업을 수행해야 하며, 예시에서는 12월 31일과 1월 5일에 전체 백업을 수행하였다.

- **1차 증분 백업(백업 수준 1):** 백업 수준 1은 수준 0의 전체 백업 이후의 변경 사항만 저장하는 증분 백업으로서, 이를 “1차 증분 백업”이라 한다.

주의할 점은 예시의 <1-1>, <1-2>, <1-3>과 같이 1차 증분 백업이 연속적으로 시도되더라도 언제나 수준 0의 전체 백업을 기본으로 증분 백업을 수행한다는 점이다.

만약, 동일 디렉터리에서 백업 파일이 생성된다고 할 때, 1월 1일에 이미 1차 증분 백업 <1-1>이 수행되고, 1월 2일에 또다시 1차 증분 백업 <1-2>가 시도되면, <1-1>에서 생성된 증분 백업 파일을 덮어쓰게 된다. 1월 3일에 1차 증분 백업이 다시 수행되었으므로, 최종 증분 파일은 이 때 생성된다.

그러나, 1월 1일이나 1월 2일의 상태로 데이터베이스를 복구해야 하는 상황이 발생할 수 있으므로, **DBA**는 최종 증분 파일로 덮어쓰기 전에 <1-1>과 <1-2> 각각의 증분 백업 파일을 저장 매체에 별도로 보관하는 것이 좋다.

- **2차 증분 백업(백업 수준 2):** 백업 수준 2는 1차 증분 백업 이후의 변경 사항만 저장하는 증분 백업으로 이를 “2차 증분 백업”이라 한다.

1차 증분 백업이 선행되어야만 2차 증분 백업을 수행할 수 있으므로, 1월 4일에 시도한 2차 증분 백업 시도는 성공할 것이고, 1월 6일에 시도한 2차 증분 백업 시도는 당연히 허용되지 않을 것이다.

이러한 백업 수준 0, 1, 2로 생성된 백업 파일들은 모두 데이터베이스를 복구할 때 필요하므로, 2차 증분 백업이 완료된 1월 4일의 상태로 데이터베이스를 복구하기 위해서는 <2-1>에서 생성된 2차 증분 백업 파일, <1-3>에서 생성된 1차 증분 백업 파일, <0-1>에서 생성된 전체 백업 파일이 모두 필요하다. 즉, 완전한 복구를 위해서는 직전에 생성된 증분 백업 파일보다 앞서서 최종으로 생성된 전체 백업 파일이 요구된다.

### 압축 백업 모드

압축 백업(compress backup)은 데이터베이스를 압축하여 백업을 수행하기 때문에 백업 볼륨의 크기가 줄어들어 디스크 I/O 비용을 감소시킬 수 있고, 디스크 공간을 절약할 수 있다.

큐브리드 11.2버전부터는 압축 백업 모드가 기본값으로 설정되어 있다.

### 병렬 백업 모드

병렬 백업 또는 다중 백업(multi-thread backup)은 지정된 스레드 개수만큼 동시 백업을 수행하기 때문에 백업 시간을 크게 단축시켜 준다. 기본적으로 시스템의 CPU 수만큼 스레드를 부여하게 된다.

병렬 백업을 하려면 **cubrid backupdb -t | --thread-count** 명령어를 사용한다.

### 백업 파일 관리

백업 대상 데이터베이스의 크기에 따라 하나 이상의 백업 파일이 연속적으로 생성될 수 있으며, 각각의 백업 파일의 확장자에는 생성 순서에 따라 000, 001~0xx와 같은 유닛 번호가 순차적으로 부여된다.

### 백업 작업 중 디스크 용량 관리

백업 작업 도중, 백업 파일이 저장되는 디스크 용량에 여유가 없는 경우 백업 작업을 진행할 수 없다는 안내 메시지가 화면에 나타난다. 안내 메시지에는 백업 대상이 되는 데이터베이스의 이름과 경로명, 백업 파일명, 백업 파일의 유닛 번호, 백업 수준이 표시된다. 백업 작업을 계속 진행하려는 관리자는 다음과 같이 옵션을 선택할 수 있다.

- 옵션 0: 백업 작업을 더 이상 진행하지 않을 경우, 0을 입력한다.
- 옵션 1: 백업 작업을 진행하기 위해 관리자는 현재 장치에 새로운 디스크를 삽입한 후 1을 입력한다.

- 옵션 2: 백업 작업을 진행하기 위해 관리자는 장치를 변경하거나 백업 파일이 저장되는 디렉터리 경로를 변경한 후 2를 입력한다.

```

Backup destination is full, a new destination is required to
continue:
Database Name: /local1/testing/demodb
Volume Name: /dev/rst1
Unit Num: 1
Backup Level: 0 (FULL LEVEL)
Enter one of the following options:
Type
- 0 to quit.
- 1 to continue after the volume is mounted/loaded. (retry)
- 2 to continue after changing the volume's directory or device.

```

## 보관 로그 관리

운영체제의 파일 삭제 명령(rm, del)을 사용하여 보관 로그(archive log)를 임의로 삭제해서는 안 되며, 시스템의 설정, **cubrid backupdb** 유틸리티 또는 서버 프로세스에 의해 보관 로그가 삭제되어야 한다. 보관 로그가 삭제될 수 있는 경우는 다음의 3가지이다.

- HA 환경이 아닌 경우(ha\_mode=off)

**force\_remove\_log\_archives** 를 yes(기본값)로 설정하면, 최대 **log\_max\_archives** 개수만큼만 보관 로그가 유지되고 나머지는 자동으로 삭제된다. 단, 가장 오래된 보관 로그 파일에 액티브한 트랜잭션이 있다면 이 트랜잭션이 종료될 때까지 해당 로그 파일이 삭제되지 않는다.

- HA 환경인 경우(ha\_mode=on)

**force\_remove\_log\_archives**를 no로 설정하고, **log\_max\_archives** 개수를 지정하면 복제 반영 후 자동으로 삭제된다.

### 참고

ha\_mode=on일 때 **force\_remove\_log\_archives**를 yes로 설정하면 복제 반영이 안 된 보관 로그가 삭제될 수 있으므로, 이를 권장하지는 않는다. 다만, 복제 재구축을 감수하더라도 마스터 노드의 디스크 여유 공간을 확보하는 것이 우선된다면 **force\_remove\_log\_archives**를 yes로 설정하고, **log\_max\_archives**를 적당한 값으로 설정한다.

- **cubrid backupdb -r** 옵션을 사용하여 명령을 실행하면 삭제된다. 하지만 HA 환경에서는 -r 옵션을 사용하면 안 된다.



즉, 데이터베이스 운영 중에 보관 로그 볼륨을 가급적 남기고 싶지 않다면 **cubrid.conf**의 **log\_max\_archives** 값을 작은 값 또는 0으로 설정하되, **force\_remove\_log\_archives**의 값은 HA 환경이면 가급적 no로 설정한다.

## restoredb

데이터베이스 복구는 동일 버전의 CUBRID 환경에서 수행된 백업 작업에 의해 생성된 백업 파일, 활성 로그 및 보관 로그를 이용하여 특정 시점의 데이터베이스로 복구하는 작업이다. 데이터베이스 복구를 진행하려면 **cubrid restoredb** 유틸리티 또는 CUBRID 매니저를 사용한다.

**cubrid restoredb** 유틸리티는 최종 백업을 수행한 이후 모든 활성 로그와 보관 로그의 정보를 이용해 데이터베이스 백업본으로부터 데이터베이스를 복구한다.

```
cubrid restoredb [options] database_name
```

어떠한 옵션도 지정되지 않은 경우 기본적으로 마지막 커밋 시점까지 데이터베이스가 복구된다. 만약, 마지막 커밋 시점까지 복구하기 위해 필요한 활성 로그/보관 로그 파일이 없다면 마지막 백업 시점까지만 부분 복구된다.

```
cubrid restoredb demodb
```

다음은 **cubrid restoredb** 유틸리티와 결합할 수 있는 옵션을 정리한 표이다. 대소문자가 구분됨을 주의한다.

|                                         |                          |
|-----------------------------------------|--------------------------|
| <b>-d, --up-to-date=DATE</b>            | 주어진 시간으로 데이터베이스 상태를 복구   |
| <b>--list</b>                           | 복구를 하지 않고 모든 백업 볼륨을 나열   |
| <b>-B, --backup-file-path=PATH</b>      | 백업된 볼륨이 존재하는 디렉터리 경로     |
| <b>-l, --level=LEVEL</b>                | 복구에 사용될 백업의 레벨           |
| <b>-p, --partial-recovery</b>           | archive 로그가 존재하지 않으면 강제로 |
| <b>partial recovery</b>                 | 수행함                      |
| <b>-o, --output-file=FILE</b>           | 출력 메시지가 저장될 파일           |
| <b>-u, --use-database-location-path</b> | 데이터베이스 위치 파일에 설정된 경로에 데  |
| <b>이터베이스 볼륨과 로그 볼륨을 복구</b>              |                          |
| <b>-k, --keys-file-path</b>             | 복구에 사용될 키 파일 (_keys)의 경로 |

### -d, --up-to-date=DATE

**-d** 옵션으로 지정된 날짜-시간까지 데이터베이스를 복구한다. 사용자는 dd-mm-yyyy:hh:mi:ss(예: 14-10-2008:14:10:00)의 형식으로 복구 시점을 직접 지정할 수 있다. 만약 지정한 복구 시점까지 복구하기 위해 필요한 활성 로그/보관 로그 파일이 없다면 마지막 백업 시점까지만 부분 복구된다.

```
cubrid restoredb -d 14-10-2008:14:10:00 demodb
```

**backuptime**이라는 키워드를 복구 시점으로 지정하면 데이터베이스를 마지막 백업이 수행된 시점까지 복구한다.

```
cubrid restoredb -d backuptime demodb
```

### --list

대상 데이터베이스의 백업 파일에 관한 정보를 화면에 출력하며 복구는 수행하지 않는다. 이 옵션은 CUBRID 9.3부터 데이터베이스가 운영 중인 경우에도 수행 가능하다.

```
cubrid restoredb --list demodb
```

다음은 **--list** 옵션에 의해 출력되는 백업 정보의 예로서, 복구 작업을 수행하기 전에 대상 데이터베이스의 백업 파일이 최초 저장된 경로와 백업 수준을 검증할 수 있다.

```
*** BACKUP HEADER INFORMATION ***
Database Name: /local1/testing/demodb
DB Creation Time: Mon Oct 1 17:27:40 2008
Pagesize: 4096
Backup Level: 1 (INCREMENTAL LEVEL 1)
Start_lsa: 513|3688
Last_lsa: 513|3688
Backup Time: Mon Oct 1 17:32:50 2008
Backup Unit Num: 0
Release: 8.1.0
Disk Version: 8
Backup Pagesize: 4096
Zip Method: 0 (NONE)
Zip Level: 0 (NONE)
Previous Backup Level: 0
Time: Mon Oct 1 17:31:40 2008
(start_lsa was -1|-1)
Database Volume Name: /local1/testing/demodb_keys
Volume Identifier: -6, Size: 65 bytes (1 pages)
Database Volume Name: /local1/testing/demodb_vinf
Volume Identifier: -5, Size: 308 bytes (1 pages)
Database Volume Name: /local1/testing/demodb
Volume Identifier: 0, Size: 2048000 bytes (500 pages)
Database Volume Name: /local1/testing/demodb_lginf
Volume Identifier: -4, Size: 165 bytes (1 pages)
Database Volume Name: /local1/testing/demodb_bkvinf
Volume Identifier: -3, Size: 132 bytes (1 pages)
```

**--list** 옵션을 이용하여 출력된 백업 정보를 확인하면, 백업 파일이 백업 수준 1로 생성되었고, 앞의 백업 수준 0의 전체 백업이 수행된 시점을 확인할 수 있다. 따라서, 예시된 데이터베이스의 복구를 위해서는 백업 수준 0인 백업 파일과 백업 수준 1인 백업 파일이 준비되어야 한다.

**-B, --backup-file-path=PATH**

백업 파일이 위치하는 디렉터리를 지정할 수 있다. 만약, 이 옵션이 지정되지 않으면 시스템은 데이터베이스 위치 정보 파일인 **databases.txt**에 지정된 **log-path** 디렉터리에서 대상 데이터베이스를 백업했을 때 생성된 백업 정보 파일(*dbname \_bkvinf*)을 검색하고, 백업 정보 파일에 지정된 디렉터리 경로에서 백업 파일을 찾는다. 그러나, 백업 정보 파일이 손상되거나 백업 파일의 위치 정보가 삭제된 경우라면 시스템이 백업 파일을 찾을 수 없으므로, 관리자가 **-B** 옵션을 이용하여 백업 파일이 위치하는 디렉터리 경로를 직접 지정해야 한다.

```
cubrid restoredb -B /home/cubrid/backup demodb
```

데이터베이스의 백업 파일이 현재 디렉터리에 있는 경우, 관리자는 **-B** 옵션을 이용하여 백업 파일이 위치하는 디렉터리를 지정할 수 있다.

```
cubrid restoredb -B . demodb
```

**-l, --level=LEVEL**

대상 데이터베이스의 백업 수준(0, 1, 2)을 지정하여 복구를 수행한다. 백업 수준에 대한 자세한 내용은 **증분 백업**을 참조한다.

```
cubrid restoredb -l 1 demodb
```

**-p, --partial-recovery**

사용자 응답을 요청하지 않고 부분 복구를 수행하라는 명령이다. 백업 시점 이후에 기록된 활성 로그나 보관 로그가 완전하지 않을 때 기본적으로 시스템은 로그 파일이 필요하다는 것을 알리면서 실행 옵션을 입력하라는 요청 메시지를 출력하는데, **-p** 옵션을 이용하면 이러한 요청 메시지의 출력 없이 직접 부분 복구를 수행할 수 있다. 따라서, **-p** 옵션을 이용하여 복구를 수행하면 언제나 마지막 백업 시점까지 데이터가 복구된다.

```
cubrid restoredb -p demodb
```

**-p** 옵션이 지정되지 않은 경우, 사용자에게 실행 옵션을 선택하라는 요청 메시지는 다음과 같다.

```

Log Archive /home/cubrid/test/log/demodb_lgar002
is needed to continue normal execution.
Type
- 0 to quit.
- 1 to continue without present archive. (Partial recovery)
- 2 to continue after the archive is mounted/loaded.
```

```
- 3 to continue after changing location/name of archive.

```

- 옵션 0: 복구 작업을 더 이상 진행하지 않을 경우, 0을 입력한다.
- 옵션 1: 로그 파일 없이 부분 복구를 진행하려면, 1을 입력한다.
- 옵션 2: 복구 작업을 진행하기 위해 관리자는 현재 장치에 보관 로그를 위치시킨 후 2를 입력한다.
- 옵션 3: 복구 작업을 계속하기 위해 관리자는 로그 위치를 변경한 후 3을 입력한다.

#### **-o, --output-file=FILE**

대상 데이터베이스의 복구에 관한 진행 정보를 info\_restore라는 파일에 기록하는 명령이다.

```
cubrid restoredb -o info_restore demodb
```

#### **-u, --use-database-location-path**

데이터베이스 위치 정보 파일(**databases.txt**)에 지정된 경로에서 대상 데이터베이스를 복구하는 구문이다. **-u** 옵션은 A 서버에서 백업을 수행하고 B 서버에서 백업 파일을 복구하고자 할 때 사용할 수 있는 유용한 옵션이다.

```
cubrid restoredb -u demodb
```

#### **-k, --keys-file-path=PATH**

대상 데이터베이스 복구 시 필요한 키 파일을 지정한다. 올바른 키 파일이 주어지지 않았을 경우 복구에 실패한다. 옵션을 지정해주지 않을 경우 적절한 키 파일을 탐색하여 사용한다. 이에 대한 자세한 내용은 **백업 볼륨 암호화** 을 참고한다.

```
cubrid restoredb -k /home/cubrid/backup_keys/demodb_bk1_keys
demodb
```

데이터베이스 서버 시작이나 백업 볼륨 복구 시 서버 에러 로그 또는 restoredb 에러 로그 파일에 로그 회복(log recovery) 시작 시간과 종료 시간에 대한 NOTIFICATION 메시지를 출력하여, 해당 작업의 소요 시간을 확인할 수 있다. 해당 메시지에는 적용(redo)해야 할 로그의 개수와 로그 페이지 개수가 함께 기록된다.

```
Time: 06/14/13 21:29:04.059 - NOTIFICATION *** file ../../src/
transaction/log_recovery.c, line 748 CODE = -1128 Tran = -1, EID = 1
Log recovery is started. The number of log records to be applied:
96916. Log page: 343 ~ 5104.
.....
Time: 06/14/13 21:29:05.170 - NOTIFICATION *** file ../../src/
```

```
transaction/log_recovery.c, line 843 CODE = -1129 Tran = -1, EID = 4
Log recovery is finished.
```

## 복구 정책과 절차

데이터베이스를 복구할 때 고려해야 할 사항은 다음과 같다.

### · 백업 파일 준비

- 백업 파일 및 로그 파일이 저장된 디렉터리를 파악한다.
- 증분 백업으로 대상 데이터베이스가 백업된 경우, 각 백업 수준에 따른 백업 파일이 존재하는지를 파악한다.
- 백업이 수행된 CUBRID 데이터베이스의 버전과 복구가 이루어질 CUBRID 데이터베이스 버전이 동일한지를 파악한다.

### · 복구 방식 결정

- 부분 복구인지 전체 복구인지를 결정한다.
- 증분 백업 파일을 이용한 복구인지를 결정한다.
- 사용 가능한 복구 도구 및 복구 장비를 준비한다.

### · 복구 시점 판단

- 데이터베이스 서버가 종료된 시점을 파악한다.
- 장애 발생 전에 이루어진 마지막 백업 시점을 파악한다.
- 장애 발생 전에 이루어진 마지막 커밋 시점을 파악한다.

## 데이터베이스 복구 절차

다음은 백업 및 복구 작업의 절차를 시간별로 예시한 것이다.

1. 2008/8/14 04:30분에 운영이 중단된 *demodb* 를 전체 백업을 수행한다.
2. 2008/8/14 10:00분에 운영 중인 *demodb* 를 1차 증분 백업 수행한다.
3. 2008/8/14 15:00분에 운영 중인 *demodb* 를 1차 증분 백업을 수행한다. 2번의 1차 증분 백업 파일을 덮어쓴다.
4. 2008/8/14 15:30분에 시스템 장애가 발생하였고, 관리자는 *demodb* 의 복구 작업을 준비한다. 장애 발생 전의 마지막 커밋 시점이 15:25분이므로 이를 복구 시점으로 지정한다.
5. 관리자는 1.에서 생성된 전체 백업 파일 및 3.에서 생성된 1차 증분 백업 파일, 활성 로그 및 보관 로그를 준비하여 마지막 커밋 시점인 15:25 시점까지 *demodb* 를 복구한다.

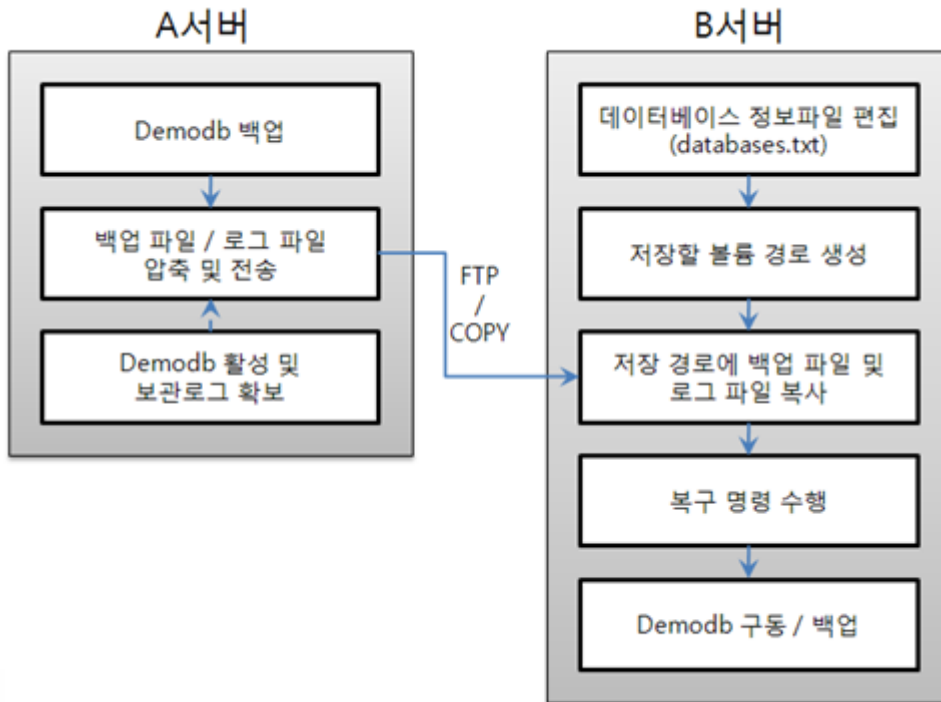
| Time            | Command                                                | 설명                                                                                                                   |
|-----------------|--------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|
| 2008/8/14 04:25 | cubrid server stop demodb                              | demodb 운영을 중단한다.                                                                                                     |
| 2008/8/14 04:30 | cubrid backupdb -S -D /<br>home/backup -l 0 demodb     | 오프라인에서 demodb 를 전체 백업하여 지정된 디렉터리에 백업 파일을 생성한다.                                                                       |
| 2008/8/14 05:00 | cubrid server start demodb                             | demodb 운영을 시작한다.                                                                                                     |
| 2008/8/14 10:00 | cubrid backupdb -C -D /<br>home/backup -l 1 demodb     | 온라인에서 demodb 를 1차 증분 백업하여 지정된 디렉터리에 백업 파일을 생성한다.                                                                     |
| 2008/8/14 15:00 | cubrid backupdb -C -D /<br>home/backup -l 1 demodb     | 온라인에서 demodb 를 1차 증분 백업하여 지정된 디렉터리에 백업 파일을 생성한다.<br>10:00에 생성된 1차 증분 백업 파일을 덮어쓴다.                                    |
| 2008/8/14 15:30 |                                                        | 시스템 장애가 발생한 시각이다.                                                                                                    |
| 2008/8/14 15:40 | cubrid restoredb -l 1 -d<br>08/14/2008:15:25:00 demodb | 전체 백업 파일, 1차 증분 백업 파일, 활성 로그 및 보관 로그를 기반으로 demodb 를 복구한다. 전체 백업 파일, 1차 증분된 백업 파일, 활성 로그 및 보관 로그에 의해 15:25 시점까지 복구된다. |

## 다른 서버로의 데이터베이스 복구

다음은 A 서버에서 demodb 를 백업하고, 백업된 파일을 기반으로 B 서버에서 demodb 를 복구하는 방법이다.

### 백업 환경과 복구 환경

A 서버의 /home/cubrid/db/demodb 디렉터리에서 *demodb* 를 백업하고, B 서버의 /home/cubrid/data/demodb 디렉터리에 *demodb* 를 복구하는 것으로 가정한다.



#### 1. A 서버에서 백업

A 서버에서 *demodb* 를 백업한다. 이보다 먼저 백업을 수행하였다면 이후 변경된 부분만 증분 백업을 수행할 수 있다. 백업 파일이 생성되는 디렉터리는 **-D** 옵션에 의해 지정하지 않으면, 기본적으로 로그 볼륨이 저장되는 위치에 생성된다. 다음은 권장되는 옵션을 사용한 백업 명령이며, 옵션에 관한 보다 자세한 내용은 **backupdb**를 참고한다.

```
cubrid backupdb -z demodb
```

#### 2. B 서버에서 데이터베이스 위치 정보 파일 편집

동일한 서버에서 백업 및 복구 작업이 이루어지는 일반적인 시나리오와는 달리, 타 서버 환경에서 백업 파일을 복구하는 시나리오에서는 B 서버의 데이터베이스 위치 정보 파일(**databases.txt**)에서 데이터베이스를 복구할 위치 정보를 추가해야 한다. 위 그림에서는 B 서버(호스트명은 pmlinux)의 /home/cubrid/data/demodb 디렉터리에 *demodb* 를 복구하는 것을 가정하였으므로, 이에 따라 데이터베이스 위치 정보 파일을 편집하고, 해당 디렉터리를 B 서버에서 생성한다.

데이터베이스 위치 정보는 한 라인으로 작성하고, 각 항목은 공백으로 구분한다. 한 라인은 [데이터베이스명] [데이터볼륨경로] [호스트명] [로그볼륨경로]의 형식으로 작성한다. 따라서 다음과 같이 *demodb* 의 위치 정보를 작성한다.

```
demodb /home/cubrid/data/demodb pmlinux /home/cubrid/data/demodb
```

### 3. B 서버로 백업 파일 전송

복구를 위해서는 대상 데이터베이스의 백업 파일이 필수적으로 준비되어야 한다. 따라서, A 서버에서 생성된 백업 파일(예: demodb\_bk0v000)을 B 서버에 전송한다. 즉, B 서버의 임의 디렉터리(예: /home/cubrid/temp)에는 백업 파일이 위치해야 한다.

#### 참고

백업 이후에 현재 시점까지 모두 복구하려면 백업 이후의 로그, 즉 활성 로그(예: demodb\_lgat)와 보관 로그(예: demodb\_lgar000)까지 모두 추가적으로 복사해야 된다. 활성 로그와 보관 로그는 복구될 데이터베이스의 로그 디렉터리, 즉 \$CUBRID/databases/databases.txt 파일에서 명시한 로그 파일의 디렉터리(예: \$CUBRID/databases/demodb/log)에 위치해야 한다. 하지만 백업 파일의 위치는 -D 옵션으로 명시할 수 있기 때문에 다른 위치에 있어도 된다.

또한, 백업 이후 추가된 로그를 반영하려면 보관 로그 파일이 삭제되기 전에 복사해야 하는데, 보관 로그의 삭제 관련 시스템 파라미터인 log\_max\_archives의 기본값이 0으로 설정되어 있으므로 백업 이후 보관 로그 파일이 삭제될 수 있다. 이러한 상황을 방지하기 위해, log\_max\_archives의 값을 적당히 크게 설정하여야 한다. [log\\_max\\_archives](#)를 참고한다.

### 4. B 서버에서 복구

B 서버로 전송한 백업 파일이 있는 디렉터리에서 **cubrid restoredb** 유틸리티를 호출하여 데이터베이스 복구 작업을 수행한다. -u 옵션에 의해 **databases.txt**에 지정된 디렉터리 경로에 demodb가 복구된다.

```
cubrid restoredb -u demodb
```

다른 위치에서 **cubrid restoredb** 유틸리티를 호출하려면, 다음과 같이 -B 옵션을 이용하여 백업 파일이 위치하는 디렉터리 경로를 지정해야 한다.

```
cubrid restoredb -u -B /home/cubrid/temp demodb
```

## unloaddb

데이터베이스를 언로드/로드하는 목적은 다음과 같다.

- 데이터베이스 볼륨을 재구성하여 데이터베이스 재구축
- 시스템이 다른 환경에서 마이그레이션 수행



- 버전이 다른 DBMS에서 마이그레이션 수행

```
cubrid unloaddb [options] database_name
```

**cubrid unloaddb**가 생성하는 파일은 다음과 같다.

- 스키마 파일(*database-name\_schema*): 해당 데이터베이스에 정의된 스키마 정보를 포함하는 파일이다.
- 객체 파일(*database-name\_objects*): 해당 데이터베이스에 포함된 인스턴스 정보를 포함하는 파일이다.
- 인덱스 파일(*database-name\_indexes*): 해당 데이터베이스에 정의된 인덱스 정보를 포함하는 파일이다.
- 트리거 파일(*database-name\_trigger*): 해당 데이터베이스에 정의된 트리거 정보를 포함하는 파일이다. 만약 데이터를 로딩하는 동안 트리거가 구동되는 것을 원치 않는다면, 데이터 로딩을 완료한 후에 트리거 정의를 로딩하면 된다.
- 분할된 스키마 파일(*database-name\_schema\_objecttype*): **--split-schema-files** 옵션을 사용할 경우 해당 데이터베이스에 정의된 스키마 정보를 오브젝트 단위로 분할하여 해당 오브젝트 정보만 가지고 있는 파일들이다. 분할된 파일 목록은 **--split-schema-files** 옵션을 참고 한다.
- 스키마 목록 파일(*database-name\_schema\_info*): **--split-schema-files** 옵션을 사용할 경우 분할된 스키마 정보 파일들의 목록을 가지고 있는 파일이다.

이러한 스키마, 객체, 인덱스, 트리거 파일은 같은 디렉터리에 생성된다.

다음은 **cubrid unloaddb** 에서 사용하는 [options]이다.

```
-u, --user=ID 데이터베이스 접근에 관한 사용자 이름
-p, --password=PASS 사용자 패스워드
-i, --input-class-file=FILE 테이블 이름이 기입된 파일
 --include-reference 참조되는 테이블 포함
 --input-class-only 명세한 클래스만 처리
 --estimated-size=NUMBER 인스턴스의 대략적인 개수
 --cached-pages=NUMBER cached 페이지 개수
-O, --output-path=PATH 출력 디렉터리 경로
-s, --schema-only 스키마만 처리
-d, --data-only 오브젝트만 처리
 --output-prefix=PREFIX 출력 파일의 prefix
 --hash-file=FILE 해쉬 파일이름
 --latest-image 오브젝트 (객체)를 언로드할 때, 데이터 볼륨의 최신 이미지를 언로드합니다.
 커밋되지않은 데이터와 제거되었지만 백업되지 않은
데이터가 포함될 수 있습니다
-t, --thread-count=COUNT 사용할 쓰레드 개수를 지정; 기본값: 1 최대값:
127
```

```

--enhanced-estimates 언로드할 테이블의 정확한 레코드 건수 수집; 기본
값 : 미적용
-v, --verbose 많은 상태 메시지를 출력
--use-delimiter 식별자 처음과 끝에 ''' 사용
-S, --SA-mode 독립 모드 실행
-C, --CS-mode 클라이언트 서버 모드 실행
--datafile-per-class 각 클래스별 오브젝트 파일 생성
--split-schema-files 스키마 정보를 오브젝트별로 분리하여 각각의 파일
로 생성
--skip-index-detail 인덱스를 생성할 때 지정된 WITH 절 옵션을 인쇄하
지 않음
--as-dba 로그인 사용자가 DBA 그룹의 구성원인 경우 DBA와
동일한 스키마 파일을 추출합니다.

```

#### **-u, --user=ID**

언로딩할 데이터베이스의 사용자 계정을 지정한다. 옵션을 지정하지 않으면 기본값은 **DBA**가 된다.

```
cubrid unloaddb -u dba -i table_list.txt demodb
```

#### **-p, --password=PASS**

언로딩할 데이터베이스의 사용자 암호를 지정한다. 옵션을 지정하지 않으면 빈 문자열을 입력한 것으로 간주한다.

```
cubrid unloaddb -u dba -p dba_pwd -i table_list.txt demodb
```

#### **-i, --input-class=FILE**

모든 테이블의 스키마와 인덱스를 언로드하되, 파일에 명시된 테이블의 데이터만 언로드한다. 언로딩할 사용자의 스키마와(-u 옵션 참고) 테이블의 스키마가 다른 경우 테이블 이름은 스키마 이름을 접두어로 사용해야 한다.

```
cubrid unloaddb -i table_list.txt demodb
```

다음은 입력 파일 table\_list.txt의 예이다.

```

table_1
table_2
..
table_n

```

**-i** 옵션이 **--input-class-only** 와 결합되면, **-i** 옵션의 입력 파일에서 지정된 테이블의 스키마, 인덱스, 데이터 파일만 언로드한다.

```
cubrid unloaddb --input-class-only -i table_list.txt demodb
```

-i 옵션이 --include-reference 와 결합되면, 참조되는 테이블도 함께 언로드된다.

```
cubrid unloaddb --include-reference -i table_list.txt demodb
```

### --include-reference

-i 옵션과 함께 사용되며, 참조되는 테이블도 함께 언로드된다.

### --input-class-only

-i 옵션과 함께 사용되며, -i 옵션의 입력 파일에서 지정된 테이블의 스키마 파일만 생성한다.

### --estimated-size=NUMBER

언로드할 데이터베이스의 레코드 저장을 위한 해시 메모리를 사용자 임의로 할당하기 위한 옵션이다. 만약 --estimated-size 옵션이 지정되지 않으면 최근의 통계 정보를 기반으로 데이터베이스의 레코드 수를 결정하게 되는데, 만약 최근 통계 정보가 갱신되지 않았거나 해시 메모리를 크게 할당하고 싶은 경우 이 옵션을 이용할 수 있다. 따라서, 옵션의 인수로 너무 적은 레코드 개수를 정의한다면 해시 충돌로 인해 언로드 성능이 저하된다.

```
cubrid unloaddb --estimated-size=1000 demodb
```

### --cached-pages=NUMBER

CUBRID는 특정 클래스(테이블)를 다른 클래스의 데이터 타입으로 정의할 수 있는 객체지향성 특성을 지원한다. 이러한 구조에서는 데이터 간의 연결을 위해 특정 레코드가 다른 레코드를 직접 가리키는 OID(Object Identifier) 참조가 발생한다. -cached-pages 옵션은 데이터 추출 과정에서 발생하는 객체 간 참조(OID reference)를 효율적으로 처리하기 위해 해시 테이블에 할당할 페이지 수를 지정할 수 있다. 각 페이지의 크기는 4,096바이트이며, 관리자는 시스템 메모리 용량과 처리 성능을 고려하여 이 값을 조정할 수 있다. 이 옵션을 지정하지 않으면 기본값은 100페이지이다.

```
cubrid unloaddb --cached-pages 500 demodb
```

### -O, --output-path=PATH

스키마와 객체 파일이 생성될 디렉터리를 지정한다. 옵션이 지정되지 않으면 현재 디렉터리에 생성된다.

```
cubrid unloaddb -O ./CUBRID/Databases/demodb demodb
```

지정된 디렉터리가 존재하지 않는 경우 다음과 같은 에러 메시지가 출력된다.

```
unloaddb: No such file or directory.
```

### **-s, --schema-only**

언로드 작업을 통해 생성되는 출력 파일 중 스키마 파일만 생성되도록 지정하는 옵션이다.

```
cubrid unloaddb -s demodb
```

### **-d, --data-only**

언로드 작업을 통해 생성되는 출력 파일 중 데이터 파일만 생성되도록 지정하는 옵션이다.

```
cubrid unloaddb -d demodb
```

### **--output-prefix=PREFIX**

언로드 작업에 의해 생성되는 스키마 파일과 객체 파일의 이름 앞에 붙는 prefix를 지정하기 위한 옵션이다. 예제를 수행하면 스키마 파일명은 `abcd_schema` 가 되고, 객체 파일명은 `abcd_objects` 가 된다. 만약, **--output-prefix** 옵션을 지정하지 않으면 언로드할 데이터베이스 이름이 prefix로 사용된다.

```
cubrid unloaddb --output-prefix abcd demodb
```

### **--hash-file=FILE**

해시 파일의 이름을 지정한다.

### **--latest-image**

인스턴스들을 언로드할 때 현재 데이터 볼륨의 가장 마지막 이미지에서 언로드한다. 커밋되지 않은 데이터나 삭제되었지만 백업되지 않은 데이터가 포함될 수 있다. 이 옵션을 지정하면 MVCC 버전은 무시되며 로그 볼륨을 참조하지 않는다.

### **-t, --thread-count=COUNT**

사용할 쓰레드 개수를 지정하는 옵션으로, COUNT 개수 만큼의 쓰레드를 이용해서 병렬로 수행하도록 하며 COUNT는 0 이상, 127 이하의 범위여야 한다. 설정을 생략하면 1로 지정한 것과 같고, 만약 0으로 지정되면 쓰레드 방식으로 동작하지 않는다. 쓰레드 방식을 지정하더라도 테이블에 Object 타입 또는, Object 타입을 내재하는 Set, MultiSet, Sequence(List) 타입이 있는 경우에는 해당 테이블에 대해서는 쓰레드 방식으로 동작하지 않는다.

```
cubrid unloaddb -t 4 demodb
```

### **--enhanced-estimates**

언로드할 테이블의 정확한 레코드 개수를 수집하는 옵션으로, -v 옵션을 함께 지정해서 사용해야 하며, -v 옵션을 사용하지 않는 경우 무시된다. 작업 대상 테이블의 레코드 수를 파악 할 때

통계 정보 대신 실제 값을 구한다. 이 옵션이 지정되면 레코드 수를 파악하기 위한 수행 시간이 추가되므로 언로드 수행시간이 길어질 수 있으므로 유의해서 사용해야 한다.

```
cubrid unloaddb --enhanced-estimates demodb
```

#### **-v, --verbose**

언로드 작업이 진행되는 동안 언로드되는 데이터베이스의 테이블 및 인스턴스에 관한 상세 정보를 화면에 출력하는 옵션이다.

```
cubrid unloaddb -v demodb
```

#### **--use-delimiter**

식별자의 시작과 끝에 겹따옴표(")를 기록한다. 기본 설정은 식별자의 시작과 끝에 겹따옴표를 기록하지 않는다.

#### **-S, --SA-mode**

독립 모드에서 데이터베이스를 언로드한다.

```
cubrid unloaddb -S demodb
```

#### **-C, --CS-mode**

클라이언트/서버 모드에서 데이터베이스를 언로드한다.

```
cubrid unloaddb -C demodb
```

#### **--datafile-per-class**

언로드 작업으로 생성되는 데이터 파일을 각 테이블별로 생성되도록 지정하는 옵션이다. 파일 이름은 <데이터베이스 이름>\_<테이블 이름>\_objects 로 생성된다. 단, 객체 타입의 칼럼 값은 모두 **NULL** 로 언로드되며, 언로드된 파일에는 %id class\_name class\_id 부분이 작성되지 않는다. 자세한 내용은 [가져오기용 파일 작성 방법](#) 을 참고한다.

```
cubrid unloaddb --datafile-per-class demodb
```

#### **--split-schema-files**

스키마 정보를 오브젝트 단위로 분할 생성하고, 분할된 파일 목록을 기록한 파일도 생성한다. 자세한 사용방법은 `loaddb` 의 `-schema-file-list=FILE` 를 참조한다. 이 옵션을 지정하지 않으면 모든 오브젝트가 포함된 하나의 스키마 파일이 생성된다.

```
cubrid unloaddb --split-schema-files demodb
```

스키마 정보를 오브젝트 단위로 분할하여 생성한 파일 목록

```
demodb_schema_user
demodb_schema_class
demodb_schema_vclass
demodb_schema_synonym
demodb_schema_serial
demodb_schema_procedure
demodb_schema_server
demodb_schema_pk
demodb_schema_fk
demodb_schema_uk
demodb_schema_grant
demodb_schema_vclass_query_spec
```

분할된 스키마 파일들의 목록을 가지고 있는 파일명

```
demodb_schema_info
```

### --as-dba

사용자가 DBA 그룹의 구성원인 경우, DBA 사용자인 경우와 동일한 내용으로 스키마 파일을 생성한다.

```
cubrid unloaddb -u u1_dba_group --as-dba demodb
```

### --skip-index-detail

이 옵션은 인덱스 생성시 지정된 WITH 절 옵션들을 출력하지 않는다. unloaddb는 **print\_index\_detail** 설정값에 상관없이 항상 WITH 절의 옵션들을 출력하는 것이 기본적인 동작이다. 이 옵션이 지정되면 WITH 절의 옵션들은 생략하고 출력하게 된다. 영향을 받는 출력 파일은 스키마파일(*database-name\_schema*)과 인덱스파일(*database-name\_indexes*)이다. 이 옵션이 생략되면 *deduplicate level*이 0이더라도 반드시 명시적으로 출력된다.

```
cubrid unloaddb --skip-index-detail demodb
```

아래 예시에서 이 옵션의 지정 여부에 따른 차이를 확인해 볼 수 있다.

스키마파일

--skip-index-detail 미지정

```
ALTER CLASS [dba].[ts] ADD CONSTRAINT
[fk_ts_fid] FOREIGN KEY([fid]) WITH
DEDUPLICATE=0 REFERENCES [dba].[tm]
```

```
ON DELETE RESTRICT ON UPDATE
RESTRICT ;
```

--skip-index-detail 지정

```
ALTER CLASS [dba].[ts] ADD CONSTRAINT
[fk_ts_fid] FOREIGN KEY([fid])
REFERENCES [dba].[tm] ON DELETE
RESTRICT ON UPDATE RESTRICT ;
```

## 인덱스파일

--skip-index-detail 미지정

```
CREATE INDEX [idx_val] ON [dba].[ts]
([val]) WITH DEDUPLICATE=0;
```

--skip-index-detail 지정

```
CREATE INDEX [idx_val] ON [dba].[ts]
([val]);
```

### 참고

언로드 시간을 단축하기 위해서 서로 다른 테이블에 대한 언로드는 동시에 실행 가능하며 **unloaddb.sh** 스크립트를 이용하면 이러한 과정을 쉽게 할 수 있다 ([unloaddb.sh 스크립트](#) 참고).

## loaddb

데이터베이스 로드는 다음과 같은 경우에 **cubrid loaddb** 유틸리티를 이용하여 수행된다.

- 예전 버전의 CUBRID 데이터베이스를 새로운 버전의 데이터베이스로 마이그레이션하는 경우
- 타 DBMS의 데이터베이스를 CUBRID 데이터베이스로 마이그레이션하는 경우
- **INSERT** 구문 실행보다 빠른 성능으로 대용량 데이터를 입력하는 경우

일반적으로 **cubrid loaddb** 유틸리티는 **cubrid unloaddb** 유틸리티가 생성한 파일(스키마 정의 파일, 객체 입력 파일, 인덱스 정의 파일)을 사용한다.

```
cubrid loaddb [options] database_name
```

## 입력 파일

- 스키마 파일(*database-name\_schema*): 언로드 작업에 의해 생성된 파일로서, 데이터베이스에 정의된 스키마 정보를 포함하는 파일이다.
- 객체 파일(*database-name\_objects*): 언로드 작업에 의해 생성된 파일로서, 데이터베이스에 포함된 레코드 정보를 포함하는 파일이다.
- 인덱스 파일(*database-name\_indexes*): 언로드 작업에 의해 생성된 파일로서, 데이터베이스에 정의된 인덱스 정보를 포함하는 파일이다.
- 트리거 파일(*database-name\_trigger*): 언로드 작업에 의해 생성된 파일로서, 데이터베이스에 정의된 트리거 정보를 포함하는 파일이다.
- 사용자 정의 객체 파일(*user\_defined\_object\_file*): 대용량 데이터 입력을 위해 사용자가 테이블 형식으로 작성한 입력 파일이다([가져오기용 파일 작성 방법](#) 참고).
- 스키마 목록 파일(*database-name\_schema\_info*): **--split-schema-files** 옵션과 함께 언로드 작업에 의해 생성된 파일로서, 데이터베이스에 정의된 스키마 정보를 오브젝트 단위로 분할된 오브젝트 파일들의 목록을 가지고 있는 파일이다.(`unloaddb --split-schema-files` 참고)

## 입력 파일 처리 순서

**cubrid loaddb** 유틸리티는 아래와 같이 특정 순서대로 파싱을 하며 입력 파일을 처리한다. 이것은 loaddb의 정확성과 일관성을 보장한다.

1. 스키마 파일 로드 (기본키 포함)
2. 오브젝트 파일 로드
3. 인덱스 파일 로드 (보조키와 외래키)
4. 트리거 파일 로드

## 로딩 모드

**cubrid loaddb** 유틸리티는 데이터를 데이터베이스에 로드하기 위해서 두가지 모드를 제공한다.

- **Stand-alone** 모드에서는 단일 스레드 환경에서 오프라인으로 데이터를 로드한다. CUBRID 10.1과 그 이하 버전에서는 이 모드만 제공된다.
- **Client-server** 모드에서는 데이터베이스 서버에 접속하여 온라인으로 데이터를 로드하며 병렬로 처리된다. **Client-server** 모드는 **stand-alone** 모드보다 훨씬 빠르지만 오브젝트의 참조와 클래스/공유 속성은 지원하지 않는다.

다음은 **cubrid loaddb** 에서 사용하는 [options]이다.

|                                  |                      |
|----------------------------------|----------------------|
| <code>-u, --user=ID</code>       | 데이터베이스 접근에 관한 사용자 이름 |
| <code>-p, --password=PASS</code> | 사용자 패스워드             |



|                               |                                                |
|-------------------------------|------------------------------------------------|
| -S, --SA-mode                 | 독립 모드 실행                                       |
| -C, --CS-mode                 | 클라이언트 서버 모드 실행                                 |
| --data-file-check-only        | 데이터 파일을 로딩하지 않고 선택스만 검사                        |
| -l, --load-only               | 선택스 체크없이 데이터 파일 로딩                             |
| --estimated-size=NUMBER       | 인스턴스의 대략적인 개수                                  |
| -v, --verbose                 | 많은 상태 정보를 출력                                   |
| -c, --periodic-commit=COUNT   | 주기적인 커밋에 대한 카운트                                |
| --no-oid                      | OID 사용 안 함                                     |
| --no-statistics               | 로딩 후에 통계 정보 갱신하지 않음                            |
| -s, --schema-file=FILE[:LINE] | 파일[:라인] 정보로 스키마 생성                             |
| -i, --index-file=FILE[:LINE]  | 파일[:라인] 정보로 인덱스 생성                             |
| -d, --data-file=FILE          | 적재할 데이터 파일                                     |
| -t, --table=TABLE             | 데이터를 적재할 테이블 이름                                |
| --error-control-file=FILE     | 적재 시 발생하는 에러에 대한 제어 파일                         |
| --ignore-class-file=FILE      | 적재하지 않을 클래스 이름이 있는 파일                          |
| --trigger-file=FILE[:LINE]    | 파일[:라인] 정보로 트리거 생성                             |
| --no-user-specified-name      | 소유자 이름 없이 객체 이름으로 CLASS, SERIAL 및 TRIGGER를 찾음. |
| --schema-file-list=FILE       | 이 파일의 내용은 loadadb에서 사용할 스키마 파일 이름 목록이다.        |

#### -u, --user=ID

레코드를 로딩할 데이터베이스의 사용자 계정을 지정한다. 옵션을 지정하지 않으면 기본값은 **PUBLIC** 이 된다.

```
cubrid loadadb -u admin -d demodb_objects newdb
```

#### -p, --password=PASS

레코드를 로딩할 데이터베이스의 사용자 암호를 지정한다. 옵션을 지정하지 않으면 암호 입력을 요청하는 프롬프트가 출력된다.

```
cubrid loadadb -u dba -p pass -d demodb_objects newdb
```

#### -S, --SA-MODE

데이터를 **stand-alone** 모드로 데이터베이스에 로드한다. 이것은 **loadadb** 의 기본 로드 모드이며 CUBRID 10.1 이하의 버전과 완벽히 호환된다.

```
cubrid loadadb -S -u dba -d demodb_objects newdb
```

#### -C, --CS-MODE

이 옵션에서는 **loadadb** 가 가동중인 데이터베이스 서버 프로세스에 접속한다. **stand-alone** 모드와 다르게 **client-server** 모드에서는 여러 개의 스레드를 사용하여 데이터를 로드할 수 있다. 이 모드에서는 동시에 여러 개의 **loadadb** 세션을 연결하며, 대용량의 데이터를 로드할 때 특히

우수한 성능을 발휘한다. 이 모드에서는 오브젝트 참조나 클래스/공유 속성은 제공하지 않는다.

```
cubrid loaddb -C -u dba -d demodb_objects newdb
```

### 참고

**HA** (High Availability)환경의 **client-server** 모드(-C, --CS-MODE)에서는 -d 옵션(data loading)만 사용 가능하다.

### --data-file-check-only

demodb\_objects에 포함된 데이터의 구문이 정상인지 확인만 하며 데이터를 데이터베이스에 로딩하지 않는다.

```
cubrid loaddb --data-file-check-only -d demodb_objects newdb
```

### -l, --load-only

**Stand-alone** 모드는 기본적으로 데이터베이스에 로드하기 전에 전체 데이터에 대해 사전 구문 검사가 실행된다. 오류가 발생하면 데이터의 로드는 실행되지 않고 오류가 보고된다.

**-l** 옵션을 이용하면 로드 전에 전체 데이터에 대해 사전 구문 검사 실행이 생략되어 속도가 빠르다. 그러나, 각 레코드를 데이터베이스에 로드하기 전에 구문 검사가 진행된다. 이 과정에서 구문 오류가 발생하는 경우 로드는 중지되며, 결과적으로 데이터의 일부분만 로드되는 경우가 발생할 수도 있다.

```
cubrid loaddb -l -d demodb_objects newdb
```

### --estimated-size=NUMBER

언로드할 레코드의 수가 기본값인 5,000개보다 많은 경우 로딩 성능 향상을 위해 사용할 수 있다. 이 옵션을 통해 레코드 저장을 위한 해시 메모리를 크게 할당함으로써 로드 성능을 향상시킬 수 있다.

```
cubrid loaddb --estimated-size 8000 -d demodb_objects newdb
```

### -v, --verbose

데이터베이스 로딩 작업이 진행되는 동안, 로딩되는 데이터베이스의 테이블 및 레코드에 관한 상세 정보를 화면에 출력한다. 진행 단계, 로딩되는 클래스, 입력된 레코드의 개수와 같은 상세 정보를 확인할 수 있다.

```
cubrid loaddb -v -d demodb_objects newdb
```

### **-c, --periodic-commit=COUNT**

지정된 개수의 레코드가 데이터베이스에 입력될 때마다 주기적으로 커밋을 실행한다. **-c** 옵션이 지정되지 않은 경우, 데이터 파일에 포함된 모든 레코드가 데이터베이스로 로드된 후에 트랜잭션이 커밋된다. 또한, **-c** 옵션이 **-s** 옵션이나 **-i** 옵션과 함께 사용하는 경우에는 100개의 DDL문이 로드될 때마다 주기적으로 커밋을 실행한다.

권장되는 커밋 주기는 로딩되는 데이터에 따라 다른데, 스키마 로딩의 경우에는 **-c**의 인수를 50으로, 인덱스 로딩의 경우는 인수를 1로 설정하는 것이 권고된다. 레코드 로딩의 경우는 10,240이 권고되며 기본값이다.

CUBRID 10.1 또는 그 이하의 버전에서 **--periodic-commit**의 기본값은 없으며 따라서 전체 데이터를 로드한 후에 트랜잭션 커밋을 수행할 것이다. 현재 상태에서, 이전과 같이 동작하기를 원한다면 **--periodic-commit**을 매우 큰 값으로 설정하라. 그러면 모든 데이터가 로드된 후에 커밋이 실행될 것이다.

```
cubrid loaddb -c 100 -d demodb_objects newdb
```

### **--no-oid**

demodb\_objects에 포함된 OID를 무시하고 레코드를 newdb로 로딩하는 명령이다.

```
cubrid loaddb --no-oid -d demodb_objects newdb
```

### **--no-statistics**

demodb\_objects를 로딩한 후 newdb의 통계 정보를 갱신하지 않는 명령이다. 특히, 대상 데이터베이스의 데이터 용량에 비해 매우 적은 데이터만 로딩할 경우 이 옵션을 이용하여 로드 성능을 향상시킬 수 있다.

```
cubrid loaddb --no-statistics -d demodb_objects newdb
```

### **-s, --schema-file=FILE[:LINE]**

스키마 파일의 LINE번째부터 정의된 스키마 정보를 새로 생성한 newdb에 로딩하는 구문이다. **-s** 옵션을 이용하여 스키마 정보를 먼저 로딩한 후, 실제 레코드를 로딩할 수 있다.

다음 예제에서 demodb\_schema 파일은 언로드 작업에 의해 생성된 파일이며 언로드된 데이터베이스의 스키마 정보를 포함한다.

```
cubrid loaddb -u dba -s demodb_schema newdb
```

```
Start schema loading.
Total 86 statements executed.
```

```
Schema loading from demodb_schema finished.
Statistics for Catalog classes have been updated.
```

다음은 demodb에 정의된 트리거 정보를 새로 생성한 newdb에 로딩하는 구문이다. demodb\_trigger 파일은 언로드 작업에 의해 생성된 파일이며, 언로드된 데이터베이스의 트리거 정보를 포함한다. 레코드를 모두 로딩한 후, -s 옵션을 이용하여 트리거를 생성할 것을 권장한다.

```
cubrid loaddb -u dba -s demodb_trigger newdb
```

### **-i, --index-file=FILE[:LINE]**

인덱스 파일의 LINE번째부터 정의된 인덱스 정보를 데이터베이스에 로딩하는 명령이다. 다음 예제에서, demodb\_indexes 파일은 언로드 작업에 의해 생성된 파일이며 언로드된 데이터베이스의 인덱스 정보를 포함한다. -d 옵션을 이용하여 레코드를 로딩한 후, -i 옵션을 이용하여 인덱스를 생성할 수 있다.

```
cubrid loaddb -c 100 -d demodb_objects newdb
cubrid loaddb -u dba -i demodb_indexes newdb
```

### **-d, --data-file=FILE**

-d 옵션을 이용하여 데이터 파일 또는 사용자 정의 객체 파일을 지정함으로써 레코드 정보를 newdb로 로딩하는 명령이다. demodb\_objects 파일은 언로드 작업에 의해 생성된 객체 파일이거나, 사용자가 대량의 데이터 로딩을 위하여 작성한 사용자 정의 객체 파일 중 하나이다.

```
cubrid loaddb -u dba -d demodb_objects newdb
```

### **-t, --table=TABLE**

로딩할 데이터 파일에 테이블 이름 헤더가 생략되어 있는 경우, 이 옵션 뒤에 테이블 이름을 지정한다. 테이블 이름은 스키마 이름을 접두사로 사용해야 한다.

```
cubrid loaddb -u dba -d demodb_objects -t tbl_name newdb
```

### **--error-control-file=FILE**

데이터베이스 로드 작업 중에 발생하는 에러 중 특정 에러를 처리하는 방식에 관해 명시한 파일을 지정하는 옵션이다.

```
cubrid loaddb --error-control-file=error_test -d demodb_objects
newdb
```

서버 에러 코드 이름은 `$CUBRID/include/dbi.h` 파일을 참고하도록 한다.

에러 코드(에러 번호) 별 에러 메시지는 **\$CUBRID/msg/<문자셋 이름>/cubrid.msg** 파일의 \$set 5 MSGCAT\_SET\_ERROR 이하에 있는 번호들을 참고하도록 한다.

```
vi $CUBRID/msg/en_US/cubrid.msg

$set 5 MSGCAT_SET_ERROR
1 Missing message for error code %1$d.
2 Internal system failure: no more specific information is
available.
3 Out of virtual memory: unable to allocate %1$d memory bytes.
4 Has been interrupted.
...
670 Operation would have caused one or more unique constraint
violations.
...
```

특정 에러 명세 파일의 형식은 다음과 같다.

- **-<에러 코드> : <에러 코드>에 해당하는 에러를 무시하도록 설정 (loaddb 수행 중 해당 에러가 발생해도 계속 수행)**
- **+<에러 코드> : <에러 코드>에 해당하는 에러를 무시하지 않도록 설정 (loaddb 수행 중 해당 에러가 발생하면 작업을 종료함)**
- **+DEFAULT : 24번부터 33번까지의 에러를 무시하지 않도록 설정**

**--error-control-file** 옵션으로 에러 명세 파일을 설정하지 않을 경우, **loaddb** 유틸리티는 기본적으로 24번부터 33번까지의 에러를 무시하도록 설정되어 있다. 이들은 데이터베이스 볼륨의 여유 공간이 얼마 남지 않았다는 경고성 에러로서, 이후 할당된 데이터베이스 볼륨의 여유 공간이 없어지면 자동으로 범용 볼륨(generic volume)을 생성하게 된다.

다음은 에러 명세 파일을 작성한 예이다.

- **+DEFAULT**를 설정하여, 24번부터 33번까지의 DB 볼륨 여유 공간 경고성 에러는 무시되지 않는다.
- 앞에서 -2를 설정했으나, 뒤에서 +2를 설정했기 때문에 2번 에러 코드는 무시되지 않는다.
- -670을 설정하여, 670번 에러인 고유성 위반 에러(unique violation error)는 무시된다.
- #-115는 앞에 #이 있어 커멘트 처리되었다.

```
vi error_file

+DEFAULT
-2
-670
```

```
#-115 --> comment
+2
```

### --ignore-class-file=FILE

로딩 작업 중 무시할 클래스 목록을 명시한 파일을 지정한다. 지정된 파일에 포함된 클래스를 제외한 나머지 클래스의 레코드만 로딩된다.

```
cubrid loaddb --ignore-class-file=skip_class_list -d
demodb_objects newdb
```

### --trigger-file=FILE[:LINE]

트리거 파일의 LINE번째부터 정의된 트리거 정보를 새로 생성한 newdb에 로딩하는 구문이다.

```
cubrid loaddb --trigger-file=demodb_triggers -d demodb_objects
newdb
```

### --no-user-specified-name

11.2 이전 버전에서 언로드된 파일은 **--no-user-specified-name** 옵션을 사용하는 경우에만 11.2 버전에서 로드할 수 있다.

이 옵션은 -u 옵션과 함께 데이터베이스의 사용자 계정이 **DBA**로 지정된 경우에만 동작한다.

```
cubrid loaddb -u dba -s demodb_schema -d demodb_objects --no-
user-specified-name demodb
```

### ⚠ 경고

**--no-user-specified-name** 옵션은 11.2버전 또는 11.2 이후 버전에서 언로드된 파일을 로드할 때 사용하면 안 된다.

### --schema-file-list=FILE

목록에 포함된 모든 스키마 파일들을 한번에 로딩하는 옵션이다. 스키마 파일 목록이 있어야 하며, 이 목록 파일의 맨 위 스키마 파일부터 아래로 순차적으로 로딩된다. 이 옵션과 -s 옵션은 함께 사용할 수 없다.

이 목록 파일은 unloaddb에서 **--split-schema-files** 옵션을 사용할 때 자동으로 생성된다.

```
cubrid loaddb -u dba --schema-file-list=demodb_schema_info
```

**⚠ 경고**

이 파일에 작성된 스키마 파일의 순서를 변경할 수 있으나 순서가 잘못된 경우 로딩 오류가 발생할 수 있다. 예를 들어, CLASS 정보가 있는 파일이 로드되지 않은 상태에서 PK 파일을 먼저 로드하면 오류가 발생할 수 있다. 따라서 목록에서 CLASS 정보를 로드하기 위한 파일명이 PK를 로드하기 위한 파일보다 위에 있어야 한다.

**⚠ 경고**

**--no-logging** 옵션을 사용하면 **loaddb** 를 수행하면서 트랜잭션 로그를 저장하지 않으므로 데이터 파일을 빠르게 로드할 수 있다. 그러나 로드 도중 파일 형식이 잘못되거나 시스템이 다운되는 등의 문제가 발생했을 때 데이터를 복구할 수 없으므로 데이터베이스를 새로 구축해야 한다. 즉, 데이터를 복구할 필요가 없는 새로운 데이터베이스를 구축하는 경우를 제외하고는 사용하지 않도록 주의한다. 이 옵션을 사용하면 고유 위반 등의 오류를 검사하지 않아 기본 키에 값이 중복되는 경우 등이 발생할 수 있으므로, 사용 시 이러한 문제를 반드시 감안해야 한다.

**⚠ 경고**

**loaddb** 실행 시 대량의 데이터 로딩에 대한 성능 보장과 로딩 데이터 간의 의존성을 최소화하기 위해 외래키 제약 조건에 대한 검사를 수행하지 않는다. 그러므로, HA 환경에서 **-C** 옵션을 이용하여 데이터를 로딩할 경우 슬레이브 또는 레플리카 노드 복제 과정에서 외래키 제약 조건 검사가 수행되어 마스터 노드와 데이터 불일치가 발생할 수 있다.

## 가져오기용 파일 작성 방법

**cubrid loaddb** 유틸리티에서 사용되는 객체 입력 파일을 직접 작성하여 사용하면 데이터베이스에 대량의 데이터를 보다 신속하게 추가할 수 있다. 객체 입력 파일은 간단한 테이블 모양의 형식으로 구성되며 주석, 명령 라인, 데이터 라인으로 이루어진 텍스트 파일이다.

## 주석

CUBRID에서는 주석은 두 개의 연속된 하이픈(-- )을 이용하여 처리한다.

```
-- This is a comment!
```

## 명령 라인

명령 라인은 퍼센트(%) 문자로 시작하며, 명령어로는 클래스를 정의하는 **%class** 명령어와, 클래스 식별을 위해 사용하는 별칭(alias)이나 식별자(identifier)를 정의하는 **%id** 명령어가 있다.

- 클래스에 식별자 부여

**%id** 를 이용하여 참조 관계에 있는 클래스에 식별자를 부여할 수 있다.

```
%id [schema_name.]class_name class_id
schema_name:
 identifier
class_name:
 identifier
class_id:
 integer
```

**%id** 명령어에 의해 명시된 *class\_name*은 해당 데이터베이스에 정의된 클래스 이름이다. *class\_name*은 *schema\_name*을 접두사로 사용하며, *schema\_name*은 클래스가 정의된 스키마 이름이다. *schema\_name*은 생략할 수 있으며, *schema\_name*을 생략하면 -u 옵션과 함께 지정된 데이터베이스의 사용자 이름이 스키마 이름으로 사용된다.

```
%id [employee] 21
%id [office] 22
%id [project] 23
%id [phone] 24
```

*schema\_name*과 *class\_name*은 대괄호([ ])로 묶어야 한다. *schema\_name*과 *class\_name*의 각각을 대괄호로 묶거나, 함께 대괄호로 묶을 수 있다.

```
%id [public].[employee] 21
%id [public].[office] 22
%id [public].[project] 23
%id [public].[phone] 24
```

```
%id [public.employee] 21
%id [public.office] 22
%id [public.project] 23
%id [public.phone] 24
```

- 클래스 및 속성 명시

**%class** 명령어를 이용하여 데이터가 로딩될 클래스(테이블) 및 속성(칼럼)을 명시하며, 명시된 속성의 순서에 따라 데이터 라인이 작성되어야 한다. **cubrid loaddb** 유틸리티를 실행할 때 **-t** 옵션으



로 클래스 이름을 제공하는 경우에는 데이터 파일에 클래스 및 속성을 명시하지 않아도 된다. 단, 데이터가 작성되는 순서는 클래스 생성 시의 속성 순서를 따라야 한다.

```
%class [schema_name.]class_name (attr_name [attr_name...])
schema_name:
 identifier
class_name:
 identifier
attr_name:
 identifier
```

데이터를 로딩하고자 하는 데이터베이스에는 클래스가 이미 정의되어 있어야 한다.

**%class** 명령어에 의해 명시된 *class\_name*은 해당 데이터베이스에 정의된 클래스 이름이며, *schema\_name*은 클래스가 정의된 스키마 이름이다. *attr\_name*은 정의된 속성 이름을 의미한다. *schema\_name*과 *class\_name*의 작성 방법은 위와 동일하다.

다음은 *employee*라는 클래스에 데이터를 입력하기 위하여 **%class** 명령으로 클래스 및 3개의 속성을 명시한 예제이다. **%class** 명령 다음에 나오는 데이터 라인에서는 3개의 데이터가 입력되어야 하며, 이는 **참조 관계 설정**을 참조한다.

```
%class [public].[employee] (name age department)
```

## 데이터 라인

데이터 라인은 **%class** 명령 라인 다음에 위치하며, 입력되는 데이터는 **%class** 명령에 의해 명시된 클래스 속성과 타입이 일치해야 한다. 만약, 명시된 속성과 타입이 일치하지 않으면 데이터 로드 작업은 중지된다.

또한, 각각의 속성에 대응되는 데이터는 적어도 하나의 공백에 의해 분리되어야 하며, 한 라인에 작성되는 것이 원칙이다. 다만, 입력되는 데이터가 많은 경우에는 첫 번째 데이터 라인의 맨 마지막 데이터 다음에 플러스 기호(+)를 명시하여 다음 라인에 데이터를 연속적으로 입력할 수 있다. 이 때, 맨 마지막 데이터와 플러스 기호 사이에는 공백이 허용되지 않음을 유의한다.

### · 인스턴스 입력

다음과 같이 명시된 클래스 속성과 타입이 일치하는 인스턴스를 입력할 수 있다. 각각의 데이터는 적어도 하나의 공백에 의해 구분된다.

```
%class [public].[employee] (name)
'jordan'
'james'
'garnett'
'malone'
```

## · 인스턴스 번호 부여

데이터 라인의 처음에 '번호:'의 형식으로 해당 인스턴스에 대한 번호를 부여할 수 있다. 인스턴스 번호는 명시된 클래스 내에서 유일한 양수이며, 번호와 콜론(:) 사이에는 공백이 허용되지 않는다. 이와 같이 인스턴스 번호를 부여하는 이유는 추후 객체 참조 관계를 설정하기 위함이다.

```
%class [public].[employee] (name)
1: 'jordan'
2: 'james'
3: 'garnett'
4: 'malone'
```

## · 참조 관계 설정

@ 다음에 참조하는 클래스를 명시하고, 수직바(|) 다음에 참조하는 인스턴스의 번호를 명시하여 객체 참조 관계를 설정할 수 있다.

```
@class_ref | instance_no
class_ref:
 class_name
 class_id
```

@ 다음에는 클래스명 또는 클래스 id를 명시하고, 수직바(|) 다음에는 인스턴스 번호를 명시한다. 수직바(|)의 양쪽에는 공백을 허용하지 않는다. `class_ref`의 작성 방법은 **명령 라인**의 `schema_name`과 `class_name`의 작성 방법과 동일하다. 하지만 `schema_name`과 `class_name` 각각을 대괄호([ ])로 묶는 것은 허용하지 않는다.

다음은 `paycheck` 클래스에 인스턴스를 입력하는 예제이며, `name` 속성은 `employee` 클래스의 인스턴스를 참조한다. 마지막 라인과 같이 앞에서 정의되지 아니한 인스턴스 번호를 이용하여 참조 관계를 설정하는 경우 해당 데이터는 **NULL**로 입력된다.

```
%class [public].[paycheck] (name department salary)
@[public.employee]|1 'planning' 8000000
@[public.employee]|2 'planning' 6000000
@[public.employee]|3 'sales' 5000000
@[public.employee]|4 'development' 4000000
@[public.employee]|5 'development' 5000000
```

클래스에 식별자 부여에서 **%id** 명령어로 `employee` 클래스에 21이라는 식별자를 부여했으므로, 위의 예를 다음과 같이 작성할 수 있다.

```
%class [public].[paycheck] (name department salary)
@21|1 'planning' 8000000
@21|2 'planning' 6000000
@21|3 'sales' 5000000
```

```
@21|4 'development' 4000000
@21|5 'development' 5000000
```

## 데이터베이스 마이그레이션

신규 버전의 CUBRID 데이터베이스를 사용하기 위해서는 기존 버전의 CUBRID 데이터베이스를 신규 버전의 CUBRID 데이터베이스로 이전하는 작업을 진행해야 할 경우가 있다. 이때 CUBRID에서 제공하는 텍스트 파일로 내보내기과 텍스트 파일에서 가져오기 기능을 활용할 수 있다.

**cubrid unloadb** 및 **cubrid loadb** 유틸리티를 이용하는 마이그레이션 절차를 설명한다.

### 권장 시나리오 및 절차

기존 버전의 CUBRID가 운영 중인 상태에서 적용할 수 있는 마이그레이션 시나리오를 설명한다. 데이터베이스 마이그레이션을 위해서는 **cubrid unloadb**와 **cubrid loadb** 유틸리티를 사용한다. 자세한 내용은 **unloadb** 및 **loadb** 를 참조한다.

#### 1. 기존 CUBRID 서비스 종료

**cubrid service stop** 을 실행하여 기존 CUBRID로 운영되는 모든 서비스 프로세스를 종료한 후, CUBRID 관련 프로세스들이 모두 정상 종료되었는지 확인한다.

CUBRID 관련 프로세스들이 모두 정상 종료되었는지 확인하려면, Linux에서는 **ps -ef|grep cub\_**를 실행한다. cub\_로 시작하는 프로세스가 없으면 정상적으로 종료된 것이다. Windows에서는 <Ctrl + Alt + Delete> 키를 누른 후 [작업 관리자 시작]을 선택한다. [프로세스] 탭에 cub\_로 시작하는 프로세스가 없으면 정상적으로 종료된 것이다. CUBRID 서비스 종료 후에도 관련 프로세스가 존재하면 Linux에서는 **kill** 명령으로 강제 종료한 후 **ipcs -m** 명령으로 CUBRID 브로커가 사용 중이던 공유 메모리를 확인하고 삭제한다. Windows에서는 작업 관리자의 [프로세스] 탭에서 해당 이미지 이름을 마우스 오른쪽 버튼으로 클릭하고 [프로세스 끝내기]를 선택하여 강제 종료한다.

#### 2. 기존 데이터베이스 백업

**cubrid backupdb** 유틸리티를 이용하여 기존 버전의 데이터베이스 백업을 수행한다. 그 이유는 데이터베이스 언로드/로드 작업 중 발생 가능한 장애에 대비하기 위함이다. 데이터베이스 백업에 관한 자세한 내용은 **backupdb**를 참조한다.

#### 3. 기존 데이터베이스 언로드

**cubrid unloadb** 유틸리티를 이용하여 기존 버전의 CUBRID에서 생성된 데이터베이스를 언로드한다. 데이터베이스 언로드에 관한 자세한 내용은 **unloadb** 를 참조한다.

#### 4. 기존 CUBRID의 환경 설정 파일 보관

**CUBRID/conf** 디렉터리 아래의 **cubrid.conf**, **cubrid\_broker.conf**, **cm.conf** 등의 환경 설정 파일을 보관한다. 이는 기존 CUBRID 데이터베이스 환경에 적용된 파라미터 설정값을 신규 CUBRID 데이터베이스 환경에서 편리하게 적용할 수 있기 때문이다.

## 5. 신규 버전의 CUBRID 설치

기존 버전의 CUBRID에서 생성된 데이터의 백업 및 언로드 작업이 완료되었으므로, 기존 버전의 CUBRID 및 데이터베이스를 삭제하고 신규 버전의 CUBRID를 설치한다. CUBRID 설치에 대한 자세한 내용은 **시작하기**를 참조한다.

## 6. 신규 CUBRID의 환경 설정

**기존 CUBRID의 환경 설정 파일 보관하기** 에서 보관한 기존 데이터베이스의 환경 설정 파일을 참고하여 신규 버전의 CUBRID 환경을 설정할 수 있다. 환경 설정에 대한 자세한 내용은 “CUBRID 시작”의 **설치와 실행**을 참조한다.

## 7. 신규 데이터베이스 로드

**cubrid createdb** 유틸리티를 이용하여 데이터베이스를 생성하고, **cubrid loaddb** 유틸리티를 이용하여 언로드한 데이터를 해당 데이터베이스에 로드한다. 데이터베이스 생성에 대한 자세한 내용은 “관리자 안내서”의 **createdb** 을 참조하고, 데이터베이스 로드에는 **loaddb**를 참조한다.

## 8. 신규 데이터베이스 백업

신규 데이터베이스에 데이터 로딩이 완료되면, **cubrid backupdb** 유틸리티를 이용하여 신규 버전의 CUBRID 환경에서 생성된 데이터베이스를 백업한다. 그 이유는 기존 버전의 CUBRID 환경에서 백업한 데이터를 신규 버전의 CUBRID 환경에서 복구할 수 없기 때문이다. 데이터베이스 백업에 대한 자세한 내용은 **backupdb**를 참조한다.

### ⚠ 경고

같은 버전이라 하더라도 백업 및 복구 시 32비트 데이터베이스 볼륨과 64비트 데이터베이스 볼륨 간에는 상호 호환을 보장하지 않는다. 따라서 32비트 CUBRID에서 백업한 데이터베이스를 64비트 CUBRID에서 복구하거나, 이와 반대로 작업하는 것을 권장하지 않는다.

### ⚠ 경고

CUBRID 11버전에서 TDE 기능을 사용할 경우 하위호환성을 제공하지 않아, 언로드된 파일로 더 낮은 버전에서 로드할 수 없다.

## spacedb

**cubrid spacedb** 유틸리티는 사용 중인 데이터베이스 볼륨의 공간을 확인하기 위해서 사용된다. 이 도구에서는 옵션에 따라 데이터베이스 공간 사용에 대한 간략한 집계 정보나 사용 중인 모든 볼륨

및 파일에 대한 자세한 설명을 볼 수 있다. **cubrid spacedb** 유틸리티에서 반환되는 정보는 볼륨의 ID, 각 볼륨의 이름, 용도, 저장 공간의 총 합계 및 남은 저장 공간 등이다.

```
cubrid spacedb [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **spacedb**: 대상 데이터베이스에 대한 공간을 확인하는 명령으로 데이터베이스 서버가 구동 정지 상태인 경우에만 정상적으로 수행된다.
- **database\_name**: 공간을 확인하고자 하는 데이터베이스의 이름이며, 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않는다.

다음은 **cubrid spacedb** 에 대한 [options]이다.

```
-o, --output-file=FILE 파일로 출력을 재지정
-S, --SA-mode 독립 모드 실행
-C, --CS-mode 클라이언트 서버 모드 실행
 --size-unit={PAGE|M|G|T|H} 출력 단위(page, mega, giga, tera, human
readable) 설정
-s, --summarize 볼륨 용도별 합계 결과 출력
-p, --purpose 각 볼륨의 목적별 (데이터/인덱스/임시) 사용량을
출력
```

#### **-o FILE**

데이터베이스의 공간 정보에 대한 결과를 지정한 파일에 저장한다.

```
cubrid spacedb -o db_output testdb
```

#### **-S, --SA-mode**

서버 프로세스를 구동하지 않고 데이터베이스에 접근하는 독립 모드(standalone)로 작업하기 위해 지정되며, 인수는 없다. **-S** 옵션을 지정하지 않으면, 시스템은 클라이언트/서버 모드로 인식한다.

```
cubrid spacedb --SA-mode testdb
```

#### **-C, --CS-mode**

**-C** 옵션은 서버 프로세스와 클라이언트 프로세스를 각각 구동하여 데이터베이스에 접근하는 클라이언트/서버 모드로 작업하기 위한 옵션이며, 인수는 없다. **-C** 옵션을 지정하지 않더라도 시스템은 기본적으로 클라이언트/서버 모드로 인식한다.

```
cubrid spacedb --CS-mode testdb
```

**--size-unit={PAGE|M|G|T|H}**

데이터베이스 볼륨의 공간을 지정한 크기 단위로 출력하기 위한 옵션이며, 기본값은 **H** 이다. 값을 H로 설정할 경우 단위는 다음과 같이 자동으로 지정된다. DB size < 1024MB 보다 작을 경우 M는 MB 으로 , DB size < 1024GB 보다 작을 경우 G는 GB 로 지정된다.

```
$ cubrid spacedb --size-unit=H testdb

Space description for database 'testdb' with pagesize 16.0K.
(log pagesize: 16.0K)

type purpose volume_count
used_size free_size total_size
PERMANENT PERMANENT DATA 2
61.0 M 963.0 M 1.0 G
PERMANENT TEMPORARY DATA 1
12.0 M 500.0 M 512.0 M
TEMPORARY TEMPORARY DATA 1
40.0 M 88.0 M 128.0 M
- - 4
113.0 M 1.5 G 1.6 G

Space description for all volumes:
volid type purpose
used_size free_size total_size
volume_name
 0 PERMANENT PERMANENT DATA
60.0 M 452.0 M 512.0 M /home1/
cubrid/testdb
 1 PERMANENT PERMANENT DATA
1.0 M 511.0 M 512.0 M /home1/
cubrid/testdb_x001
 2 PERMANENT TEMPORARY DATA
12.0 M 500.0 M 512.0 M /home1/
cubrid/testdb_x002
32766 TEMPORARY TEMPORARY DATA
40.0 M 88.0 M 128.0 M /home1/
cubrid/testdb_t32766

LOB space description file:/home1/cubrid/lob
```

**-s, --summarize**

볼륨 타입 및 용도별로 볼륨 수, 사용된 크기, 남은 저장 공간 크기 및 총 저장 공간 크기를 집계한다. 볼륨에는 영구적 데이터를 사용하는 영구적 볼륨, 일시적 데이터를 사용하는 영구적 볼륨과 일시적 데이터를 사용하는 일시적 볼륨, 이 세 가지 종류가 있으며 영구적 데이터를 사용하는 일시적 볼륨은 없다. 마지막 행은 모든 볼륨 타입의 총 공간 값을 출력한다.

```
$ cubrid spacedb -s testdb
```

```
Space description for database 'testdb' with pagesize 16.0K.
(log pagesize: 16.0K)
```

| type      | purpose        | volume_count |
|-----------|----------------|--------------|
| used_size | free_size      | total_size   |
| PERMANENT | PERMANENT DATA | 2            |
| 61.0 M    | 963.0 M        | 1.0 G        |
| PERMANENT | TEMPORARY DATA | 1            |
| 12.0 M    | 500.0 M        | 512.0 M      |
| TEMPORARY | TEMPORARY DATA | 1            |
| 40.0 M    | 88.0 M         | 128.0 M      |
| -         | -              | 4            |
| 113.0 M   | 1.5 G          | 1.6 G        |

### -p, --purpose

저장된 데이터의 용도에 대한 자세한 정보를 출력한다. 이 정보에는 파일 수, 사용된 크기, 파일 테이블 크기, 예약된 섹터 크기 및 총 크기가 포함된다.

```
Space description for database 'testdb' with pagesize 16.0K.
(log pagesize: 16.0K)
```

```
Detailed space description for files:
```

| data_type       | file_count    | used_size  |
|-----------------|---------------|------------|
| file_table_size | reserved_size | total_size |
| INDEX           | 17            | 0.3        |
| M               | 0.3 M         | 16.5 M     |
| HEAP            | 28            | 7.6        |
| M               | 0.4 M         | 26.0 M     |
| SYSTEM          | 8             | 0.4        |
| M               | 0.1 M         | 7.5 M      |
| TEMP            | 10            | 0.0        |
| M               | 0.2 M         | 49.8 M     |
| -               | 63            | 8.2        |
| M               | 1.0 M         | 99.8 M     |

## compactdb

**cubrid compactdb** 유틸리티는 데이터베이스 볼륨 중에 사용되지 않는 공간을 확보하기 위해서 사용된다. 데이터베이스 서버가 정지된 경우(offline)에는 독립 모드(stand-alone mode)로, 데이터베이스가 구동 중인 경우(online)에는 클라이언트 서버 모드(client-server mode)로 공간 정리 작업을 수행할 수 있다.

### 참고

**cubrid compactdb** 유틸리티는 삭제된 객체들의 OID와 클래스 변경에 의해 점유되고 있는 공간을 확보한다. 객체를 삭제하면 삭제된 객체를 참조하는 다른 객체가 있을 수 있기 때문에 삭제된 객체에 대한 OID는 바로 사용 가능한 빈 공간이 될 수 없다.

따라서 OID 재사용을 위해 테이블 생성 시에는 아래 예와 같이 REUSE\_OID 옵션을 사용할 것을 권장한다.

```
CREATE TABLE tbl REUSE_OID
(
 id INT PRIMARY KEY,
 b VARCHAR
);
```

단, REUSE\_OID 옵션을 사용한 테이블은 다른 테이블이 참조할 수 없다. 즉, 다른 테이블의 타입으로 사용될 수 없다.

```
CREATE TABLE reuse_tbl (a INT PRIMARY KEY) REUSE_OID;
CREATE TABLE tbl_1 (a reuse_tbl);
```

```
ERROR: The class 'reuse_tbl' is marked as REUSE_OID and is non-referable. Non-referable classes can't be the domain of an attribute and their instances' OIDs cannot be returned.
```

REUSE\_OID에 대한 자세한 설명은 [REUSE\\_OID](#)를 참고한다.

**cubrid compactdb** 유틸리티를 수행하면 삭제된 객체에 대한 참조를 **NULL**로 표시하는데, 이렇게 **NULL**로 표시된 공간은 OID가 재사용할 수 있는 공간임을 의미한다.

```
cubrid compactdb [options] database_name [schema_name.class_name [{,
schema_name.class_name}]]
```

- **cubrid**: 큐브리드 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **compactdb**: 대상 데이터베이스에 대하여 삭제된 데이터에 할당되었던 OID가 재사용될 수 있도록 공간을 정리하는 명령입니다.
- *database\_name*: 공간을 정리할 데이터베이스의 이름이며, 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않는다.
- *class\_name\_list*: 공간을 정리할 테이블 이름 리스트를 데이터베이스 이름 뒤에 직접 명시할 수 있으며, **-i** 옵션과 함께 사용할 수 없다. 테이블 이름에는 스키마 이름이 접두사로 붙어야 한다. 클라



이언트/서버 모드에서 사용한 경우 catalog, delete files 그리고 tracker와 같은 객체에 대한 정리 작업을 수행하지 않는다.

클라이언트/서버 모드에서만 **-l**, **-c**, **-d**, **-p** 옵션을 사용할 수 있다.

다음은 **cubrid compactdb**에 대한 [options]이다.

|                                          |                         |
|------------------------------------------|-------------------------|
| <b>-v, --verbose</b>                     | 많은 상태 정보 출력             |
| <b>-S, --SA-mode</b>                     | 독립 모드 실행                |
| <b>-C, --CS-mode</b>                     | 클라이언트 서버 모드 실행          |
| <b>-i, --input-class-file=FILE</b>       | 테이블 이름을 포함하는 FILE 이름    |
| <b>-p, --pages-committed-once=NUMBER</b> | 한 번에 커밋할 수 있는 최대 페이지 개수 |
| <b>-d, --delete-old-repr</b>             | 카탈로그에서 과거 테이블 표현을 삭제    |
| <b>-I, --Instance-lock-timeout</b>       | 인스턴스 잠금에 대한 타임아웃        |
| <b>-c, --class-lock-timeout</b>          | 클래스 잠금에 대한 타임아웃         |

#### **-v, --verbose**

어느 클래스가 현재 정리되고 있는지, 얼마나 많은 인스턴스가 그 클래스를 위하여 처리되었는지를 알리는 메시지를 화면에 출력할 수 있다.

```
cubrid compactdb -v testdb
```

#### **-S, --SA-mode**

데이터베이스 서버가 구동 중단된 상태에서 독립 모드(standalone)로 공간 정리 작업을 수행하기 위해 지정되며, 인수는 없다. **-S** 옵션을 지정하지 않으면, 시스템은 클라이언트/서버 모드로 인식한다.

```
cubrid compactdb --SA-mode testdb
```

#### **-C, --CS-mode**

**-C** 옵션은 데이터베이스 서버가 구동 중인 상태에서 클라이언트/서버 모드로 공간 정리 작업을 수행하기 위해 지정되며, 인수는 없다. **-C** 옵션이 생략되더라도 시스템은 기본적으로 클라이언트/서버 모드로 인식한다.

#### **-i, --input-class-file=FILE**

대상 테이블 이름을 포함하는 입력 파일 이름을 지정할 수 있다. 라인 당 하나의 테이블 이름을 명시하며, 유효하지 않은 테이블 이름은 무시된다. 테이블 이름에는 스키마 이름이 접두사로 붙어야 한다. 이 옵션을 지정하는 경우, 데이터베이스 이름 뒤에 대상 테이블 이름 리스트를 직접 명시할 수 없으므로 주의한다. 클라이언트/서버 모드에서 사용한 경우 catalog, delete files 그리고 tracker와 같은 객체에 대한 정리 작업을 수행하지 않는다.

다음은 클라이언트/서버 모드에서만 사용할 수 있는 옵션이다.

#### **-p, --pages-committed-once=NUMBER**

한 번에 커밋할 수 있는 최대 페이지 수를 지정한다. 기본값은 **10** 이며, 최소 값은 1, 최대 값은 10이다. 옵션 값이 작으면 클래스/인스턴스에 대한 잠금 비용이 작으므로 동시성은 향상될 수 있으나 작업 속도는 저하될 수 있고, 옵션 값이 크면 동시성은 저하되나 작업 속도는 향상될 수 있다.

```
cubrid compactdb --CS-mode -p 10 testdb public.tbl1,
public.tbl2, public.tbl5
```

#### **-d, --delete-old-repr**

카탈로그에서 과거 테이블 표현(스키마 구조)을 삭제할 수 있다. **ALTER** 문에 의해 칼럼이 추가되거나 삭제되는 경우 기존의 레코드에 대해 과거의 스키마를 참조하고 있는 상태로 두면, 스키마를 업데이트하는 비용을 들이지 않기 때문에 평소에는 과거의 테이블 표현을 유지하는 것이 좋다.

#### **-l, --Instance-lock-timeout=NUMBER**

인스턴스 잠금 타임아웃 값을 지정할 수 있다. 기본값은 **2** (초)이며, 최소 값은 1, 최대 값은 10이다. 설정된 시간동안 잠금 인스턴스를 대기하므로, 옵션 값이 작을수록 작업 속도는 향상될 수 있으나 처리 가능한 인스턴스 개수가 적어진다. 반면, 옵션 값이 클수록 작업 속도는 저하되나 더 많은 인스턴스에 대해 작업을 수행할 수 있다.

#### **-c, --class-lock-timeout=NUMBER**

클래스 잠금 타임아웃 값을 지정할 수 있다. 기본값은 **10** (초)이며, 최소값은 1, 최대 값은 10이다. 설정된 시간동안 잠금 테이블을 대기하므로, 옵션 값이 작을수록 작업 속도는 향상될 수 있으나 처리 가능한 테이블 개수가 적어진다. 반면, 옵션 값이 클수록 작업 속도는 저하되나 더 많은 테이블에 대해 작업을 수행할 수 있다.

## optimizedb

CUBRID의 질의 최적화기가 사용하는 테이블에 있는 객체들의 수, 접근하는 페이지들의 수, 속성 값들의 분산 같은 통계 정보를 갱신한다.

```
cubrid optimizedb [<option>] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **optimizedb**: 대상 데이터베이스에 대하여 비용 기반 질의 최적화에 사용되는 통계 정보를 업데이트한다. 옵션을 지정하는 경우, 지정한 클래스에 대해서만 업데이트한다.
- *database\_name*: 비용기반 질의 최적화용 통계 자료를 업데이트하려는 데이터베이스 이름이다.

다음은 *cubrid optimizedb*에 대한 <option>이다.

#### **-n, --class-name**

**-n** 옵션을 이용하여 해당 클래스의 질의 통계 정보를 업데이트하는 명령이다. 테이블 이름에는 스키마 이름이 접두사로 붙어야 한다.

```
cubrid optimizedb -n public.event_table testdb
```

다음은 대상 데이터베이스의 전체 클래스의 질의 통계 정보를 업데이트하는 명령이다.

```
cubrid optimizedb testdb
```

## plandump

**cubrid plandump** 유틸리티를 사용해서 서버에 저장(캐시)되어 있는 질의 수행 계획들의 정보를 출력할 수 있다.

```
cubrid plandump [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **plandump**: 대상 데이터베이스에 대하여 현재 캐시에 저장되어 있는 질의 수행 계획을 출력하는 명령이다.
- *database\_name*: 데이터베이스 서버 캐시로부터 질의 수행 계획을 확인 또는 제거하고자 하는 데이터베이스 이름이다

다음은 **cubrid plandump** 에 대한 [options]이다.

|                               |                             |
|-------------------------------|-----------------------------|
| <b>-d, --drop</b>             | 서버 캐시에 있는 모든 플랜 삭제          |
| <b>-o, --output-file=FILE</b> | 출력 메시지를 재지정할 파일             |
| <b>-s, --sha1=SHA1</b>        | 서버 캐시에 있는 SHA1 코드의 특정 플랜 삭제 |

### **-d, --drop**

캐시에 저장된 질의 수행 계획을 제거한다.

```
cubrid plandump -d testdb
```

### **-o, --output-file=FILE**

캐시에 저장된 질의 수행 계획 결과 파일에 저장

```
cubrid plandump -o output.txt testdb
```

**-s, --sha1=SHA1**

캐시에 저장된 SHA1 코드의 특정 질의 수행 계획을 제거한다.

```
$ cubrid plandump testdb

Entries:

 XASL_ID = {
 sha1 = { da9cab5d 597f2fe4 48aa8ae8 66b87d81
49fdd7ca },
 time_stored = 1718614753 sec, 257499 usec
 }
 sql info:
 SQL_ID = 4dfa6d901e048
 sql user text = select * from game where host_year > '2004'
 sql plan text = Sequential scan(public.game dba.game)

cubrid plandump -s 'da9cab5d 597f2fe4 48aa8ae8 66b87d81
49fdd7ca' demodb
```

옵션 없이 사용하면 플랜 캐시(plan cache)에 저장된 질의 실행 계획(XASL)들을 확인할 수 있다.

```
cubrid plandump testdb
```

```
XASL cache
Stats:
Max size: 1000 -- plan cache에 저장할 수 있는
최대 항목 수
Current entry count: 4 -- 현재 저장되어 있는 항목 수
Lookups: 25 -- LOOKUP한 횟수
Hits: 15 -- HIT한 횟수
Miss: 10 -- 실패한 횟수. 이는 대체로 NUM
과 비슷함.
Inserts: 5
Found at insert: 0
Recompiles: 0
Failed recompiles: 0
Deletes: 1
Fix: 15
Unfix: 20
Cache cleanups: 0
Deletes at cleanup: 0

Entries:

XASL_ID = {
 sha1 = { 939db0ab 7ead8a87 2a1ed142 67e2f059 36a3e601 },
 time_stored = 1750657813 sec, 784232 usec
```

```

 }
fix_count = 0
cache flags = 00000000
reference count = 4
time second last used = 1750657814
clone count = 1
sql info:
 SQL_ID = 67d47c14f637a
 sql user text = select * from game where host_year > '2004'
 sql hash text = select [dba.game].[host_year], [dba.game].[
[event_code], [dba.game].[athlete_code], [dba.game].[stadium_code],
[dba.game].[nation_code], [dba.game].[medal], [dba.game].[game_date]
from [dba.game] [dba.game] w here ([dba.game].[host_year]> ? :0)?
193="en_US";194="en_US";249="Asia/Seoul";user=0|897|1;bind_var_cnt=1
 sql plan text =
Sequential scan(public.game dba.game)
OID_LIST (count = 1):
 OID = 0|208|10, LOCK = IS_LOCK, TCARD = 31

```

CUBRID는 플랜 캐시 기능을 통해 SQL 질의의 실행 계획을 재사용할 수 있도록 지원한다. 플랜 캐시는 서버에서 관리되며, CAS(Common Application Server)와 서버 간의 협력을 통해 동작한다. 주요 처리 과정은 다음과 같다.

1. 질의분석(파싱 및 재작성) : 클라이언트에서 전달된 SQL 질의는 CAS에서 먼저 파싱(parsing)과 재작성(rewrite) 과정을 거친다.
2. XASL\_ID 생성 : 재작성된 질의를 기반으로 XASL\_ID가 생성된다. 이 XASL\_ID는 질의의 고유한 식별자로 사용되며, 서버의 플랜 캐시에 해당 질의가 이미 저장되어 있는지 확인하는 데 사용된다.
3. 플랜 캐시 확인생성된 XASL\_ID를 사용하여 플랜 캐시에 이미 저장된 실행 계획이 있는지 확인한다.
  - 저장되어 있는 경우: 서버는 플랜 캐시에 저장된 실행 계획을 이용해 질의를 실행
  - 저장되어 있지 않은 경우: CAS에서 새로운 실행 계획을 생성하여 서버로 전달

#### 참고

플랜 캐시는 실행 계획의 재 사용을 통해 성능을 향상시킨다.

plandump 결과의 상세항목 은 다음과 같다.

- XASL\_IDsql hash text를 SHA1으로 변환한 문자열로 플랜 캐시의 키(key)로 사용
  - time\_stored : 플랜 캐시에 저장된 시간(unix timestamp)

- 예제) `$> date -d @1750657813.784232 (Mon Jun 23 14:50:13 KST 2025)`
- `fix_count` : 동시 사용하는 스레드 수
- **cache flags** 캐시된 실행 계획의 상태정보
  - 8000000 : mark\_deleted
  - 4000000 : to be recompiled
  - 2000000 : was recompiled
  - 0800000 : clean up
  - 0400000 : recomiled requested
- `reference count` : 참조된 횟수
- **time second last used** 최종 사용된 시간(unix timestamp)
  - 예제) `$> date -d @1750657814 (Mon Jun 23 14:50:14 KST 2025)`
- `clone count` : 사용된 클론(clone) 개수 (서버에는 실행계획이 메모리 스트림 형태로 저장되어 있으며, 실제 실행 시에는 이를 XASL 구조체로 변환한다. 이 변환 비용을 줄이기 위해 클론 캐시(clone cache)를 사용하게된다. 클론 캐시는 변환된 XASL 구조체를 보관하여 재 사용하며, 동시에 여러 스레드가 요청할 경우 사용할 클론이 없으면 새로 생성하여 사용 후 클론 캐시에 등록한다. 이때 해당값이 증가 한다.)
- **SQL\_ID** 원문 질의를 MD5 변환하여 32자리(16진수) 문자열 중 마지막 13자리의 16진수 값(hexa-digit) 을 SQL ID로 사용한다.
  - MD5결과값이 'e1faffb3e614e6c2fba74296962386b7' 인 경우, SQL\_ID는 **74296962386b7** 이다.
- `sql user text` : 원문 질의문
- `sql hash text` : 재작성된 질의문
- `sql plan text` : 실행계획
- `OID_LIST` : 참조하고 있는 객체(table, serial 등)의 OID 정보

#### 참고

- unix timestamp는 date명령어를 사용하여 사람이 이해할 수 있는 형식의 시간으로 변환하여 확인할 수 있다.

## statdump

**cubrid statdump** 유틸리티를 이용해 CUBRID 데이터베이스 서버가 실행한 통계 정보를 확인할 수 있으며, 통계 정보 항목은 크게 File I/O 관련, 페이지 버퍼 관련, 로그 관련, 트랜잭션 관련, 동시성 관련, 인덱스 관련, 쿼리 수행 관련, 네트워크 요청 관련으로 구분된다.

CSQL의 해당 연결에 대해서만 통계 정보를 확인하려면 CSQL의 세션 명령어를 이용할 수 있으며 **CSQL 실행 통계 정보 출력** 를 참고한다.

```
cubrid statdump [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **statdump**: 데이터베이스 서버 수행 시 통계 정보를 출력하는 명령어이다.
- *database\_name*: 통계 자료를 확인하고자 하는 대상 데이터베이스 이름이다.

다음은 **cubrid statdump** 에 대한 [options]이다.

```
-i, --interval=S 매 S 초마다 통계를 출력
-o, --output-file=FILE 출력 메시지를 재지정할 파일
-c, --cumulative 누적값을 출력
-s, --substr=STRING 지정한 문자열이 포함된 항목만 출력
```

### **-i, --interval=SECOND**

지정한 초 단위로 주기적으로 출력한다. **-i** 옵션이 주어질 때만 정보가 갱신된다.

다음은 1초마다 누적된 정보 값을 출력한다.

```
cubrid statdump -i 1 -c demodb
```

다음은 1초 마다 0으로 리셋하고 1초 동안 누적된 값을 출력한다.

```
cubrid statdump -i 1 demodb
```

다음은 **-i** 옵션으로 가장 마지막에 실행한 값을 출력한다.

```
cubrid statdump demodb
```

다음은 위와 같은 결과를 출력한다. **-c** 옵션은 **-i** 옵션과 같이 쓰이지 않으면 옵션을 설정하지 않은 것과 동일하다.

```
cubrid statdump -c demodb
```

다음은 5초마다 결과를 출력한다.

```
$ cubrid statdump -i 5 -c testdb

Mon November 11 23:44:36 KST 2019

*** SERVER EXECUTION STATISTICS ***
Num_file_creates = 0
Num_file_removes = 0
Num_file_ireads = 0
Num_file_iowrites = 3
Num_file_iosynches = 3
The timer values for file_iosync_all are:
Num_file_iosync_all = 0
Total_time_file_iosync_all = 0
Max_time_file_iosync_all = 0
Avg_time_file_iosync_all = 0
Num_file_page_allocs = 0
Num_file_page_deallocs = 0
Num_data_page_fetches = 0
Num_data_page_dirties = 0
Num_data_page_ireads = 0
Num_data_page_iowrites = 0
Num_data_page_flushed = 0
Num_data_page_private_quota = 11327
Num_data_page_private_count = 0
Num_data_page_fixed = 1
Num_data_page_dirty = 0
Num_data_page_lru1 = 18
Num_data_page_lru2 = 10
Num_data_page_lru3 = 0
Num_data_page_victim_candidate = 0
Num_log_page_fetches = 0
Num_log_page_ireads = 0
Num_log_page_iowrites = 6
Num_log_append_records = 9
Num_log_archives = 0
Num_log_start_checkpoints = 0
Num_log_end_checkpoints = 0
Num_log_wals = 0
Num_log_page_iowrites_for_replacement = 0
Num_log_page_replacements = 0
Num_page_locks_acquired = 0
Num_object_locks_acquired = 0
Num_page_locks_converted = 0
Num_object_locks_converted = 0
Num_page_locks_re-requested = 0
Num_object_locks_re-requested = 0
```



```

Num_page_locks_waits = 0
Num_object_locks_waits = 0
Num_object_locks_time_waited_usec = 0
Num_tran_commits = 0
Num_tran_rollback = 0
Num_tran_savepoints = 0
Num_tran_start_topops = 0
Num_tran_end_topops = 0
Num_tran_interrupts = 0
Num_btree_inserts = 0
Num_btree_deletes = 0
Num_btree_updates = 0
Num_btree_covered = 0
Num_btree_noncovered = 0
Num_btree_resumes = 0
Num_btree_multirange_optimization = 0
Num_btree_splits = 0
Num_btree_merges = 0
Num_btree_get_stats = 0
Num_btree_online_inserts = 0
Num_btree_online_inserts_same_page_hold = 0
Num_btree_online_inserts_reject_no_more_keys = 0
Num_btree_online_inserts_reject_max_key_len = 0
Num_btree_online_inserts_reject_no_space = 0
Num_btree_online_release_latch = 0
Num_btree_online_inserts_reject_key_not_in_range1 = 0
Num_btree_online_inserts_reject_key_not_in_range2 = 0
Num_btree_online_inserts_reject_key_not_in_range3 = 0
Num_btree_online_inserts_reject_key_not_in_range4 = 0
Num_btree_online_inserts_reject_key_false_failed_range1
= 0
Num_btree_online_inserts_reject_key_false_failed_range2
= 0
The timer values for btree_online are:
Num_btree_online = 0
Total_time_btree_online = 0
Max_time_btree_online = 0
Avg_time_btree_online = 0
The timer values for btree_online_insert_task are:
Num_btree_online_insert_task = 0
Total_time_btree_online_insert_task = 0
Max_time_btree_online_insert_task = 0
Avg_time_btree_online_insert_task = 0
The timer values for btree_online_prepare_task are:
Num_btree_online_prepare_task = 0
Total_time_btree_online_prepare_task = 0
Max_time_btree_online_prepare_task = 0
Avg_time_btree_online_prepare_task = 0
The timer values for btree_online_insert_same_leaf are:
Num_btree_online_insert_same_leaf = 0
Total_time_btree_online_insert_same_leaf = 0
Max_time_btree_online_insert_same_leaf = 0

```

```

Avg_time_btree_online_insert_same_leaf = 0
Num_query_selects = 0
Num_query_inserts = 0
Num_query_deletes = 0
Num_query_updates = 0
Num_query_sscans = 0
Num_query_iscans = 0
Num_query_lscans = 0
Num_query_setscans = 0
Num_query_methscans = 0
Num_query_nljoins = 0
Num_query_mjoins = 0
Num_query_objfetches = 0
Num_query_holdable_cursors = 0
Num_sort_io_pages = 0
Num_sort_data_pages = 0
Num_network_requests = 3
Num_adaptive_flush_pages = 0
Num_adaptive_flush_log_pages = 0
Num_adaptive_flush_max_pages = 14464
Num_prior_lsa_list_size = 0
Num_prior_lsa_list_maxed = 0
Num_prior_lsa_list_removed = 3
Time_ha_replication_delay = 0
Num_plan_cache_add = 0
Num_plan_cache_lookup = 0
Num_plan_cache_hit = 0
Num_plan_cache_miss = 0
Num_plan_cache_full = 0
Num_plan_cache_delete = 0
Num_plan_cache_invalid_xasl_id = 0
Num_plan_cache_entries = 0
Num_vacuum_log_pages_vacuumed = 0
Num_vacuum_log_pages_to_vacuum = 0
Num_vacuum_prefetch_requests_log_pages = 0
Num_vacuum_prefetch_hits_log_pages = 0
Num_heap_home_inserts = 0
Num_heap_big_inserts = 0
Num_heap_assign_inserts = 0
Num_heap_home_deletes = 0
Num_heap_home_mvcc_deletes = 0
Num_heap_home_to_rel_deletes = 0
Num_heap_home_to_big_deletes = 0
Num_heap_rel_deletes = 0
Num_heap_rel_mvcc_deletes = 0
Num_heap_rel_to_home_deletes = 0
Num_heap_rel_to_big_deletes = 0
Num_heap_rel_to_rel_deletes = 0
Num_heap_big_deletes = 0
Num_heap_big_mvcc_deletes = 0
Num_heap_home_updates = 0
Num_heap_home_to_rel_updates = 0

```

```

Num_heap_home_to_big_updates = 0
Num_heap_rel_updates = 0
Num_heap_rel_to_home_updates = 0
Num_heap_rel_to_rel_updates = 0
Num_heap_rel_to_big_updates = 0
Num_heap_big_updates = 0
Num_heap_home_vacuums = 0
Num_heap_big_vacuums = 0
Num_heap_rel_vacuums = 0
Num_heap_insid_vacuums = 0
Num_heap_remove_vacuums = 0
The timer values for heap_insert_prepare are:
Num_heap_insert_prepare = 0
Total_time_heap_insert_prepare = 0
Max_time_heap_insert_prepare = 0
Avg_time_heap_insert_prepare = 0
The timer values for heap_insert_execute are:
Num_heap_insert_execute = 0
Total_time_heap_insert_execute = 0
Max_time_heap_insert_execute = 0
Avg_time_heap_insert_execute = 0
The timer values for heap_insert_log are:
Num_heap_insert_log = 0
Total_time_heap_insert_log = 0
Max_time_heap_insert_log = 0
Avg_time_heap_insert_log = 0
The timer values for heap_delete_prepare are:
Num_heap_delete_prepare = 0
Total_time_heap_delete_prepare = 0
Max_time_heap_delete_prepare = 0
Avg_time_heap_delete_prepare = 0
The timer values for heap_delete_execute are:
Num_heap_delete_execute = 0
Total_time_heap_delete_execute = 0
Max_time_heap_delete_execute = 0
Avg_time_heap_delete_execute = 0
The timer values for heap_delete_log are:
Num_heap_delete_log = 0
Total_time_heap_delete_log = 0
Max_time_heap_delete_log = 0
Avg_time_heap_delete_log = 0
The timer values for heap_update_prepare are:
Num_heap_update_prepare = 0
Total_time_heap_update_prepare = 0
Max_time_heap_update_prepare = 0
Avg_time_heap_update_prepare = 0
The timer values for heap_update_execute are:
Num_heap_update_execute = 0
Total_time_heap_update_execute = 0
Max_time_heap_update_execute = 0
Avg_time_heap_update_execute = 0
The timer values for heap_update_log are:

```

```

Num_heap_update_log = 0
Total_time_heap_update_log = 0
Max_time_heap_update_log = 0
Avg_time_heap_update_log = 0
The timer values for heap_vacuum_prepare are:
Num_heap_vacuum_prepare = 0
Total_time_heap_vacuum_prepare = 0
Max_time_heap_vacuum_prepare = 0
Avg_time_heap_vacuum_prepare = 0
The timer values for heap_vacuum_execute are:
Num_heap_vacuum_execute = 0
Total_time_heap_vacuum_execute = 0
Max_time_heap_vacuum_execute = 0
Avg_time_heap_vacuum_execute = 0
The timer values for heap_vacuum_log are:
Num_heap_vacuum_log = 0
Total_time_heap_vacuum_log = 0
Max_time_heap_vacuum_log = 0
Avg_time_heap_vacuum_log = 0
The timer values for heap_stats_sync_bestspace are:
Num_heap_stats_sync_bestspace = 0
Total_time_heap_stats_sync_bestspace = 0
Max_time_heap_stats_sync_bestspace = 0
Avg_time_heap_stats_sync_bestspace = 0
Num_heap_stats_bestspace_entries = 0
Num_heap_stats_bestspace_maxed = 0
The timer values for bestspace_add are:
Num_bestspace_add = 0
Total_time_bestspace_add = 0
Max_time_bestspace_add = 0
Avg_time_bestspace_add = 0
The timer values for bestspace_del are:
Num_bestspace_del = 0
Total_time_bestspace_del = 0
Max_time_bestspace_del = 0
Avg_time_bestspace_del = 0
The timer values for bestspace_find are:
Num_bestspace_find = 0
Total_time_bestspace_find = 0
Max_time_bestspace_find = 0
Avg_time_bestspace_find = 0
The timer values for heap_find_page_bestspace are:
Num_heap_find_page_bestspace = 0
Total_time_heap_find_page_bestspace = 0
Max_time_heap_find_page_bestspace = 0
Avg_time_heap_find_page_bestspace = 0
The timer values for heap_find_best_page are:
Num_heap_find_best_page = 0
Total_time_heap_find_best_page = 0
Max_time_heap_find_best_page = 0
Avg_time_heap_find_best_page = 0
The timer values for bt_fix_ovf_oids are:

```

```

Num_bt_fix_ovf_oids = 0
Total_time_bt_fix_ovf_oids = 0
Max_time_bt_fix_ovf_oids = 0
Avg_time_bt_fix_ovf_oids = 0
The timer values for bt_unique_rlocks are:
Num_bt_unique_rlocks = 0
Total_time_bt_unique_rlocks = 0
Max_time_bt_unique_rlocks = 0
Avg_time_bt_unique_rlocks = 0
The timer values for bt_unique_wlocks are:
Num_bt_unique_wlocks = 0
Total_time_bt_unique_wlocks = 0
Max_time_bt_unique_wlocks = 0
Avg_time_bt_unique_wlocks = 0
The timer values for bt_leaf are:
Num_bt_leaf = 0
Total_time_bt_leaf = 0
Max_time_bt_leaf = 0
Avg_time_bt_leaf = 0
The timer values for bt_traverse are:
Num_bt_traverse = 0
Total_time_bt_traverse = 0
Max_time_bt_traverse = 0
Avg_time_bt_traverse = 0
The timer values for bt_find_unique are:
Num_bt_find_unique = 0
Total_time_bt_find_unique = 0
Max_time_bt_find_unique = 0
Avg_time_bt_find_unique = 0
The timer values for bt_find_unique_traverse are:
Num_bt_find_unique_traverse = 0
Total_time_bt_find_unique_traverse = 0
Max_time_bt_find_unique_traverse = 0
Avg_time_bt_find_unique_traverse = 0
The timer values for bt_range_search are:
Num_bt_range_search = 0
Total_time_bt_range_search = 0
Max_time_bt_range_search = 0
Avg_time_bt_range_search = 0
The timer values for bt_range_search_traverse are:
Num_bt_range_search_traverse = 0
Total_time_bt_range_search_traverse = 0
Max_time_bt_range_search_traverse = 0
Avg_time_bt_range_search_traverse = 0
The timer values for bt_insert are:
Num_bt_insert = 0
Total_time_bt_insert = 0
Max_time_bt_insert = 0
Avg_time_bt_insert = 0
The timer values for bt_insert_traverse are:
Num_bt_insert_traverse = 0
Total_time_bt_insert_traverse = 0

```

```

Max_time_bt_insert_traverse = 0
Avg_time_bt_insert_traverse = 0
The timer values for bt_delete_obj are:
Num_bt_delete_obj = 0
Total_time_bt_delete_obj = 0
Max_time_bt_delete_obj = 0
Avg_time_bt_delete_obj = 0
The timer values for bt_delete_obj_traverse are:
Num_bt_delete_obj_traverse = 0
Total_time_bt_delete_obj_traverse = 0
Max_time_bt_delete_obj_traverse = 0
Avg_time_bt_delete_obj_traverse = 0
The timer values for bt_mvcc_delete are:
Num_bt_mvcc_delete = 0
Total_time_bt_mvcc_delete = 0
Max_time_bt_mvcc_delete = 0
Avg_time_bt_mvcc_delete = 0
The timer values for bt_mvcc_delete_traverse are:
Num_bt_mvcc_delete_traverse = 0
Total_time_bt_mvcc_delete_traverse = 0
Max_time_bt_mvcc_delete_traverse = 0
Avg_time_bt_mvcc_delete_traverse = 0
The timer values for bt_mark_delete are:
Num_bt_mark_delete = 0
Total_time_bt_mark_delete = 0
Max_time_bt_mark_delete = 0
Avg_time_bt_mark_delete = 0
The timer values for bt_mark_delete_traverse are:
Num_bt_mark_delete_traverse = 0
Total_time_bt_mark_delete_traverse = 0
Max_time_bt_mark_delete_traverse = 0
Avg_time_bt_mark_delete_traverse = 0
The timer values for bt_undo_insert are:
Num_bt_undo_insert = 0
Total_time_bt_undo_insert = 0
Max_time_bt_undo_insert = 0
Avg_time_bt_undo_insert = 0
The timer values for bt_undo_insert_traverse are:
Num_bt_undo_insert_traverse = 0
Total_time_bt_undo_insert_traverse = 0
Max_time_bt_undo_insert_traverse = 0
Avg_time_bt_undo_insert_traverse = 0
The timer values for bt_undo_delete are:
Num_bt_undo_delete = 0
Total_time_bt_undo_delete = 0
Max_time_bt_undo_delete = 0
Avg_time_bt_undo_delete = 0
The timer values for bt_undo_delete_traverse are:
Num_bt_undo_delete_traverse = 0
Total_time_bt_undo_delete_traverse = 0
Max_time_bt_undo_delete_traverse = 0
Avg_time_bt_undo_delete_traverse = 0

```

```

The timer values for bt_undo_mvcc_delete are:
Num_bt_undo_mvcc_delete = 0
Total_time_bt_undo_mvcc_delete = 0
Max_time_bt_undo_mvcc_delete = 0
Avg_time_bt_undo_mvcc_delete = 0
The timer values for bt_undo_mvcc_delete_traverse are:
Num_bt_undo_mvcc_delete_traverse = 0
Total_time_bt_undo_mvcc_delete_traverse = 0
Max_time_bt_undo_mvcc_delete_traverse = 0
Avg_time_bt_undo_mvcc_delete_traverse = 0
The timer values for bt_vacuum are:
Num_bt_vacuum = 0
Total_time_bt_vacuum = 0
Max_time_bt_vacuum = 0
Avg_time_bt_vacuum = 0
The timer values for bt_vacuum_traverse are:
Num_bt_vacuum_traverse = 0
Total_time_bt_vacuum_traverse = 0
Max_time_bt_vacuum_traverse = 0
Avg_time_bt_vacuum_traverse = 0
The timer values for bt_vacuum_insid are:
Num_bt_vacuum_insid = 0
Total_time_bt_vacuum_insid = 0
Max_time_bt_vacuum_insid = 0
Avg_time_bt_vacuum_insid = 0
The timer values for bt_vacuum_insid_traverse are:
Num_bt_vacuum_insid_traverse = 0
Total_time_bt_vacuum_insid_traverse = 0
Max_time_bt_vacuum_insid_traverse = 0
Avg_time_bt_vacuum_insid_traverse = 0
The timer values for vacuum_master are:
Num_vacuum_master = 561
Total_time_vacuum_master = 1259
Max_time_vacuum_master = 6
Avg_time_vacuum_master = 2
The timer values for vacuum_job are:
Num_vacuum_job = 0
Total_time_vacuum_job = 0
Max_time_vacuum_job = 0
Avg_time_vacuum_job = 0
The timer values for vacuum_worker_process_log are:
Num_vacuum_worker_process_log = 0
Total_time_vacuum_worker_process_log = 0
Max_time_vacuum_worker_process_log = 0
Avg_time_vacuum_worker_process_log = 0
The timer values for vacuum_worker_execute are:
Num_vacuum_worker_execute = 0
Total_time_vacuum_worker_execute = 0
Max_time_vacuum_worker_execute = 0
Avg_time_vacuum_worker_execute = 0
Time_get_snapshot_acquire_time = 0
Count_get_snapshot_retry = 0

```

```

Time_tran_complete_time = 0
The timer values for compute_oldest_visible are:
Num_compute_oldest_visible = 561
Total_time_compute_oldest_visible = 569
Max_time_compute_oldest_visible = 4
Avg_time_compute_oldest_visible = 1
Count_get_oldest_mvcc_retry = 0
Data_page_buffer_hit_ratio = 0.00
Log_page_buffer_hit_ratio = 0.00
Vacuum_data_page_buffer_hit_ratio = 0.00
Vacuum_page_efficiency_ratio = 0.00
Vacuum_page_fetch_ratio = 0.00
Data_page_fix_lock_acquire_time_msec = 0.00
Data_page_fix_hold_acquire_time_msec = 0.00
Data_page_fix_acquire_time_msec = 0.00
Data_page_allocate_time_ratio = 0.00
Data_page_total_promote_success = 0.00
Data_page_total_promote_fail = 0.00
Data_page_total_promote_time_msec = 0.00
Num_unfix_void_to_private_top = 0
Num_unfix_void_to_private_mid = 0
Num_unfix_void_to_shared_mid = 0
Num_unfix_lru1_private_to_shared_mid = 0
Num_unfix_lru2_private_to_shared_mid = 0
Num_unfix_lru3_private_to_shared_mid = 0
Num_unfix_lru2_private_keep = 0
Num_unfix_lru2_shared_keep = 0
Num_unfix_lru2_private_to_top = 0
Num_unfix_lru2_shared_to_top = 0
Num_unfix_lru3_private_to_top = 0
Num_unfix_lru3_shared_to_top = 0
Num_unfix_lru1_private_keep = 0
Num_unfix_lru1_shared_keep = 0
Num_unfix_void_to_private_mid_vacuum = 0
Num_unfix_lru1_any_keep_vacuum = 0
Num_unfix_lru2_any_keep_vacuum = 0
Num_unfix_lru3_any_keep_vacuum = 0
Num_unfix_void_aout_found = 0
Num_unfix_void_aout_not_found = 0
Num_unfix_void_aout_found_vacuum = 0
Num_unfix_void_aout_not_found_vacuum = 0
Num_data_page_hash_anchor_waits = 0
Time_data_page_hash_anchor_wait = 0
The timer values for flush_collect are:
Num_flush_collect = 0
Total_time_flush_collect = 0
Max_time_flush_collect = 0
Avg_time_flush_collect = 0
The timer values for flush_flush are:
Num_flush_flush = 0
Total_time_flush_flush = 0
Max_time_flush_flush = 0

```



```

Avg_time_flush_flush = 0
The timer values for flush_sleep are:
Num_flush_sleep = 4
Total_time_flush_sleep = 4000307
Max_time_flush_sleep = 1000077
Avg_time_flush_sleep = 1000076
The timer values for flush_collect_per_page are:
Num_flush_collect_per_page = 0
Total_time_flush_collect_per_page = 0
Max_time_flush_collect_per_page = 0
Avg_time_flush_collect_per_page = 0
The timer values for flush_flush_per_page are:
Num_flush_flush_per_page = 0
Total_time_flush_flush_per_page = 0
Max_time_flush_flush_per_page = 0
Avg_time_flush_flush_per_page = 0
Num_data_page_writes = 0
Num_data_page_dirty_to_post_flush = 0
Num_data_page_skipped_flush = 0
Num_data_page_skipped_flush_need_wal = 0
Num_data_page_skipped_flush_already_flushed = 0
Num_data_page_skipped_flush_fixed_or_hot = 0
The timer values for compensate_flush are:
Num_compensate_flush = 0
Total_time_compensate_flush = 0
Max_time_compensate_flush = 0
Avg_time_compensate_flush = 0
The timer values for assign_direct_bcb are:
Num_assign_direct_bcb = 0
Total_time_assign_direct_bcb = 0
Max_time_assign_direct_bcb = 0
Avg_time_assign_direct_bcb = 0
The timer values for wake_flush_waiter are:
Num_wake_flush_waiter = 0
Total_time_wake_flush_waiter = 0
Max_time_wake_flush_waiter = 0
Avg_time_wake_flush_waiter = 0
The timer values for alloc_bcb are:
Num_alloc_bcb = 0
Total_time_alloc_bcb = 0
Max_time_alloc_bcb = 0
Avg_time_alloc_bcb = 0
The timer values for alloc_bcb_search_victim are:
Num_alloc_bcb_search_victim = 0
Total_time_alloc_bcb_search_victim = 0
Max_time_alloc_bcb_search_victim = 0
Avg_time_alloc_bcb_search_victim = 0
The timer values for alloc_bcb_cond_wait_high_prio are:
Num_alloc_bcb_cond_wait_high_prio = 0
Total_time_alloc_bcb_cond_wait_high_prio = 0
Max_time_alloc_bcb_cond_wait_high_prio = 0
Avg_time_alloc_bcb_cond_wait_high_prio = 0

```

```

The timer values for alloc_bcb_cond_wait_low_prio are:
Num_alloc_bcb_cond_wait_low_prio = 0
Total_time_alloc_bcb_cond_wait_low_prio = 0
Max_time_alloc_bcb_cond_wait_low_prio = 0
Avg_time_alloc_bcb_cond_wait_low_prio = 0
Num_alloc_bcb_prioritize_vacuum = 0
Num_victim_use_invalid_bcb = 0
The timer values for
alloc_bcb_get_victim_search_own_private_list are:
Num_alloc_bcb_get_victim_search_own_private_list = 0
Total_time_alloc_bcb_get_victim_search_own_private_list
= 0
Max_time_alloc_bcb_get_victim_search_own_private_list =
0
Avg_time_alloc_bcb_get_victim_search_own_private_list =
0
The timer values for
alloc_bcb_get_victim_search_others_private_list are:
Num_alloc_bcb_get_victim_search_others_private_list = 0
Total_time_alloc_bcb_get_victim_search_others_private_list
= 0
Max_time_alloc_bcb_get_victim_search_others_private_list
= 0
Avg_time_alloc_bcb_get_victim_search_others_private_list
= 0
The timer values for alloc_bcb_get_victim_search_shared_list are:
Num_alloc_bcb_get_victim_search_shared_list = 0
Total_time_alloc_bcb_get_victim_search_shared_list = 0
Max_time_alloc_bcb_get_victim_search_shared_list = 0
Avg_time_alloc_bcb_get_victim_search_shared_list = 0
Num_victim_assign_direct_vacuum_void = 0
Num_victim_assign_direct_vacuum_lru = 0
Num_victim_assign_direct_flush = 0
Num_victim_assign_direct_panic = 0
Num_victim_assign_direct_adjust_lru = 0
Num_victim_assign_direct_adjust_lru_to_vacuum = 0
Num_victim_assign_direct_search_for_flush = 0
Num_victim_shared_lru_success = 0
Num_victim_own_private_lru_success = 0
Num_victim_other_private_lru_success = 0
Num_victim_shared_lru_fail = 0
Num_victim_own_private_lru_fail = 0
Num_victim_other_private_lru_fail = 0
Num_victim_all_lru_fail = 0
Num_victim_get_from_lru = 0
Num_victim_get_from_lru_was_empty = 0
Num_victim_get_from_lru_fail = 0
Num_victim_get_from_lru_bad_hint = 0
Num_lfcq_prv_get_total_calls = 0
Num_lfcq_prv_get_empty = 0
Num_lfcq_prv_get_big = 0
Num_lfcq_shr_get_total_calls = 0

```

```

Num_lfcq_shr_get_empty = 0
The timer values for Time_DWB_flush_block_time are:
Num_Time_DWB_flush_block_time = 0
Total_time_Time_DWB_flush_block_time = 0
Max_time_Time_DWB_flush_block_time = 0
Avg_time_Time_DWB_flush_block_time = 0
The timer values for Time_DWB_flush_block_helper_time are:
Num_Time_DWB_flush_block_helper_time = 0
Total_time_Time_DWB_flush_block_helper_time = 0
Max_time_Time_DWB_flush_block_helper_time = 0
Avg_time_Time_DWB_flush_block_helper_time = 0
The timer values for Time_DWB_flush_block_cond_wait_time are:
Num_Time_DWB_flush_block_cond_wait_time = 5352
Total_time_Time_DWB_flush_block_cond_wait_time = 5629578
Max_time_Time_DWB_flush_block_cond_wait_time = 1059
Avg_time_Time_DWB_flush_block_cond_wait_time = 1051
The timer values for Time_DWB_flush_block_sort_time are:
Num_Time_DWB_flush_block_sort_time = 0
Total_time_Time_DWB_flush_block_sort_time = 0
Max_time_Time_DWB_flush_block_sort_time = 0
Avg_time_Time_DWB_flush_block_sort_time = 0
The timer values for Time_DWB_flush_remove_hash_entries are:
Num_Time_DWB_flush_remove_hash_entries = 0
Total_time_Time_DWB_flush_remove_hash_entries = 0
Max_time_Time_DWB_flush_remove_hash_entries = 0
Avg_time_Time_DWB_flush_remove_hash_entries = 0
The timer values for Time_DWB_checksum_time are:
Num_Time_DWB_checksum_time = 0
Total_time_Time_DWB_checksum_time = 0
Max_time_Time_DWB_checksum_time = 0
Avg_time_Time_DWB_checksum_time = 0
The timer values for Time_DWB_wait_flush_block_time are:
Num_Time_DWB_wait_flush_block_time = 0
Total_time_Time_DWB_wait_flush_block_time = 0
Max_time_Time_DWB_wait_flush_block_time = 0
Avg_time_Time_DWB_wait_flush_block_time = 0
The timer values for Time_DWB_wait_flush_block_helper_time are:
Num_Time_DWB_wait_flush_block_helper_time = 0
Total_time_Time_DWB_wait_flush_block_helper_time = 0
Max_time_Time_DWB_wait_flush_block_helper_time = 0
Avg_time_Time_DWB_wait_flush_block_helper_time = 0
The timer values for Time_DWB_flush_force_time are:
Num_Time_DWB_flush_force_time = 0
Total_time_Time_DWB_flush_force_time = 0
Max_time_Time_DWB_flush_force_time = 0
Avg_time_Time_DWB_flush_force_time = 0
Num_alloc_bcb_wait_threads_high_priority = 0
Num_alloc_bcb_wait_threads_low_priority = 0
Num_flushed_bcbs_wait_for_direct_victim = 0
Num_lfcq_big_private_lists = 0
Num_lfcq_private_lists = 0
Num_lfcq_shared_lists = 0

```

```

Num_data_page_avoid_dealloc = 0
Num_data_page_avoid_victim = 0
Num_data_page_fix_ext:
Num_data_page_promote_ext:
Num_data_page_promote_time_ext:
Num_data_page_unfix_ext:
Time_data_page_lock_acquire_time:
Time_data_page_hold_acquire_time:
Time_data_page_fix_acquire_time:
Num_mvcc_snapshot_ext:
Time_obj_lock_acquire_time:
Thread_stats_counters_timers:
Thread_pgbuf_daemon_stats_counters_timers:
Num_dwb_flushed_block_volumes:
Thread_loaddb_stats_counters_timers:

```

다음은 위의 통계 정보에 대한 설명이다. 통계 카테고리 (데이터베이스 모듈), 이름, 통계 유형 및 각 통계에 대한 간략한 설명을 볼 수 있다.

수집 방법에 따라 몇 가지 통계 유형이 있다.:

- Accumulator: 트래킹이 발생할 때마다 통계치가 누적된다.
- Counter/timer: 동작(action)별 횟수와 지속 시간에 대한 통계 뿐만 아니라 최대 및 평균 지속 시간에 대한 통계치를 산출한다.
- Snapshot: 통계는 데이터베이스의 내부 정보이다.
- Complex: 통계는 다양한 속성별로 산출된 복합 통계치이다.

대부분 통계치는 누적이 되는 Accumulator 유형이다. 이 외에도 동작 횟수와 동작 지속 시간이 반영된 Counter/timer 유형, 데이터베이스에서 수집된 Snapshot 유형 및 다른 값을 기반으로 계산된 Computed 유형과 일부 동작에 대한 자세한 정보를 반영하는 복합(Complex) 유형이다.

## File I/O

| 항목               | 통계 타입       | 설명           |
|------------------|-------------|--------------|
| Num_file_removes | Accumulator | 삭제한 파일 개수    |
| Num_file_creates | Accumulator | 생성한 파일 개수    |
| Num_file_ireads  | Accumulator | 디스크로부터 읽을 횟수 |

|                        |               |                    |
|------------------------|---------------|--------------------|
| Num_file_iowrites      | Accumulator   | 디스크로 저장한 횟수        |
| Num_file_iosynches     | Accumulator   | 디스크와 동기화를 수행한 횟수   |
| ..file_iosync_all      | Counter/timer | 모든 파일을 동기화한 횟수와 시간 |
| Num_file_page_allocs   | Accumulator   | 할당한 페이지 개수         |
| Num_file_page_deallocs | Accumulator   | 반환한 페이지 개수         |

## 페이지 버퍼

| 항목                          | 통계 타입       | 설명                                                   |
|-----------------------------|-------------|------------------------------------------------------|
| Num_data_page_fetches       | Accumulator | 가져오기(fetch)한 페이지 개수                                  |
| Num_data_page_dirties       | Accumulator | 더티 페이지 개수                                            |
| Num_data_page_ioreads       | Accumulator | 디스크에서 읽은 페이지 수<br>(이 값이 클수록 덜 효율적이며, 히트율이 낮은 것과 상관됨) |
| Num_data_page_iowrites      | Accumulator | 디스크에 기록한 페이지 수<br>(이 값이 클수록 덜 효율적임)                  |
| Num_data_page_private_quota | Snapshot    | 전용 LRU 리스트의 대상 페이지 개수                                |

|                                |          |                              |
|--------------------------------|----------|------------------------------|
| Num_data_page_private_count    | Snapshot | 전용 LRU 리스트에 대한 실제 페이지 개수     |
| Num_data_page_fixed            | Snapshot | 데이터 버퍼의 고정 페이지 개수            |
| Num_data_page_dirty            | Snapshot | 데이터 버퍼의 더티 페이지 개수            |
| Num_data_page_lru1             | Snapshot | 데이터 버퍼의 LRU1 존의 페이지 개수       |
| Num_data_page_lru2             | Snapshot | 데이터 버퍼의 LRU2 존의 페이지 개수       |
| Num_data_page_lru3             | Snapshot | 데이터 버퍼의 LRU3 존의 페이지 개수       |
| Num_data_page_victim_candidate | Snapshot | 데이터 버퍼의 희생(victim) 후보 페이지 개수 |

## 로그

| 항목                     | 통계 타입       | 설명                      |
|------------------------|-------------|-------------------------|
| Num_log_page_fetches   | Accumulator | 가져오기(fetch)한 로그 페이지의 개수 |
| Num_log_page_ioreads   | Accumulator | 읽은 로그 페이지의 개수           |
| Num_log_page_iowrites  | Accumulator | 저장한 로그 페이지의 개수          |
| Num_log_append_records | Accumulator | 추가(append)한 로그 레코드의 개수  |

|                                       |             |                                                                                                                                                           |
|---------------------------------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| Num_log_archives                      | Accumulator | 보관 로그의 개수                                                                                                                                                 |
| Num_log_start_checkpoints             | Accumulator | 체크포인트 시작 횟수                                                                                                                                               |
| Num_log_end_checkpoints               | Accumulator | 체크포인트 종료 횟수                                                                                                                                               |
| Num_log_wals                          | Accumulator | 데이터 페이지를 쓰기 위해 요청된 로그 플러시 횟수                                                                                                                              |
| Num_log_page_iowrites_for_replacement | Accumulator | 페이지 교체로 인해 디스크에 기록된 로그 데이터 페이지 수 (0 이 되어야 함)                                                                                                              |
| Num_log_page_replacements             | Accumulator | 페이지 교체로 인해 버려지는 로그 데이터 페이지 수                                                                                                                              |
| Num_prior_lsa_list_size               | Accumulator | 이전 LSA (Log Sequence Address) 목록의 현재 크기<br>CUBRID는 로그 버퍼에서 디스크로 쓰기 작업을 하기 전에<br>이전 LSA 목록에 쓰기 순서를 쓴다.<br>이 목록은 디스크에 기록하는 트랜잭션 대기시간을 줄임으로써 동시성을 높이는데 사용된다. |
| Num_prior_lsa_list_maxed              | Accumulator | 최대 크기에 도달한 이전 LSA 목록의 횟수<br>이전 LSA 목록의 최대 크기는 $\text{log\_buffer\_size} * 2$ 이다.<br>이 값이 크면 로그 작성 작업이 동시에 많이 발생한다고 가정할 수 있다.                              |
| Num_prior_lsa_list_removed            | Accumulator | 이전 LSA 목록에서 로그 버퍼로 이동한 횟수                                                                                                                                 |

이 값과 비슷한 횟수로 커밋이 발생했다고 가정할 수 있다.

|                           |          |                                                                                           |
|---------------------------|----------|-------------------------------------------------------------------------------------------|
| Log_page_buffer_hit_ratio | Computed | 로그 페이지 버퍼의 히트률<br>(Num_log_page_fetches - Num_log_page_ireads)*100 / Num_log_page_fetches |
|---------------------------|----------|-------------------------------------------------------------------------------------------|

### 동시성/잠금

| 항목                            | 통계 타입       | 설명                 |
|-------------------------------|-------------|--------------------|
| Num_page_locks_acquired       | Accumulator | 페이지 잠금을 획득한 횟수     |
| Num_object_locks_acquired     | Accumulator | 오브젝트 잠금을 획득한 횟수    |
| Num_page_locks_converted      | Accumulator | 페이지 잠금 타입을 변환한 횟수  |
| Num_object_locks_converted    | Accumulator | 오브젝트 잠금 타입을 변환한 횟수 |
| Num_page_locks_re-requested   | Accumulator | 페이지 잠금을 재요청한 횟수    |
| Num_object_locks_re-requested | Accumulator | 오브젝트 잠금을 재요청한 횟수   |
| Num_page_locks_waits          | Accumulator | 잠금을 대기하는 페이지 개수    |
| Num_object_locks_waits        | Accumulator | 잠금을 대기하는 오브젝트 개수   |



|                                   |             |                                      |
|-----------------------------------|-------------|--------------------------------------|
| Num_object_locks_time_waited_usec | Accumulator | 모든 오브젝트를 잠금하는데 소요된 시간 (microseconds) |
|-----------------------------------|-------------|--------------------------------------|

|                            |         |                          |
|----------------------------|---------|--------------------------|
| Time_obj_lock_acquire_time | Complex | 잠금 모드별로 객체를 잠그는데 소요되는 시간 |
|----------------------------|---------|--------------------------|

## 트랜잭션

| 항목 | 통계 타입 | 설명 |
|----|-------|----|
|----|-------|----|

|                  |             |        |
|------------------|-------------|--------|
| Num_tran_commits | Accumulator | 커밋한 횟수 |
|------------------|-------------|--------|

|                   |             |        |
|-------------------|-------------|--------|
| Num_tran_rollback | Accumulator | 롤백한 횟수 |
|-------------------|-------------|--------|

|                     |             |           |
|---------------------|-------------|-----------|
| Num_tran_savepoints | Accumulator | 세이프포인트 횟수 |
|---------------------|-------------|-----------|

|                       |             |                       |
|-----------------------|-------------|-----------------------|
| Num_tran_start_topops | Accumulator | 시작한 top operation의 개수 |
|-----------------------|-------------|-----------------------|

|                     |             |                       |
|---------------------|-------------|-----------------------|
| Num_tran_end_topops | Accumulator | 종료한 top operation의 개수 |
|---------------------|-------------|-----------------------|

|                     |             |         |
|---------------------|-------------|---------|
| Num_tran_interrupts | Accumulator | 인터럽트 횟수 |
|---------------------|-------------|---------|

|                              |             |                                                |
|------------------------------|-------------|------------------------------------------------|
| Num_tran_postpone_cache_hits | Accumulator | 트랜잭션 지연 (postpone) 연산들을 실행할 때 캐시에서 찾은 (hit) 횟수 |
|------------------------------|-------------|------------------------------------------------|

|                              |             |                                                                                             |
|------------------------------|-------------|---------------------------------------------------------------------------------------------|
| Num_tran_postpone_cache_miss | Accumulator | 트랜잭션 지연 (postpone) 연산들을 실행할 때 캐시에서 찾지 못한 (miss) 횟수. 캐시 미스 (cache miss)는 트랜잭션 커밋의 성능을 저하시킬 수 |
|------------------------------|-------------|---------------------------------------------------------------------------------------------|

있고, 전체 로그 연산들에 영향을 끼칠 수 있다.

Num\_tran\_topop\_postpone\_  
cache\_hits      Accumulator

top operation 지연 (postpone)연산들을 실행할 때 캐시에서 찾은 (hit) 횟수.

Num\_tran\_topop\_postpone\_  
cache\_miss      Accumulator

top operation 지연 (postpone)연산들을 실행할 때 캐시에서 찾지 못한 (miss) 횟수. 캐시 미스 (cache miss)는 top operation 커밋의 성능을 저하시킬 수 있고, 전체 로그 연산들에 영향을 끼칠 수 있다.

## 인덱스

| 항목                   | 통계 타입       | 설명                                           |
|----------------------|-------------|----------------------------------------------|
| Num_btree_inserts    | Accumulator | 삽입된 항목의 개수                                   |
| Num_btree_deletes    | Accumulator | 삭제된 항목의 개수                                   |
| Num_btree_updates    | Accumulator | 갱신된 항목의 개수                                   |
| Num_btree_covered    | Accumulator | 질의 시 인덱스가 데이터를 모두 포함한 경우의 개수                 |
| Num_btree_noncovered | Accumulator | 질의 시 인덱스가 데이터를 일부분만 포함하거나, 전혀 포함하지 않은 경우의 개수 |

|                                   |               |                                                                               |
|-----------------------------------|---------------|-------------------------------------------------------------------------------|
| Num_btree_resumes                 | Accumulator   | 많은 결과에 의한 지정된 인덱스 스캔 횟수를 초과한 개수                                               |
| Num_btree_multirange_optimization | Accumulator   | WHERE ... IN ... LIMIT 조건 질의문에 대해 다중 범위 최적화(multi-range optimization)를 수행한 횟수 |
| Num_btree_splits                  | Accumulator   | B-tree 노드의 분할 연산 횟수                                                           |
| Num_btree_merges                  | Accumulator   | B-tree 노드의 합병 연산 횟수                                                           |
| Num_btree_get_stats               | Accumulator   | B-tree 노드의 통계 호출 횟수                                                           |
| ..bt_leaf                         | Counter/timer | B-tree 단말들에서 모든 연산 시간과 횟수                                                     |
| ..bt_find_unique                  | Counter/timer | B-tree의 'find unique' 연산 시간과 횟수                                               |
| ..btrange_search                  | Counter/timer | B-tree의 'range search' 연산 시간과 횟수                                              |
| ..bt_insert_obj                   | Counter/timer | B-tree의 'insert object' 연산 시간과 횟수                                             |
| ..bt_delete_obj                   | Counter/timer | B-tree의 'physical delete object' 연산 시간과 횟수                                    |

|                       |               |                                           |
|-----------------------|---------------|-------------------------------------------|
| ..bt_mvcc_delete      | Counter/timer | B-tree의 'mvcc delete' 연산 시간과 횟수           |
| ..bt_mark_delete      | Counter/timer | B-tree의 'mark delete' 연산 시간과 횟수           |
| ..bt_undo_insert      | Counter/timer | B-tree의 'undo physical insert' 연산 시간과 횟수  |
| ..bt_undo_delete      | Counter/timer | B-tree의 'undo physical delete' 연산 시간과 횟수  |
| ..bt_undo_mvcc_delete | Counter/timer | B-tree의 'undo mvcc delete' 연산 시간과 횟수      |
| ..bt_vacuum           | Counter/timer | B-tree의 'vacuum deleted object' 연산 시간과 횟수 |
| ..bt_vacuum_insid     | Counter/timer | B-tree의 'vacuum insert id' 연산 시간과 횟수      |
| ..bt_fix_ovf_oids     | Counter/timer | B-tree의 'overflow page' 고정 시간과 횟수         |
| ..bt_unique_rlocks    | Counter/timer | 고유 인덱스에서 읽기 잠금이 차단된 시간과 횟수                |
| ..bt_unique_wlocks    | Counter/timer | 고유 인덱스에서 쓰기 잠금이 차단된 시간과 횟수                |
| ..bt_traverse         | Counter/timer | B-tree 순회 시간과 횟수                          |

|                                |               |                                                        |
|--------------------------------|---------------|--------------------------------------------------------|
| ..bt_find_unique_traverse      | Counter/timer | B-tree의 'find unique' 연산을 위한 B-tree 순회 시간과 횟수          |
| ..bt_range_search_traverse     | Counter/timer | B-tree의 'range search' 연산을 위한 B-tree 순회 시간과 횟수         |
| ..bt_insert_traverse           | Counter/timer | B-tree의 'insert' 연산을 위한 B-tree 순회 시간과 횟수               |
| ..bt_delete_traverse           | Counter/timer | B-tree의 'physical delete' 연산을 위한 B-tree 순회 시간과 횟수      |
| ..bt_mvcc_delete_traverse      | Counter/timer | B-tree의 'mvcc delete' 연산을 위한 B-tree 순회 시간과 횟수          |
| ..bt_mark_delete_traverse      | Counter/timer | B-tree의 'mark delete' 연산을 위한 B-tree 순회 시간과 횟수          |
| ..bt_undo_insert_traverse      | Counter/timer | B-tree의 'undo physical insert' 연산을 위한 B-tree 순회 시간과 횟수 |
| ..bt_undo_delete_traverse      | Counter/timer | B-tree의 'undo physical delete' 연산을 위한 B-tree 순회 시간과 횟수 |
| ..bt_undo_mvcc_delete_traverse | Counter/timer | B-tree의 'undo mvcc delete' 연산을 위한 B-tree 순회 시간과 횟수     |
| ..bt_vacuum_traverse           | Counter/timer |                                                        |

B-tree의 'vacuum deleted object' 연산을 위한 B-tree 순회 시간과 횟수

..bt\_vacuum\_insid\_traverse      Counter/timer

B-tree의 'vacuum insert id' 연산을 위한 B-tree 순회 시간과 횟수

## 온라인 인덱스 로드

### 항목

### 통계 타입

### 설명

..btree\_online\_load

Counter/timer

B-tree 온라인 인덱스를 로딩하는 구문의 수와 시간

..btree\_online\_insert\_task

Counter/timer

인덱스 로더가 수행한 배치 삽입 작업의 수와 시간

..btree\_online\_prepare\_task

Counter/timer

인덱스 로더 작업을 준비(정렬)한 수와 시간

..btree\_online\_insert\_leaf

Counter/timer

인덱스 로더가 처리한 리프 페이지의 수와 시간  
(리프 페이지를 처리하는 동안 여러 키가 삽입될 수 있음)

Num\_btree\_online\_inserts

Accumulator

인덱스 로더가 키를 삽입한 수

Num\_btree\_online\_inserts\_same\_page\_hold

Accumulator

동일한 리프 페이지에 인덱스 로더가 연속적으로 삽입한 수  
(값비싼 인덱스 순회를 피하기 위한 로더 최적화)

Accumulator

Num\_btree\_online\_inserts\_retry

연속적인 키가 동일한 리프에 속하지 않거나 공간이 부족하여 재시작된 삽입 수

Num\_btree\_online\_inserts\_retry\_nice

Accumulator

다른 요청이 리프 페이지에 액세스 할 수 있도록 재시작된 삽입 수

## 쿼리

| 항목                  | 통계 타입       | 설명                              |
|---------------------|-------------|---------------------------------|
| Num_query_selects   | Accumulator | SELECT 쿼리의 수행 횟수                |
| Num_query_inserts   | Accumulator | INSERT 쿼리의 수행 횟수                |
| Num_query_deletes   | Accumulator | DELETE 쿼리의 수행 횟수                |
| Num_query_updates   | Accumulator | UPDATE 쿼리의 수행 횟수                |
| Num_query_sscans    | Accumulator | 순차 스캔(full scan) 횟수             |
| Num_query_iscans    | Accumulator | 인덱스 스캔 횟수                       |
| Num_query_lscans    | Accumulator | LIST 스캔 횟수                      |
| Num_query_setscans  | Accumulator | SET 스캔 횟수                       |
| Num_query_methscans | Accumulator | METHOD 스캔 횟수                    |
| Num_query_nljoints  | Accumulator | 중첩 루프 조인 (nested loop joins) 횟수 |

|                  |             |          |
|------------------|-------------|----------|
| Num_query_mjoins | Accumulator | 병합 조인 횟수 |
|------------------|-------------|----------|

|                      |             |                      |
|----------------------|-------------|----------------------|
| Num_query_objfetches | Accumulator | 객체를 가져오기(fetch) 한 횟수 |
|----------------------|-------------|----------------------|

|                            |          |                                 |
|----------------------------|----------|---------------------------------|
| Num_query_holdable_cursors | Snapshot | 서버에서 유지 커서(holdable cursor)의 개수 |
|----------------------------|----------|---------------------------------|

## 정렬

| 항목 | 통계 타입 | 설명 |
|----|-------|----|
|----|-------|----|

|                   |             |                                                   |
|-------------------|-------------|---------------------------------------------------|
| Num_sort_io_pages | Accumulator | 정렬하는 동안 디스크에서 가져온(fetch) 페이지 개수 (이 값이 클수록 덜 효율적임) |
|-------------------|-------------|---------------------------------------------------|

|                     |             |                                               |
|---------------------|-------------|-----------------------------------------------|
| Num_sort_data_pages | Accumulator | 정렬하는 동안 페이지 버퍼에서 발견된 페이지 개수 (이 값이 클수록 덜 효율적임) |
|---------------------|-------------|-----------------------------------------------|

## 네트워크

| 항목 | 통계 타입 | 설명 |
|----|-------|----|
|----|-------|----|

|                      |             |            |
|----------------------|-------------|------------|
| Num_network_requests | Accumulator | 네트워크 요청 횟수 |
|----------------------|-------------|------------|

## 힙

| 항목 | 통계 타입 | 설명 |
|----|-------|----|
|----|-------|----|

|                                |             |                                                                                          |
|--------------------------------|-------------|------------------------------------------------------------------------------------------|
| Num_heap_stats_sync_best_space | Accumulator | <p>“Best page” 목록의 갱신 횟수.</p> <p>“Best page”는 여러 INSERT 및 DELETE 환경에서 여유 공간이 30% 이상인</p> |
|--------------------------------|-------------|------------------------------------------------------------------------------------------|



페이지를 의미한다.

“Best page” 목록에는 이 페이지의 일부 정보만 저장된다.

“Best page” 목록에는 백만 페이지의 정보가 한 번에 저장된다.

이 목록은 레코드를 INSERT 할 때 검색되며 해당 페이지의 여유 공간이

없을 때 “Best page” 목록은 갱신된다.

이 목록이 여러번 갱신되어도 더 이상 저장할 공간이 없는 경우 레코드는 새로운 페이지에 저장된다.

|                              |             |                                                      |
|------------------------------|-------------|------------------------------------------------------|
| Num_heap_home_inserts        | Accumulator | HOME 타입의 레코드 삽입 횟수                                   |
| Num_heap_big_inserts         | Accumulator | BIG 타입의 레코드 삽입 횟수                                    |
| Num_heap_assign_inserts      | Accumulator | ASSIGN 타입의 레코드 삽입 횟수                                 |
| Num_heap_home_deletes        | Accumulator | non-MVCC 모드에서 HOME 타입 레코드의 삭제 횟수                     |
| Num_heap_home_mvcc_deletes   | Accumulator | MVCC 모드에서 HOME 타입 레코드의 삭제 횟수                         |
| Num_heap_home_to_rel_deletes | Accumulator | MVCC 모드에서 HOME 타입으로부터 RELOCATION 타입으로 변경된 레코드의 삭제 횟수 |
| Num_heap_home_to_big_deletes | Accumulator |                                                      |

|                                  |             |                                                                         |
|----------------------------------|-------------|-------------------------------------------------------------------------|
|                                  |             | MVCC 모드에서 HOME 타입<br>으로부터 BIG 타입으로 변<br>경된 레코드의 삭제 횟수                   |
| Num_heap_rel_deletes             | Accumulator | non-MVCC 모드에서<br>RELOCATION 타입 레코드<br>의 삭제 횟수                           |
| Num_heap_rel_mvcc_delete<br>s    | Accumulator | MVCC 모드에서<br>RELOCATION 타입 레코드<br>의 삭제 횟수                               |
| Num_heap_rel_to_home_del<br>etes | Accumulator | MVCC 모드에서<br>RELOCATION 타입으로부터<br>HOME 타입으로 변경된 레<br>코드의<br>삭제 횟수       |
| Num_heap_rel_to_big_delet<br>es  | Accumulator | MVCC 모드에서<br>RELOCATION 타입으로부터<br>BIG 타입으로 변경된 레코<br>드의<br>삭제 횟수        |
| Num_heap_rel_to_rel_delet<br>es  | Accumulator | MVCC 모드에서<br>RELOCATION 타입으로부터<br>RELOCATION 타입으로 변경<br>된<br>레코드의 삭제 횟수 |
| Num_heap_big_deletes             | Accumulator | non-MVCC 모드에서 BIG 타<br>입 레코드의 삭제 횟수                                     |
| Num_heap_big_mvcc_delete<br>s    | Accumulator | MVCC 모드에서 BIG 타입 레<br>코드의 삭제 횟수                                         |
| Num_heap_home_updates            | Accumulator | non-MVCC 모드에서 HOME<br>타입 레코드의 갱신 횟수(*)                                  |
|                                  | Accumulator |                                                                         |

|                              |             |                                                                |
|------------------------------|-------------|----------------------------------------------------------------|
| Num_heap_home_to_rel_updates |             | non-MVCC 모드에서 HOME 타입으로부터 RELOCATION 타입으로 갱신된 레코드의 횟수(*)       |
| Num_heap_home_to_big_updates | Accumulator | non-MVCC 모드에서 HOME 타입으로부터 BIG 타입으로 갱신된 레코드의 횟수(*)              |
| Num_heap_rel_updates         | Accumulator | non-MVCC 모드에서 RELOCATION 레코드의 갱신 횟수(*)                         |
| Num_heap_rel_to_home_updates | Accumulator | non-MVCC 모드(*)에서 RELOCATION 타입으로부터 HOME 타입으로 갱신된 레코드의 횟수(*)    |
| Num_heap_rel_to_rel_updates  | Accumulator | non-MVCC 모드에서 RELOCATION 타입으로부터 RELOCATION 타입으로 갱신된 레코드의 횟수(*) |
| Num_heap_rel_to_big_updates  | Accumulator | non-MVCC 모드에서 BIG 타입으로부터 RELOCATION 타입으로 갱신된 횟수(*)             |
| Num_heap_big_updates         | Accumulator | non-MVCC 모드에서 BIG 타입으로 갱신된 레코드의 횟수(*)                          |
| Num_heap_home_vacuums        | Accumulator | HOME 타입 레코드의 회수된 횟수                                            |
| Num_heap_big_vacuums         | Accumulator | BIG 타입 레코드의 회수된 횟수                                             |

|                         |               |                               |
|-------------------------|---------------|-------------------------------|
| Num_heap_rel_vacuums    | Accumulator   | RELOCATION 타입 레코드의 회수된 횟수     |
| Num_heap_insid_vacuums  | Accumulator   | 새로 삽입된 레코드의 회수된 횟수            |
| Num_heap_remove_vacuums | Accumulator   | 삭제된 레코드의 회수된 횟수               |
| ..heap_insert_prepare   | Counter/timer | heap insert 연산에 대한 준비 횟수 및 시간 |
| ..heap_insert_execute   | Counter/timer | heap insert 연산에 대한 실행 횟수 및 시간 |
| ..heap_insert_log       | Counter/timer | heap insert 연산에 대한 로깅 횟수 및 시간 |
| ..heap_delete_prepare   | Counter/timer | heap delete 연산에 대한 준비 횟수 및 시간 |
| ..heap_delete_execute   | Counter/timer | heap delete 연산에 대한 실행 횟수 및 시간 |
| ..heap_delete_log       | Counter/timer | heap delete 연산에 대한 로깅 횟수 및 시간 |
| ..heap_update_prepare   | Counter/timer | heap update 연산에 대한 준비 횟수 및 시간 |
| ..heap_update_execute   | Counter/timer | heap update 연산에 대한 실행 횟수 및 시간 |

|                       |               |                               |
|-----------------------|---------------|-------------------------------|
| ..heap_update_log     | Counter/timer | heap update 연산에 대한 로깅 횟수 및 시간 |
| ..heap_vacuum_prepare | Counter/timer | heap vacuum 연산에 대한 준비 횟수 및 시간 |
| ..heap_vacuum_execute | Counter/timer | heap vacuum 연산에 대한 실행 횟수 및 시간 |
| ..heap_vacuum_log     | Counter/timer | heap vacuum 연산에 대한 로깅 횟수 및 시간 |

### 질의 계획 캐시

| 항목                    | 통계 타입       | 설명                                           |
|-----------------------|-------------|----------------------------------------------|
| Num_plan_cache_add    | Accumulator | 쿼리 캐시 엔트리가 새로 추가된 횟수                         |
| Num_plan_cache_lookup | Accumulator | 특정 키를 사용하여 쿼리 캐시 룩업(Lookup)을 시도한 횟수          |
| Num_plan_cache_hit    | Accumulator | 질의 문자열 해시 테이블에서 엔트리를 찾은(hit) 횟수              |
| Num_plan_cache_miss   | Accumulator | 질의 문자열 해시 테이블에서 엔트리를 찾지 못한(miss) 횟수          |
| Num_plan_cache_full   | Accumulator | 캐시 엔트리의 개수가 최대 개수를 넘어 희생자(victim) 탐색을 시도한 횟수 |
| Num_plan_cache_delete | Accumulator |                                              |

|                                |             |                                                                                                             |
|--------------------------------|-------------|-------------------------------------------------------------------------------------------------------------|
|                                |             | 캐시 엔트리가 삭제된 (victimized) 횟수                                                                                 |
| Num_plan_cache_invalid_xasl_id | Accumulator | xasl_id 해시 테이블에서 엔트리를 찾지 못한(miss) 횟수.<br>서버에서 특정 엔트리가 제거(victimized)되었는데, 해당 엔트리를 클라이언트에서 요청했을 때 발생하는 에러 횟수 |
| Num_plan_cache_entries         | Snapshot    | 쿼리 캐시 엔트리의 현재 개수                                                                                            |

## HA

| 항목                        | 통계 타입       | 설명          |
|---------------------------|-------------|-------------|
| Time_ha_replication_delay | Accumulator | 복제 지연 시간(초) |

## Vacuuming

| 항목                             | 통계 타입       | 설명                                                                                                         |
|--------------------------------|-------------|------------------------------------------------------------------------------------------------------------|
| Num_vacuum_log_pages_vacuumed  | Accumulator | vacuum 워커에 의해 처리된 로그 데이터 페이지 개수                                                                            |
| Num_vacuum_log_pages_to_vacuum | Accumulator | vacuum 워커에 의해 처리될 로그 데이터 페이지 개수 (이 값이 Num_vacuum_log_pages_vacuumed 보다 훨씬 큰 경우는 vacuum 시스템이 느리다는 것을 의미한다.) |

|                                            |               |                                                                                                                                                                                      |
|--------------------------------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Num_vacuum_prefetch_requ<br>ests_log_pages | Accumulator   | 로그 페이지를 버퍼로 프리<br>패치하는 요청 수                                                                                                                                                          |
| Num_vacuum_prefetch_hits<br>_log_pages     | Accumulator   | 로그 페이지를 버퍼로 프리<br>패치하는 히트 수                                                                                                                                                          |
| ..vacuum_master                            | Counter/timer | vacuum 마스터 작업 횟수<br>와 시간                                                                                                                                                             |
| ..vacuum_job                               | Counter/timer | vacuum 작업 횟수와 시간                                                                                                                                                                     |
| ..vacuum_worker_process_lo<br>g            | Counter/timer | vacuum 워커의 로그 작업<br>수행 횟수와 시간                                                                                                                                                        |
| ..vacuum_worker_execute                    | Counter/timer | vacuum 워커의 회수 작업<br>수행 횟수와 시간                                                                                                                                                        |
| Vacuum_data_page_buffer_<br>hit_ratio      | Computed      | 데이터 페이지 버퍼의 회수<br>히트율                                                                                                                                                                |
| Vacuum_page_efficiency_rat<br>io           | Computed      | 더티 플래그가있는 vacuum<br>의 page unfix 수와 전체<br>vacuum의 page unfix<br>수의 비율. 이상적으로는<br>vacuum프로세스는 사용되<br>지 않는 모든 레코드를<br>회수하므로 쓰기 작업만 수<br>행한다.<br>최적화된 회수 작업이라도<br>100% 효율은 가능하지 않<br>다. |
| Vacuum_page_fetch_ratio                    | Computed      | vacuum 모듈에서 page<br>unfix와 총 페이지의 비율<br>(백분율)                                                                                                                                        |

Snapshot

Num\_data\_page\_avoid\_deal  
loc

vacuum에 의해 반환 될 수  
없는 데이터 페이지 수

### 힙 Bestspace

| 통계 이름                            | 통계 타입         | 설명                                           |
|----------------------------------|---------------|----------------------------------------------|
| ..heap_stats_sync_bestspace      | Counter/timer | bestspace 동기화 수행 횟수와 시간                      |
| Num_heap_stats_bestspace_entries | Accumulator   | “best page” 목록에 저장된 “best page” 개수           |
| Num_heap_stats_bestspace_maxed   | Accumulator   | “best page” 목록에 저장할 수 있는 “best page” 최대값     |
| ..bestspace_add                  | Counter/timer | bestspace cache에 엔트리를 추가한 횟수와 시간             |
| ..bestspace_del                  | Counter/timer | bestspace cache에서 엔트리를 삭제한 횟수와 시간            |
| ..bestspace_find                 | Counter/timer | bestspace cache 엔트리를 찾은 횟수와 시간               |
| ..heap_find_page_bestspace       | Counter/timer | bestspace cache에서 힙 페이지를 찾은 횟수와 시간           |
| ..heap_find_best_page            | Counter/timer | 힙에 삽입하기 위해 페이지를 찾거나 (찾지 못했을 경우)생성한<br>횟수와 시간 |

### 페이지 버퍼 fix



| 항목                                   | 통계 타입       | 설명                                           |
|--------------------------------------|-------------|----------------------------------------------|
| Data_page_fix_lock_acquire_time_msec | Computed    | 페이지를 로드하기 위해 다른 트랜잭션을 기다리는 시간                |
| Data_page_fix_hold_acquire_time_msec | Computed    | 페이지 래치를 획득하는 시간                              |
| Data_page_fix_acquire_time_msec      | Computed    | 페이지를 고정하는 총 시간                               |
| Data_page_allocate_time_ratio        | Computed    | 디스크에서 페이지를 로딩하는 데 필요한 시간과 페이지를 고정하는 총 시간의 비율 |
| Data_page_total_promote_success      | Computed    | 공유에서 상호배제로의 성공한 페이지 래치 프로모션 수                |
| Data_page_total_promote_fail         | Computed    | 실패한 페이지 래치 프로모션 수                            |
| Data_page_total_promote_time_msec    | Computed    | 페이지 래치 프로모션한 시간                              |
| Num_data_page_hash_anchor_waits      | Accumulator | 페이지 버퍼 해시 버킷의 대기 수                           |
| Time_data_page_hash_anchor_wait      | Accumulator | 페이지 버퍼 해시 버킷의 총 대기 시간                        |
| Num_data_page_fix_ext                | Complex     | 다음으로 분류된 데이터 페이지 고정 수                        |

|                                  |         |                                                                                                                                                                                                              |
|----------------------------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                  |         | <ul style="list-style-type: none"> <li>- 모듈(system, worker, vacuum)</li> <li>- 페이지 타입</li> <li>- 페이지 페치/탐색 모드</li> <li>- 페이지 래치 모드</li> <li>- 페이지 래치 조건</li> </ul>                                           |
| Time_data_page_lock_acquire_time | Complex | <p>데이터 페이지를 로드 할 때까지 다른 스레드를 기다리는 시간</p> <ul style="list-style-type: none"> <li>- 모듈 (system, worker, vacuum)</li> <li>- 페이지 타입</li> <li>- 페이지 페치/탐색 모드</li> <li>- 페이지 래치 모드</li> <li>- 페이지 래치 조건</li> </ul> |
| Time_data_page_hold_acquire_time | Complex | <p>데이터 페이지 래치 대기 시간:</p> <ul style="list-style-type: none"> <li>- 모듈(system, worker, vacuum)</li> <li>- 페이지 타입</li> <li>- 페이지 페치/탐색 모드</li> <li>- 페이지 래치 모드</li> </ul>                                       |
| Time_data_page_fix_acquire_time  | Complex | <p>데이터 페이지 고정 시간:</p> <ul style="list-style-type: none"> <li>- 모듈(system, worker, vacuum)</li> <li>- 페이지 타입</li> <li>- 페이지 페치/탐색 모드</li> <li>- 페이지 래치 모드</li> <li>- 페이지 래치 조건</li> </ul>                     |
| Num_data_page_promote_ext        | Complex | <p>다음으로 분류 된 데이터 페이지 프로모션 수:</p> <ul style="list-style-type: none"> <li>- 모듈(system, worker, vacuum)</li> <li>- 페이지 타입</li> <li>- 프로모션 래치 조건</li> <li>- 홀더 래치 모드</li> <li>- 성공/실패 프로모션</li> </ul>            |

Num\_data\_page\_promote\_time\_ext      Complex

다음으로 분류 된 데이터 페이지 대기 시간:

- 모듈(system, worker, vacuum)
- 페이지 타입
- 프로모션 래치 조건
- 홀더 래치 모드
- 성공/실패 프로모션

### 페이지 버퍼 unfix

| 항목                                   | 통계 타입       | 설명                                              |
|--------------------------------------|-------------|-------------------------------------------------|
| Num_unfix_void_to_private_top        | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 전용 LRU 상단에 추가한 수        |
| Num_unfix_void_to_private_mid        | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 전용 LRU 중간에 추가한 수        |
| Num_unfix_void_to_shared_mid         | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 공유 LRU 중간에 추가한 수        |
| Num_unfix_lru1_private_to_shared_mid | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역1에서 공유 LRU 중간으로 이동한 수 |
| Num_unfix_lru2_private_to_shared_mid | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역2에서 공유 LRU 중간으로 이동한 수 |
| Num_unfix_lru3_private_to_shared_mid | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역3에서                   |

|                                   |             |                                         |
|-----------------------------------|-------------|-----------------------------------------|
|                                   |             | 공유 LRU 중간으로 이동한 수                       |
| Num_unfix_lru2_private_kee<br>p   | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역2에 보관한 수      |
| Num_unfix_lru2_shared_kee<br>p    | Accumulator | 데이터 페이지를 고정하지 않고 공유 LRU 영역2에 보관한 수      |
| Num_unfix_lru2_private_to_<br>top | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역2에서 상단으로 올린 수 |
| Num_unfix_lru2_shared_to_<br>top  | Accumulator | 데이터 페이지를 고정하지 않고 공유 LRU 영역2에서 상단으로 올린 수 |
| Num_unfix_lru3_private_to_<br>top | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역3에서 상단으로 올린 수 |
| Num_unfix_lru3_shared_to_<br>top  | Accumulator | 데이터 페이지를 고정하지 않고 공유 LRU 영역3에서 상단으로 올린 수 |
| Num_unfix_lru1_private_kee<br>p   | Accumulator | 데이터 페이지를 고정하지 않고 전용 LRU 영역1에 보관한 수      |
| Num_unfix_lru2_shared_kee<br>p    | Accumulator | 데이터 페이지를 고정하지 않고 공유 LRU 영역2에 보관한 수      |

Accumulator

|                                      |             |                                                                                                                           |
|--------------------------------------|-------------|---------------------------------------------------------------------------------------------------------------------------|
| Num_unfix_void_to_private_mid_vacuum |             | 새로 로드된 데이터 페이지를 고정하지 않고 전용 LRU 목록 중간에 추가한 수 (vacuum 쓰레드)                                                                  |
| Num_unfix_lru1_any_keep_vacuum       | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 전용/공유 LRU 영역 1에 보관한 수 (vacuum 쓰레드)                                                                |
| Num_unfix_lru2_any_keep_vacuum       | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 전용/공유 LRU 영역 2에 보관한 수 (vacuum 쓰레드)                                                                |
| Num_unfix_lru3_any_keep_vacuum       | Accumulator | 새로 로드된 데이터 페이지를 고정하지 않고 전용/공유 LRU 영역 3에 보관한 수 (vacuum 쓰레드)                                                                |
| Num_unfix_void_aout_found            | Accumulator | 새로 로드 된 데이터 페이지가 AOUT 에서 발견된 수                                                                                            |
| Num_unfix_void_aout_not_found        | Accumulator | 새로 로드 된 데이터 페이지가 AOUT 에서 발견되지 않은 수                                                                                        |
| Num_unfix_void_aout_found_vacuum     | Accumulator | 새로 로드 된 데이터 페이지가 AOUT 에서 발견된 수 (vacuum 쓰레드)                                                                               |
| Num_unfix_void_aout_not_found_vacuum | Accumulator | 새로 로드 된 데이터 페이지가 AOUT 에서 발견되지 않은 수(vacuum 쓰레드)                                                                            |
| Num_data_page_unfix_ext              | Complex     | 다음으로 분류 된 데이터 페이지 비 고정 수 <ul style="list-style-type: none"> <li>- 모듈(system, worker, vacuum)</li> <li>- 페이지 타입</li> </ul> |

- 더티이거나 아닌 경우
- 홀더에 의한 더티이거나 아닌 경우
- 홀더 래치 모드

## 페이지 버퍼 I/O

| 항목                           | 통계 타입         | 설명                                                                                                                                     |
|------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Data_page_buffer_hit_ratio   | Computed      | 데이터 페이지 버퍼의 히트율<br>$\frac{(\text{Num\_data\_page\_fetches} - \text{Num\_data\_page\_ioreads}) * 100}{\text{Num\_data\_page\_fetches}}$ |
| Num_adaptive_flush_pages     | Accumulator   | 적응형 플러시 컨트롤러에서 요청한 데이터 페이지 수                                                                                                           |
| Num_adaptive_flush_log_pages | Accumulator   | 적응형 플러시 컨트롤러에서 요청한 로그 데이터 페이지 수                                                                                                        |
| Num_adaptive_flush_max_pages | Accumulator   | 적응형 플러시 컨트롤러에 의해 할당된 토큰 페이지 총 수                                                                                                        |
| ..compensate_flush           | Counter/timer | 적응형 플러시 컨트롤러에 의한 플러시 보정의 횟수와 시간                                                                                                        |
| ..assign_direct_bcb          | Counter/timer | 대기자 (waiter)들에게 직접적으로 bcb를 할당한 횟수와 시간                                                                                                  |
| ..wake_flush_waiter          | Counter/timer | BCB를 기다리는 쓰레드를 깨우기 위한 횟수와 시간                                                                                                           |

|                                             |               |                                                 |
|---------------------------------------------|---------------|-------------------------------------------------|
| ..flush_collect                             | Counter/timer | BCB 세트를 수집하는 플러시 쓰레드의 수와 시간                     |
| ..flush_flush                               | Counter/timer | BCB 세트를 플러싱하는 플러시 쓰레드의 수와 시간                    |
| ..flush_sleep                               | Counter/timer | 플러시 쓰레드 일시 정지 수와 시간                             |
| ..flush_collect_per_page                    | Counter/timer | 한 개의 BCB 를 수집하는 플러시 쓰레드의 수와 시간                  |
| ..flush_flush_per_page                      | Counter/timer | 한 개의 BCB 를 플러싱하는 플러시 쓰레드의 수와 시간                 |
| Num_data_page_writes                        | Accumulator   | 디스크로 내려쓰기된 데이터 페이지의 총 수                         |
| Num_data_page_dirty_to_post_flush           | Accumulator   | 포스트 플러시 쓰레드로 보내진 플러시 된 페이지 수                    |
| Num_data_page_skipped_flush                 | Accumulator   | 플러시 쓰레드가 생략한 BCB의 총 수                           |
| Num_data_page_skipped_flush_need_wal        | Accumulator   | 로그 데이터 페이지를 먼저 플러시해야하기 때문에 플러시 쓰레드가 생략한 BCB 수   |
| Num_data_page_skipped_flush_already_flushed | Accumulator   | 이미 플러시 되어서 쓰레드가 생략한 BCB 수                       |
| Num_data_page_skipped_flush_fixed_or_hot    | Accumulator   | BCB가 고정되었거나 수집된 후 고정되었기 때문에 플러시 쓰레드가 생략한 BCB의 수 |

## 페이지 버퍼 victimization

| 항목                                        | 통계 타입         | 설명                                                                                                                                                                                                       |
|-------------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ..alloc_bcb                               | Counter/timer | <p>새로운 데이터 페이지를 저장하기 위한 BCB 할당의 수와 시간.</p> <p>데이터베이스가 시작되면 페이지 버퍼는 사용가능한 BCB가 있다.</p> <p>그러나 페이지 버퍼가 모두 사용되면 모든 BCB가 사용중이므로</p> <p>페이지 버퍼 중 하나는 희생되어야 한다. 여기서 추적되는 시간은</p> <p>BCB 희생 및 디스크 로딩을 포함한다.</p> |
| ..alloc_bcb_search_victim                 | Counter/timer | <p>희생자(Victim)에 대한 모든 LRU 을 통한 검색 횟수 및 시간</p>                                                                                                                                                            |
| ..alloc_bcb_cond_wait_high_prio           | Counter/timer | <p>우선 순위가 높은 대기열에서 직접적인 희생자(Victim) 대기의 수와 시간</p>                                                                                                                                                        |
| ..alloc_bcb_cond_wait_low_prio            | Counter/timer | <p>우선 순위가 낮은 대기열에서 직접적인 희생자(Victim) 대기의 수와 시간</p>                                                                                                                                                        |
| Num_alloc_bcb_prioritize_vacuum           | Accumulator   | <p>우선 순위가 높은 대기열에서 직접적인 희생자(Victim) 회수의 대기 수</p>                                                                                                                                                         |
| Num_alloc_bcb_wait_thread_s_high_priority | Snapshot      | <p>우선 순위가 높은 대기열에서 직접적인 희생(Victim) 대기자의 현재 수</p>                                                                                                                                                         |



|                                                   |               |                                                  |
|---------------------------------------------------|---------------|--------------------------------------------------|
| Num_alloc_bcb_wait_thread_s_low_priority          | Snapshot      | 우선 순위가 낮은 대기열에서 직접적인 희생(Victim) 대기자의 현재 수        |
| Num_flushed_bcbs_wait_for_direct_victim           | Snapshot      | 희생자를 처리하여 즉시 할당하는 post-flush 쓰레드를 기다리는 BCB의 현재 수 |
| Num_victim_use_invalid_bcb                        | Accumulator   | 유효하지 않은 목록에서 할당된 BCB의 수                          |
| ..alloc_bcb_get_victim_search_own_private_list    | Counter/timer | 자체 전용 리스트에서 희생자(Victim)을 가져오는 횟수 및 시간            |
| ..alloc_bcb_get_victim_search_others_private_list | Counter/timer | 다른 전용 리스트에서 희생자(Victim)을 가져오는 횟수 및 시간            |
| ..alloc_bcb_get_victim_search_shared_list         | Counter/timer | 공유 리스트에서 희생자(Victim)을 가져오는 횟수 및 시간               |
| Num_data_page_avoid_victim                        | Accumulator   | 디스크로 내려쳐지는 중이라 희생에서 제외되는 BCB의 수                  |
| Num_victim_assign_direct_vacuum_void              | Accumulator   | vacuum 워커가 빈(void) 영역에서 할당한 직접 희생자의 수            |
| Num_victim_assign_direct_vacuum_lru               | Accumulator   | vacuum 워커가 LRU 영역 3에서 할당한 직접 희생자 수               |
| Num_victim_assign_direct_flush                    | Accumulator   | 플러시 쓰레드가 지정한 직접적인 희생자 수                          |

|                                               |             |                                                                                                                                    |
|-----------------------------------------------|-------------|------------------------------------------------------------------------------------------------------------------------------------|
| Num_victim_assign_direct_panic                | Accumulator | 패닉 LRU 검색에 의해 할당된 직접적인 희생자의 수<br>희생자를 찾기 위한 대기자가 많으면 LRU 목록을 검색하는 동안 다른 희생자를 찾는 스레드가 직접 할당하려고 시도한다.<br>페이지 버퍼 유지 스레드 할당도 여기에 포함된다. |
| Num_victim_assign_direct_adjust_lru           | Accumulator | BCB가 LRU 영역 3으로 이동할 때 할당된 직접적인 희생자 수                                                                                               |
| Num_victim_assign_direct_adjust_lru_to_vacuum | Accumulator | vacuum 스레드가 액세스할 것으로 예상되어 LRU 영역 3으로 이동할 때 직접적인 희생자로 할당되지 않은 BCB의 수                                                                |
| Num_victim_assign_direct_search_for_flush     | Accumulator | 플러시를 위해 BCB셋(Set)을 수집하는 동안 플러시 스레드에 의해 할당된 직접적인 희생자 수                                                                              |
| Num_victim_shared_lru_success                 | Accumulator | 공유 LRU 에서 성공한 희생자 검색 수                                                                                                             |
| Num_victim_own_private_lru_success            | Accumulator | 자체 전용 LRU 에서 성공한 희생자 검색 수                                                                                                          |
| Num_victim_other_private_lru_success          | Accumulator | 다른 전용 LRU 에서 성공한 희생자 검색 수                                                                                                          |
| Num_victim_shared_lru_fail                    | Accumulator | 공유 LRU 에서 실패한 희생자 검색 수                                                                                                             |

|                                   |             |                                                                                                                                                                |
|-----------------------------------|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Num_victim_own_private_lru_fail   | Accumulator | 자체 전용 LRU에서 실패한 희생자 검색 수                                                                                                                                       |
| Num_victim_other_private_lru_fail | Accumulator | 다른 전용 LRU에서 실패한 희생자 검색 수                                                                                                                                       |
| Num_victim_all_lru_fail           | Accumulator | 아래의 순서에서 희생자를 찾을 때 좋지 않은 경우의 수:<br>1. 자체 전용 LRU (쿼터가 초과될 경우)<br>2. 다른 전용 LRU (자체 전용 쿼터가 초과될 경우)<br>3. 공유 LRU<br>(자세한 설명은 다음을 참고한다. <i>pgbuf_get_victim</i> 함수) |
| Num_victim_get_from_lru           | Accumulator | 모든 LRU 에서 희생자 검색 총 수                                                                                                                                           |
| Num_victim_get_from_lru_was_empty | Accumulator | 후보 수가 0이기 때문에 모든 LRU에서 희생자 검색이 즉시 중지된 수                                                                                                                        |
| Num_victim_get_from_lru_fail      | Accumulator | 후보 수가 0이 아니더라도 모든 LRU 에서 희생자 검색이 실패된 수                                                                                                                         |
| Num_victim_get_from_lru_bad_hint  | Accumulator | 희생자가 잘못되어 모든 LRU 에서 희생자 검색이 실패된 수                                                                                                                              |
| Num_lfcq_prv_get_total_calls      | Accumulator | 후보 수가 0이 아닌 전용 LRU 큐에서 희생자 검색 수                                                                                                                                |
| Num_lfcq_prv_get_empty            | Accumulator | 후보수가 0이 아닌 전용 LRUs 큐가 비어있는 횟수                                                                                                                                  |

|                              |             |                                                                  |
|------------------------------|-------------|------------------------------------------------------------------|
| Num_lfcq_prv_get_big         | Accumulator | 후보 수가 매우 큰 전용 LRU 큐의 희생자 검색 수<br>(매우 큰 것의 의미는 할당량을 초과하는 경우를 나타냄) |
| Num_lfcq_shr_get_total_calls | Accumulator | 후보수가 0이 아닌 공유 LRU 큐에서 희생자 검색 수                                   |
| Num_lfcq_shr_get_empty       | Accumulator | 후보수가 0이 아닌 공유 LRU 큐가 비어있는 횟수                                     |
| Num_lfcq_big_private_lists   | Snapshot    | 후보자가 매우 큰 전용 LRU의 현재 수                                           |
| Num_lfcq_private_lists       | Snapshot    | 후보자가 0이 아닌 전용 LRU의 현재 수                                          |
| Num_lfcq_shared_lists        | Snapshot    | 후보자가 0이 아닌 공유 LRU의 현재 수                                          |

### 이중 쓰기 버퍼 (Double write buffer)

| 항 목                         | 통계 타입         | 설명                                     |
|-----------------------------|---------------|----------------------------------------|
| ..DWB_flush_block           | Counter/timer | 플러시 (flush)되는 블록 (block)의 개수와 전체 쓰기 시간 |
| ..DWB_file_sync_helper      | Counter/timer | DWB helper에 의해 동기화된 파일의 개수와 시간         |
| ..DWB_flush_block_cond_wait | Counter/timer | DWB 쓰레드들이 기다리는 횟수와 시간                  |
| ..DWB_flush_block_sort      | Counter/timer |                                        |

|                                     |               |                                                                                       |
|-------------------------------------|---------------|---------------------------------------------------------------------------------------|
|                                     |               | 플러시 (flush)하기 위하여<br>정렬된 페이지의 개수와 시<br>간                                              |
| ..DWB_decache_pages_after<br>_write | Counter/timer | 플러시 (flush) 이후 DWB 캐<br>시에서 제거되는 페이지들<br>의 개수와 시간                                     |
| ..DWB_wait_flush_block              | Counter/timer | 페이지를 추가하기 위하여<br>블록이 플러시 (flush)되기<br>를 기다리는<br>횟수와 시간                                |
| ..DWB_wait_file_sync_helper         | Counter/timer | DWB helper가 파일을 동기<br>화하는 것을 기다리는 횟수<br>와 시간                                          |
| ..DWB_flush_force                   | Counter/timer | 강제로 DWB를 완전히 플러<br>시 (flush)한 횟수와 시간                                                  |
| Num_dwb_flushed_block_vo<br>lumes   | Complex       | 각각의 블록 플러시 (block<br>flush)에 의해 동기화된 파<br>일들 개수의<br>히스토그램 (마지막 값은<br>10 혹은 그 이상의 파일들) |

### MVCC 스냅샷

| 항목                                  | 통계 타입       | 설명                                     |
|-------------------------------------|-------------|----------------------------------------|
| Time_get_snapshot_acquire<br>_time: | Accumulator | 스냅샷을 획득하기 위해 모<br>든 트랜잭션이 수행된 총 시<br>간 |
| Count_get_snapshot_retry:           | Accumulator | MVCC 스냅샷을 획득하기 위<br>한 재시도 횟수           |

|                                    |             |                                                                         |
|------------------------------------|-------------|-------------------------------------------------------------------------|
| Time_tran_complete_time:           | Accumulator | 커밋 / 롤백시 스냅샷 및 MVCCID를 무효화하는 데 수행된 시간                                   |
| Time_get_oldest_mvcc_acquire_time: | Accumulator | “가장 오래된 전역 MVCC ID”를 획득하기 위해 수행된 시간                                     |
| Count_get_oldest_mvcc_retry:       | Accumulator | “가장 오래된 전역 MVCC ID”를 획득하기 위한 재시도 횟수                                     |
| Num_mvcc_snapshot_ext              | Complex     | 다음과 같이 분류된 데이터 페이지 수정 수<br>- 스냅샷 타입<br>- 삽입/삭제 MVCCID의 상태<br>- 가시성/비가시성 |

## 워커 스레드

워커 스레드 풀에서 수집한 통계 정보:

| 항목                    | 통계 타입         | 설명                   |
|-----------------------|---------------|----------------------|
| ..start_thread        | Counter/timer | 새 스레드를 시작한 횟수와 시간    |
| ..create_context      | Counter/timer | 스레드 컨텍스트를 생성한 횟수와 시간 |
| ..execute_task        | Counter/timer | 실행한 작업의 수와 시간        |
| ..retire_task         | Counter/timer | 폐기한 작업 객체의 수와 시간     |
| ..found_task_in_queue | Counter/timer |                      |

대기열에서 발견한 작업의 수와 이전 작업을 완료한 후 대기열에 요청한 시간

|                    |               |                                          |
|--------------------|---------------|------------------------------------------|
| ..wakeup_with_task | Counter/timer | 대기 중인 스레드에 할당된 작업의 수와 스레드를 깨우기 위해 사용한 시간 |
| ..recycle_context  | Counter/timer | 작업 실행 간 재활용한 스레드 컨텍스트의 수와 시간             |
| ..retire_context   | Counter/timer | 폐기한 스레드 컨텍스트의 수와 시간                      |

두개의 워커 풀에서 수집한 통계 정보:

- **Thread\_stats\_counters\_timers**: 클라이언트 요청을 처리하는 워커에 대한 통계 정보
- **Thread\_loaddb\_stats\_counters\_timers**: loaddb 명령 사용시 데이터베이스에 데이터를 로딩하는 워커에 대한 통계 정보 이 통계 정보를 보려면 최소한 하나의 로드 세션이 활성화되어 있어야 한다.

데몬 스레드:

백그라운드 데몬 스레드가 수집한 통계 정보:

| 항목                    | 통계 타입       | 설명                    |
|-----------------------|-------------|-----------------------|
| ..daemon_loop_count   | Accumulator | 데몬 스레드가 실행한 반복 횟수     |
| ..daemon_execute_time | Accumulator | 데몬 스레드가 실행 시 사용한 총 시간 |
| ..daemon_pause_time   | Accumulator | 데몬 스레드가 실행 간 소비한 총 시간 |
| ..looper_sleep_count  | Accumulator |                       |

|                            |             |                            |
|----------------------------|-------------|----------------------------|
|                            |             | 스레드 반복자가 정지 상태가 된 횟수       |
| ..looper_sleep_time        | Accumulator | 스레드 반복자가 정지 상태에서 소비한 총 시간  |
| ..looper_reset_count       | Accumulator | 증분 반복자가 깨어나며 재설정되는 횟수      |
| ..waiter_wakeup_count      | Accumulator | 데몬 스레드에 깨우기 요청한 횟수         |
| ..waiter_lock_wakeup_count | Accumulator | 잠금 요청한 데몬 스레드 깨우기 요청 횟수    |
| ..waiter_sleep_count       | Accumulator | 데몬 스레드가 정지 상태가 된 횟수        |
| ..waiter_timeout_count     | Accumulator | 데몬 스레드가 시간 초과로 종료한 대기 횟수   |
| ..waiter_no_sleep_count    | Accumulator | 데몬 스레드가 대기하지 않고 반복한 횟수     |
| ..waiter_awake_count       | Accumulator | 다른 스레드의 요청으로 데몬 스레드를 깨운 횟수 |
| ..waiter_wakeup_delay_time | Accumulator | 데몬 스레드를 깨우는데 사용한 총 시간      |

다음 데몬 스레드에 대해 수집된 통계 정보:

| 데몬 이름 | 설명 |
|-------|----|
|-------|----|

Page\_flush\_daemon\_thread



디스크에 데이터 페이지를 내보내는 백그라운드 스레드

Page\_post\_flush\_daemon\_thread

데이터 페이지를 내보내기 위해 사용한 메모리 공간을 회수하는 백그라운드 스레드

Page\_flush\_control\_daemon\_thread

데이터 내보내기 비율을 조정하는 백그라운드 스레드

Page\_maintenance\_daemon\_thread

개인 LRU 리스트의 할당량을 재계산하는 백그라운드 스레드

Deadlock\_detect\_daemon\_thread

교착 상태를 찾고 시스템 정지를 막기 위해 희생자(victim)를 선택하는 백그라운드 스레드

Log\_flush\_daemon\_thread

디스크에 로그 데이터를 내보내는 백그라운드 스레드

### 참고

(\*) : 통계치는 내부에서 수행되는 비 MVCC 연산 또는 MVCC 연산에 대한 측정 자료이다 (내부적으로 결정됨)

### **-o, --output-file=FILE**

**-o** 옵션을 이용하여 대상 데이터베이스 서버의 실행 통계 정보를 지정된 파일에 저장한다.

```
cubrid statdump -o statdump.log testdb
```

### **-c, --cumulative**

**-c** 옵션을 이용하여 대상 데이터베이스 서버의 누적된 실행 통계 정보를 출력할 수 있다.

Num\_data\_page\_fix\_ext, Num\_data\_page\_unfix\_ext, Time\_data\_page\_hold\_acquire\_time, Time\_data\_page\_fix\_acquire\_time 정보는 이 옵션을 켜야만 출력되는데, 해당 정보들은 CUBRID 엔진 개발자를 위한 정보이므로 설명을 생략한다.

**-i** 옵션과 결합하면, 지정된 시간 간격(interval)마다 실행 통계 정보를 확인할 수 있다.

```
cubrid statdump -i 5 -c testdb
```

### **-s, --substr=STRING**

**-s** 옵션 뒤에 문자열을 지정하면, 항목 이름 내에 해당 문자열을 포함하는 통계 정보만 출력할 수 있다.

다음 예는 항목 이름 내에 “data”를 포함하는 통계 정보만 출력한다.

```
cubrid statdump -s data testdb

*** SERVER EXECUTION STATISTICS ***
Num_data_page_fetches = 0
Num_data_page_dirties = 0
Num_data_page_ioreads = 0
Num_data_page_iowrites = 0
Num_data_page_flushed = 0
Num_data_page_private_quota = 327
Num_data_page_private_count = 898
Num_data_page_fixed = 1
Num_data_page_dirty = 3
Num_data_page_lru1 = 857
Num_data_page_lru2 = 873
Num_data_page_lru3 = 898
Num_data_page_victim_candidate = 898
Num_sort_data_pages = 0
Vacuum_data_page_buffer_hit_ratio = 0.00
Num_data_page_hash_anchor_waits = 0
Time_data_page_hash_anchor_wait = 0
Num_data_page_writes = 0
Num_data_page_dirty_to_post_flush = 0
Num_data_page_skipped_flush = 0
Num_data_page_skipped_flush_need_wal = 0
Num_data_page_skipped_flush_already_flushed = 0
Num_data_page_skipped_flush_fixed_or_hot = 0
Num_data_page_avoid_dealloc = 0
Num_data_page_avoid_victim = 0
Num_data_page_fix_ext:
Num_data_page_promote_ext:
Num_data_page_promote_time_ext:
Num_data_page_unfix_ext:
Time_data_page_lock_acquire_time:
Time_data_page_hold_acquire_time:
Time_data_page_fix_acquire_time:
```

각 상태 정보는 64비트 **INTEGER** 로 구성되어 있으며, 누적된 값이 한도를 넘으면 해당 실행 통계 정보가 유실될 수 있다.

### 참고

아래는 활성화/비활성화 할 수 있는 성능 통계 세트 목록이다. 각 세트의 값은 2의 제곱으로 표현된다. 이 세트들은 **extended\_statistics\_activation** 시스템 매개 변수 값에 의해 활성화/비활성화되며, 이 변수 값은 2진수 형식으로 설정한다

| 값 | 이름                    | 활성 기본값 | 설명                                                                                                                                                                                                                                                                                                                                                                                                |
|---|-----------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | Detailed b-tree pages | Yes    | <p>B- 트리 페이지를 3 가지 카테고리 로 분류한다.: 루트, 비단말, 단말</p> <p>관련 통계수치:</p> <ul style="list-style-type: none"> <li>-</li> <li>Num_data_page_fix_ext</li> <li>-</li> <li>Time_data_page_lock_acquire_time</li> <li>-</li> <li>Time_data_page_hold_acquire_time</li> <li>-</li> <li>Time_data_page_fix_acquire_time</li> <li>-</li> <li>Num_data_page_promote_ext</li> <li>-</li> <li>Num_data_page</li> </ul> |

| 값  | 이름                            | 활성 기본값 | 설명                                                                                                           |
|----|-------------------------------|--------|--------------------------------------------------------------------------------------------------------------|
|    |                               |        | _promote_time_ext<br>-<br>Num_data_page_unfix_ext                                                            |
| 2  | <b>MVCC Snapshot</b>          | Yes    | MVCC 스냅샷에 대한 통계 수집 :<br>-<br>Num_mvcc_snapshot_ext                                                           |
| 4  | <b>Time locks</b>             | Yes    | 타이밍 잠금 대기<br>에 대한 통계 수<br>집 :<br>-<br>Num_object_locks_time_waited_usec                                      |
| 8  | <b>Hash anchor waits</b>      | Yes    | 해시 앵커 대기<br>에 대한 통계 수<br>집 :<br>-<br>Num_data_page_hash_anchor_waits<br>-<br>Time_data_page_hash_anchor_wait |
| 16 | <b>Extended victimization</b> | No     | 확장 페이지 버퍼 I/O 및 victim<br>에 대한<br>통계 수집 :                                                                    |

| 값 | 이름 | 활성 기본값 | 설명                                                  |
|---|----|--------|-----------------------------------------------------|
|   |    |        | -                                                   |
|   |    |        | Num_data_page<br>_writes                            |
|   |    |        | -                                                   |
|   |    |        | flush_collect_per<br>_page                          |
|   |    |        | -                                                   |
|   |    |        | flush_flush_per_<br>page                            |
|   |    |        | -                                                   |
|   |    |        | Num_data_page<br>_skipped_flush_<br>already_flushed |
|   |    |        | -                                                   |
|   |    |        | Num_data_page<br>_skipped_flush_<br>need_wal        |
|   |    |        | -                                                   |
|   |    |        | Num_data_page<br>_skipped_flush_f<br>ixed_or_hot    |
|   |    |        | -                                                   |
|   |    |        | Num_data_page<br>_dirty_to_post_fl<br>ush           |
|   |    |        | -                                                   |
|   |    |        | Num_alloc_bcb_<br>prioritize_vacuu<br>m             |
|   |    |        | -                                                   |
|   |    |        | Num_victim_assi<br>gn_direct_vacuu<br>m_void        |
|   |    |        | -                                                   |
|   |    |        | Num_victim_assi<br>gn_direct_vacuu<br>m_lru         |

| 값 | 이름 | 활성 기본값 | 설명                                            |
|---|----|--------|-----------------------------------------------|
|   |    |        | -                                             |
|   |    |        | Num_victim_assign_direct_flush                |
|   |    |        | -                                             |
|   |    |        | Num_victim_assign_direct_panic                |
|   |    |        | -                                             |
|   |    |        | Num_victim_assign_direct_adjust_lru           |
|   |    |        | -                                             |
|   |    |        | Num_victim_assign_direct_adjust_lru_to_vacuum |
|   |    |        | -                                             |
|   |    |        | Num_victim_assign_direct_search_for_flush     |
|   |    |        | -                                             |
|   |    |        | Num_victim_shared_lru_success                 |
|   |    |        | -                                             |
|   |    |        | Num_victim_own_private_lru_success            |
|   |    |        | -                                             |
|   |    |        | Num_victim_other_private_lru_success          |
|   |    |        | -                                             |
|   |    |        | Num_victim_shared_lru_fail                    |
|   |    |        | -                                             |
|   |    |        | Num_victim_own_private_lru_fail               |
|   |    |        | l                                             |

| 값  | 이름             | 활성 기본값 | 설명                                                                                                                                                                                                                                                                                                                                                                     |
|----|----------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |                |        | -<br>Num_victim_other_private_lru_fail<br>-<br>Num_victim_get_from_lru<br>-<br>Num_victim_get_from_lru_was_empty<br>-<br>Num_victim_get_from_lru_fail<br>-<br>Num_victim_get_from_lru_bad_hint<br>-<br>Num_lfcq_priv_get_total_calls<br>-<br>Num_lfcq_priv_get_empty<br>-<br>Num_lfcq_priv_get_big<br>-<br>Num_lfcq_shr_get_total_calls<br>-<br>Num_lfcq_shr_get_empty |
| 32 | Thread workers | No     | 스레드 워커 풀에 대한 통계 수집 :                                                                                                                                                                                                                                                                                                                                                   |

| 값   | 이름             | 활성 기본값 | 설명                                                                            |
|-----|----------------|--------|-------------------------------------------------------------------------------|
|     |                |        | -<br>Thread_stats_counters_timers<br>-<br>Thread_loaddb_stats_counters_timers |
| 64  | Thread daemons | No     | 데몬 스레드에 대한 통계 수집 :<br>-<br>Thread_pgbuf_daemon_stats_counters_timers          |
| 128 | Extended DWB   | No     | 이중 쓰기 버퍼에 대한 통계 수집 :<br>-<br>Num_dwb_flushed_block_volumes                    |
| MAX | All statistics | No     | 모든 통계 수집                                                                      |

## lockdb

**cubrid lockdb** 는 대상 데이터베이스에 대하여 현재 트랜잭션에서 사용되고 있는 잠금 정보를 확인하는 유틸리티이다.

```
cubrid lockdb [<option>] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.



- **lockdb**: 대상 데이터베이스에 대하여 현재 트랜잭션에서 사용되고 있는 잠금 정보를 확인하는 명령이다.
- *database\_name*: 현재 트랜잭션의 잠금 정보를 확인하는 데이터베이스 이름이다.

다음 예는 옵션 없이 testdb 데이터베이스의 잠금 정보를 화면에 출력한다.

```
cubrid lockdb testdb
```

다음은 **cubrid lockdb** 에 대한 [option]이다.

#### **-o, --output-file=FILE**

데이터베이스의 잠금 정보를 output.txt로 출력한다.

```
cubrid lockdb -o output.txt testdb
```

#### **-c, --contention**

경합중인 잠금 정보만 출력한다.

```
cubrid lockdb -c testdb
```

## 출력 내용

**cubrid lockdb** 의 출력 내용은 논리적으로 3개의 섹션으로 나뉘어져 있다.

- 서버에 대한 잠금 설정
- 현재 데이터베이스에 접속한 클라이언트들
- 객체 잠금 테이블의 내용

### 서버에 대한 잠금 설정

**cubrid lockdb** 출력 내용의 첫 번째 섹션은 데이터베이스 서버에 대한 잠금 설정이다.

```
*** Lock Table Dump ***
Lock Escalation at = 100000, Run Deadlock interval = 0
```

위에서 잠금 에스컬레이션 레벨은 100000레코드로, 교착 상태 탐지 간격은 0초로 설정되어 있다.

관련 시스템 파라미터인 **lock\_escalation** 과 **deadlock\_detection\_interval** 에 대한 설명은 **동시성/잠금 파라미터** 를 참고한다.

### 현재 데이터베이스에 접속한 클라이언트들

**cubrid lockdb** 출력 내용의 두 번째 섹션은 데이터베이스에 연결된 모든 클라이언트의 정보를 포함한다. 이 정보에는 각각의 클라이언트에 대한 트랜잭션 인덱스, 프로그램 이름, 사용자 ID, 호스트 이름, 프로세스 ID, 격리 수준, 그리고 잠금 타임아웃 설정이 포함된다.

```
Transaction (index 1, csql, dba@cubridb|12854)
Isolation COMMITTED READ
Timeout_period : Infinite wait
```

위에서 트랜잭션 인덱스는 1이고, 프로그램 이름은 csql, 사용자 id은 dba, 호스트 이름은 cubridb, 클라이언트 프로세스 id는 12854, 격리 수준은 COMMITTED READ, 그리고 잠금 타임아웃은 무제한이다.

트랜잭션 인덱스가 0인 클라이언트는 내부적인 시스템 트랜잭션이다. 이것은 데이터베이스의 체크포인트 수행과 같이 특정한 시간에 잠금을 획득할 수 있지만 대부분의 경우 이 트랜잭션은 어떤 잠금도 획득하지 않을 것이다.

**cubrid lockdb** 유틸리티는 잠금 정보를 가져오기 위해 데이터베이스에 접속하기 때문에 **cubrid lockdb** 자체가 하나의 클라이언트이고 따라서 클라이언트의 하나로 출력된다.

### 객체 잠금 테이블

**cubrid lockdb** 출력 내용의 세 번째 섹션은 객체 잠금 테이블의 내용을 포함한다. 이것은 어떤 객체에 대해서 어떤 클라이언트가 어떤 모드로 잠금을 가지고 있는지, 어떤 객체에 대해서 어떤 클라이언트가 어떤 모드로 기다리고 있는지를 보여준다. 객체 잠금 테이블 결과물의 첫 부분에는 얼마나 많은 객체가 잠금되었는지가 출력된다. 그리고 현재 메모리에 할당된 잠금의 개수와 크기가 출력된다. 성능을 위하여 잠금을 바로 해제하지 않고 재사용하지만, 메모리에 할당된 잠금 개수가 **lock\_escalation** 시스템 파라미터값을 넘을 경우에는 해당 잠금은 트랜잭션이 끝나는 시점에 메모리에서 해제된다. 관련 시스템 파라미터인 **lock\_escalation**에 대한 설명은 [동시성/잠금 파라미터](#)를 참고한다.

```
Object Lock Table:
 Current number of objects which are locked = 100
 Current number of objects which are allocated = 1000
 Current size of objects which are allocated = 242K
```

**cubrid lockdb**는 잠금을 획득한 각 객체의 OID, 객체 타입 및 테이블명을 출력한다. 또한 객체에 대한 잠금을 보유한 트랜잭션 수( *Num holders*), 잠금을 보유하지만 잠금을 상위 잠금으로 변환(예: **SCH\_S\_LOCK**에서 **SCH\_M\_LOCK**으로의 변환)할 수 없어 차단된 트랜잭션 수( *Num blocked-holders*) 및 객체의 잠금을 기다리는 다른 트랜잭션의 수( *Num waiters*)를 출력한다. 잠금을 보유한 클라이언트 트랜잭션, 차단된 클라이언트 트랜잭션 및 대기 중인 클라이언트 트랜잭션의 목록도 출력한다. 객체 타입이 클래스가 아닌 행에서는 MVCC 정보도 표시된다.

다음 예제는 OID(016215)인 클래스 타입의 객체를 나타내며, 트랜잭션 1은 **IX\_LOCK**을 보유하고 트랜잭션 2는 **SCH\_S\_LOCK**을 보유한 상태에서 **SCH\_M\_LOCK**으로 변환할 수 없어 차단된 상태를 보여준

다. 또한 트랜잭션 3은\*\*SCH\_S\_LOCK\*\*을 기다리지만 트랜잭션 2가 **SCH\_M\_LOCK**을 기다리기 때문에 차단된 상태를 보여준다.

```
OID = 0|62|5
Object type: Class = athlete.
Num holders = 1, Num blocked-holders= 1, Num waiters = 1
LOCK HOLDERS :
 Tran_index = 1, Granted_mode = IX_LOCK, Count = 1, Nsubgranules
= 1
BLOCKED LOCK HOLDERS :
 Tran_index = 2, Granted_mode = SCH_S_LOCK, Count = 1,
Nsubgranules = 0
 Blocked_mode = SCH_M_LOCK
 Start_waiting_at = Wed Feb 3 14:44:31 2016
 Wait_for_secs = -1
LOCK WAITERS :
 Tran_index = 3, Blocked_mode = SCH_S_LOCK
 Start_waiting_at = Wed Feb 3 14:45:14 2016
 Wait_for_secs = -1
```

다음 예는 **X\_LOCK** 을 보유한 트랜잭션 1에 의해서 삽입된 OID가 ( 2 | 50 | 1)인 클래스의 인스턴스를 보여준다. 트랜잭션1이 삽입한 인스턴스를 수정하기 위해 트랜잭션2가 **X\_LOCK** 을 기다리는 것을 보여준다. **-c** 옵션을 사용하면 경합이 발생한 잠금만 확인할 수 있다.

```
OID = 2|50|1
Object type: instance of class (0|62|5) = athlete.
MVCC info: insert ID = 6, delete ID = missing.
Num holders = 1, Num blocked-holders= 1, Num waiters = 1
LOCK HOLDERS :
 Tran_index = 1, Granted_mode = X_LOCK, Count = 1
LOCK WAITERS :
 Tran_index = 2, Blocked_mode = X_LOCK
 Start_waiting_at = Wed Feb 3 14:45:14 2016
 Wait_for_secs = -1
```

*Granted\_mode* 는 습득한 잠금의 모드를 나타내며, *Blocked\_mode* 는 차단된 잠금의 모드를 나타낸다. *Start\_waiting\_at* 은 잠금이 요청된 시간을 나타내며, *Wait\_for\_secs* 는 잠금의 대기 시간을 나타낸다. *Wait\_for\_secs* 값은 시스템 파라미터인 **lock\_timeout** 에 의해 결정된다.

객체 타입이 클래스(테이블)인 경우 테이블의 특정 트랜잭션에서 획득한 레코드 잠금과 키 잠금의 합계를 나타내는 *Nsubgranules* 가 표시된다.

```
OID = 0| 62| 5
Object type: Class = athlete
Num holders = 2, Num blocked-holders= 0, Num waiters= 0
LOCK HOLDERS:
 Tran_index = 3, Granted_mode = IS_LOCK, Count = 2, Nsubgranules = 0
```

```
Tran_index = 1, Granted_mode = IX_LOCK, Count = 3, Nsubgranules = 1
Tran_index = 2, Granted_mode = IS_LOCK, Count = 2, Nsubgranules = 1
```

## tranlist

**cubrid tranlist** 는 대상 데이터베이스의 트랜잭션 정보를 확인하는 유틸리티이다.

```
cubrid tranlist [options] database_name
```

옵션을 생략하면 각 트랜잭션에 대한 전체 정보를 출력한다.

“cubrid tranlist demodb”는 “cubrid killtran -q demodb”와 비슷한 결과를 출력하지만, “User name”과 “Host name”을 더 출력한다. “cubrid tranlist -s demodb”는 “cubrid killtran -d demodb”와 동일한 결과를 출력한다.

다음은 tranlist 출력 결과의 예이다.

```
$ cubrid tranlist demodb
```

| Tran index        | User name     | Host name                       | Process id |
|-------------------|---------------|---------------------------------|------------|
| Program name      | Query time    | Tran time                       | Wait for   |
| lock holder       | SQL_ID        | SQL Text                        |            |
| 1(ACTIVE)         | public        | test-server                     | 1681       |
| broker1_cub_cas_1 |               | 0.00                            |            |
| 0.00              | -1            | *** empty ***                   |            |
| 2(ACTIVE)         | public        | test-server                     | 1682       |
| broker1_cub_cas_2 |               | 0.00                            |            |
| 0.00              | -1            | *** empty ***                   |            |
| 3(ACTIVE)         | public        | test-server                     | 1683       |
| broker1_cub_cas_3 |               | 0.00                            |            |
| 0.00              | -1            | *** empty ***                   |            |
| 4(ACTIVE)         | public        | test-server                     | 1684       |
| broker1_cub_cas_4 |               | 1.80                            | 1.80       |
| 3, 2, 1           | e5899a1b76253 | update ta set a = 5 where a > 0 |            |
| 5(ACTIVE)         | public        | test-server                     | 1685       |
| broker1_cub_cas_5 |               | 0.00                            |            |
| 0.00              | -1            | *** empty ***                   |            |

```
SQL_ID: e5899a1b76253
```

```
Tran index : 4
update ta set a = 5 where a > 0
```

위의 예는 3개의 트랜잭션이 각각 INSERT문을 실행 중일 때 또 다른 트랜잭션에서 UPDATE문의 실행을 시도한다. 위에서 Tran index가 4인 update문은 3,2,1(Wait for lock holder)번의 트랜잭션이 종료되기를 대기하고 있다.

화면에 출력되는 질의문(SQL Text)은 질의 계획 캐시에 저장되어 있는 것을 보여준다. 질의 수행이 완료되면 **empty** 로 표시된다.

각 칼럼의 의미는 다음과 같다.

- Tran index: 트랜잭션 인덱스
- User name: 데이터베이스 사용자 이름
- Host name: 해당 트랜잭션이 수행되는 CAS의 호스트 이름
- Process id: 클라이언트 프로세스 ID
- Program name: 클라이언트 프로그램 이름
- Query time: 수행중인 질의의 총 수행 시간(단위: 초)
- Tran time: 현재 트랜잭션의 총 수행 시간(단위: 초)
- Wait for lock holder: 현재 트랜잭션이 락 대기중이면 해당 락을 소유하고 있는 트랜잭션의 리스트
- SQL\_ID: SQL Text에 대한 ID. cubrid killtran 명령의 **--kill-sql-id** 옵션에서 사용될 수 있다.
- SQL Text: 수행중인 질의문(최대 30자)

“Tran index”에 보여지는 transaction 상태 메시지는 다음과 같다.

- ACTIVE : 활성
- RECOVERY : 복구중인 트랜잭션
- COMMITTED : 커밋완료되어 종료될 트랜잭션
- COMMITTING : 커밋중인 트랜잭션
- ABORTED : 롤백되어 종료될 트랜잭션
- KILLED : 서버에 의해 강제 종료 중인 트랜잭션

다음은 **cubrid tranlist** 에 대한 [options]이다.

```
-s, --summary 질의정보 및 락 정보 생략
--sort-key=NUMBER 정렬할 컬럼번호
--reverse 내림차순 정렬수행
```

### **-s, --summary**

요약 정보만 출력한다(질의 수행 정보 또는 잠금 관련 정보를 생략).

```
$ cubrid tranlist -s demodb
```

| Tran index<br>id  | User name<br>Program name | Host name   | Process |
|-------------------|---------------------------|-------------|---------|
| -----             |                           |             |         |
| -----             |                           |             |         |
| 1(ACTIVE)         | public                    | test-server | 1681    |
| broker1_cub_cas_1 |                           |             |         |
| 2(ACTIVE)         | public                    | test-server | 1682    |
| broker1_cub_cas_2 |                           |             |         |
| 3(ACTIVE)         | public                    | test-server | 1683    |
| broker1_cub_cas_3 |                           |             |         |
| 4(ACTIVE)         | public                    | test-server | 1684    |
| broker1_cub_cas_4 |                           |             |         |
| 5(ACTIVE)         | public                    | test-server | 1685    |
| broker1_cub_cas_5 |                           |             |         |
| -----             |                           |             |         |
| -----             |                           |             |         |

### **--sort-key=NUMBER**

해당 NUMBER 위치의 컬럼에 대해 오름차순으로 정렬된 값을 출력한다. 컬럼의 타입이 숫자인 경우는 숫자로 정렬되고, 그렇지 않은 경우 문자열로 정렬된다. 생략되면 “Tran index”에 대한 정렬값을 보여준다.

다음은 네번째 컬럼인 “Process id”를 지정하여 정렬한 정보를 출력하는 예이다.

```
$ cubrid tranlist --sort-key=4 demodb
```

| Tran index        | User name  | Host name     | Process id |
|-------------------|------------|---------------|------------|
| Program name      | Query time | Tran time     | Wait for   |
| lock holder       | SQL_ID     | SQL Text      |            |
| -----             |            |               |            |
| -----             |            |               |            |
| -----             |            |               |            |
| 1(ACTIVE)         | public     | test-server   | 1681       |
| broker1_cub_cas_1 |            | 0.00          |            |
| 0.00              | -1         | *** empty *** |            |
| 2(ACTIVE)         | public     | test-server   | 1682       |
| broker1_cub_cas_2 |            | 0.00          |            |
| 0.00              | -1         | *** empty *** |            |

```

3(ACTIVE) public test-server 1683
broker1_cub_cas_3 0.00
0.00 -1 *** empty ***
4(ACTIVE) public test-server 1684
broker1_cub_cas_4 1.80
1.80 3, 1, 2 e5899a1b76253 update ta set
a = 5 where a > 0
5(ACTIVE) public test-server 1685
broker1_cub_cas_5 0.00
0.00 -1 *** empty ***

```

```


SQL_ID: e5899a1b76253
Tran index : 4
update ta set a = 5 where a > 0

```

#### --reverse

역순으로 정렬된 값을 출력한다.

다음은 “Tran index”의 역순으로 정렬한 정보를 출력하는 예이다.

```

Tran index User name Host name Process id
Program name Query time Tran time Wait for
lock holder SQL_ID SQL Text

5(ACTIVE) public test-server 1685
broker1_cub_cas_5 0.00
0.00 -1 *** empty ***
4(ACTIVE) public test-server 1684
broker1_cub_cas_4 1.80 1.80
3, 2, 1 e5899a1b76253 update ta set a = 5 where a > 0
3(ACTIVE) public test-server 1683
broker1_cub_cas_3 0.00
0.00 -1 *** empty ***
2(ACTIVE) public test-server 1682
broker1_cub_cas_2 0.00
0.00 -1 *** empty ***
1(ACTIVE) public test-server 1681
broker1_cub_cas_1 0.00
0.00 -1 *** empty ***

```

```


SQL_ID: e5899a1b76253

```

```
Tran index : 4
update ta set a = 5 where a > 0
```

## killtran

**cubrid killtran** 은 대상 데이터베이스의 트랜잭션을 확인하거나 특정 트랜잭션을 강제 종료하는 유틸리티로서, **DBA** 사용자만 트랜잭션을 제거할 수 있다.

```
cubrid killtran [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **killtran**: 지정된 데이터베이스에 대해 트랜잭션을 관리하는 명령어이다.
- *database\_name*: 대상 데이터베이스의 이름이다.

[options]에 따라 특정 트랜잭션을 지정하여 제거하거나, 현재 활성화된 트랜잭션을 화면 출력할 수 있다. 옵션이 지정되지 않으면, **-d** 옵션이 기본으로 적용되어 모든 트랜잭션을 화면 출력하며, cubrid tranlist 명령에 **-s** 옵션을 준 것과 동일하다.

```
$ cubrid killtran demodb
```

| Tran index<br>name | User name | Host name | Process id | Program |
|--------------------|-----------|-----------|------------|---------|
| 1(ACTIVE)          | dba       | myhost    | 664        |         |
| cub_cas            |           |           |            |         |
| 2(ACTIVE)          | dba       | myhost    | 6700       |         |
| csql               |           |           |            |         |
| 3(ACTIVE)          | dba       | myhost    | 2188       |         |
| cub_cas            |           |           |            |         |
| 4(ACTIVE)          | dba       | myhost    | 696        |         |
| csql               |           |           |            |         |
| 5(ACTIVE)          | public    | myhost    | 6944       |         |
| csql               |           |           |            |         |

다음은 **cubrid killtran** 에 대한 [options]이다.

|                                           |                          |
|-------------------------------------------|--------------------------|
| <b>-i, --kill-transaction-index=INDEX</b> | INDEX로 지정된 해당 트랜잭션 종료    |
| <b>--kill-user-name=ID</b>                | ID로 지정되는 사용자의 모든 트랜잭션 종  |
| <b>료</b>                                  |                          |
| <b>--kill-host-name=HOST</b>              | HOST로 지정되는 클라이언트의 모든 트랜잭 |



**션 종료**

|                                        |                          |
|----------------------------------------|--------------------------|
| <code>--kill-program-name=NAME</code>  | NAME으로 지정되는 모든 트랜잭션 종료   |
| <code>--kill-sql-id=SQL_ID</code>      | SQL_ID에 해당하는 질의를 실행중인 모든 |
| <b>트랜잭션 종료</b>                         |                          |
| <code>-p, --dba-password=PASS</code>   | DBA 패스워드                 |
| <code>-q, --query-exec-info</code>     | 현재 활성화된 트랜잭션에 대한 질의 수행   |
| 정보를 보여줌                                |                          |
| <code>-d, --display-information</code> | 현재 활성화된 트랜잭션에 대한 정보 보여   |
| 줌                                      |                          |
| <code>-f, --force</code>               | 사용자 확인 없이 지정된 트랜잭션 종료    |

**-i, --kill-transaction-index=ID1,ID2,ID3**

지정한 인덱스에 해당하는 트랜잭션을 제거한다. 쉼표(,)로 구분하여 제거하고자 하는 트랜잭션 ID 여러 개를 지정할 수 있다. 제거할 트랜잭션 리스트에 유효하지 않은 트랜잭션 ID가 지정되면 무시된다.:

```
$ cubrid killtran -i 1,2 demodb
Ready to kill the following transactions:

Tran index User name Host name Process
id Program name

 1(ACTIVE) DBA myhost
15771 csq
 2(ACTIVE) DBA myhost
2171 csq

Do you wish to proceed ? (Y/N)y
Killing transaction associated with transaction index 1
Killing transaction associated with transaction index 2
```

**--kill-user-name=ID**

지정한 OS 사용자 ID에 해당하는 트랜잭션을 제거한다.

```
cubrid killtran --kill-user-name=os_user_id demodb
```

**--kill-host-name=HOST**

지정한 클라이언트 호스트의 트랜잭션을 제거한다.

```
cubrid killtran --kill-host-name=myhost demodb
```

**--kill-program-name=NAME**

지정한 이름의 프로그램에 해당하는 트랜잭션을 제거한다.

```
cubrid killtran --kill-program-name=cub_cas demodb
```

### --kill-sql-id=SQL\_ID

지정한 SQL ID에 해당하는 트랜잭션을 제거한다.

```
cubrid killtran --kill-sql-id=5377225ebc75a demodb
```

### -p, --dba-password=PASSWORD

**-i** 또는 **--kill** 옵션들이 사용될 경우에만 해당 옵션을 사용할 수 있다. 이 옵션 뒤에 오는 값은 DBA의 암호이며 생략하면 프롬프트에서 입력해야 한다.

### -q, --query-exec-info

cubrid tranlist 명령에서 "User name" 칼럼과 "Host name" 칼럼이 출력되지 않는다는 점만 다르다. tranlist를 참고한다.

### -d, --display-information

기본 지정되는 옵션으로 트랜잭션의 요약 정보를 출력한다. cubrid tranlist 명령의 -s 옵션을 지정하여 실행한 것과 동일한 결과를 출력한다. tranlist -s를 참고한다.

### -f, --force

중지할 트랜잭션을 확인하는 프롬프트를 생략한다.

```
cubrid killtran -f -i 1 demodb
```

## checkdb

**cubrid checkdb** 유틸리티는 데이터베이스를 확인하기 위해 사용된다. **cubrid checkdb** 유틸리티를 사용하면 인덱스와 다른 데이터 구조를 확인하기 위해 데이터와 로그 볼륨의 내부적인 물리적 일치를 확인할 수 있다. 만일 **cubrid checkdb** 유틸리티의 실행 결과가 불일치로 나온다면 **--repair** 옵션으로 자동 수정을 시도해 보아야 한다.

```
cubrid checkdb [options] database_name [schema_name.table_name [{,
schema_name.table_name}]]
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티
- **checkdb**: 대상 데이터베이스에 대하여 데이터의 일관성(consistency)을 확인하는 명령
- *database\_name*: 일관성을 확인하거나 복구하려는 데이터베이스 이름
- *schema\_name*: 테이블의 스키마 이름

· *table\_name*: 일관성을 확인하거나 복구하려는 테이블 이름을 나열한다.

다음은 **cubrid checkdb**에 대한 [options]이다.

|                                    |                        |
|------------------------------------|------------------------|
| <b>-S, --SA-mode</b>               | 독립 모드 실행               |
| <b>-C, --CS-mode</b>               | 클라이언트 서버 모드 실행         |
| <b>-r, --repair</b>                | 불일치 발견 시 복구            |
| <b>--check-prev-link</b>           | 인덱스 링크의 오류를 검사         |
| <b>--repair-prev-link</b>          | 인덱스 링크가 잘못된 경우 복구 수행   |
| <b>-i, --input-class-file=FILE</b> | 클래스 이름이 나열된 입력 파일 명시   |
| <b>--check-file-tracker</b>        | 파일 트랙커의 모든 파일의 페이지를 검사 |
| <b>--check-heap</b>                | 모든 힙 파일 검사             |
| <b>--check-catalog</b>             | 카탈로그 정보에 일관성 검사        |
| <b>--check-btree</b>               | 모든 인덱스 유효 검사           |
| <b>--check-class-name</b>          | 클래스 정보의 일치 여부 검사       |
| <b>--check-btree-entries</b>       | 모든 인덱스 엔트리의 일관성 검사     |
| <b>-I, --index-name=INDEX_NAME</b> | 검사할 인덱스 지정             |

### **-S, --SA-mode**

서버 프로세스를 구동하지 않고 데이터베이스에 접근하는 독립 모드(standalone)로 작업하기 위해 지정되며, 인수는 없다. **-S** 옵션을 지정하지 않으면, 시스템은 클라이언트/서버 모드로 인식한다.

```
cubrid checkdb -S demodb
```

### **-C, --CS-mode**

서버 프로세스와 클라이언트 프로세스를 각각 구동하여 데이터베이스에 접근하는 클라이언트/서버 모드로 작업하기 위한 옵션이며, 인수는 없다. **-C** 옵션을 지정하지 않더라도 시스템은 기본적으로 클라이언트/서버 모드로 인식한다.

```
cubrid checkdb -C demodb
```

### **-r, --repair**

데이터베이스의 일관성에 문제가 발견되었을 때 복구를 수행한다.

```
cubrid checkdb -r demodb
```

### **--check-prev-link**

인덱스의 앞을 가리키는 링크(previous link)에 오류가 있는지를 검사한다.

```
$ cubrid checkdb --check-prev-link demodb
```

### **--repair-prev-link**

인덱스의 앞을 가리키는 링크(previous link)에 오류가 있으면 복구한다.

```
$ cubrid checkdb --repair-prev-link demodb
```

### **-i, --input-class-file=FILE**

**-i** *FILE* 옵션을 지정하거나, 데이터베이스 이름 뒤에 테이블의 이름을 나열하여 일관성 확인 또는 복구 대상을 한정할 수 있다. 두 가지 방법을 같이 사용할 수도 있으며, 대상을 지정하지 않으면 전체 데이터베이스를 대상으로 일관성을 확인하거나 복구를 수행한다. 특정 대상이 지정되지 않으면 전체 데이터베이스가 일관성 확인 또는 복구의 대상이 된다.

```
cubrid checkdb demodb public.table_1 public.table_2
cubrid checkdb -r demodb public.table_1 public.table_2
cubrid checkdb -r -i table_list.txt demodb public.table_1
public.table_2
```

**-i** 옵션으로 지정하는 테이블 목록 파일은 공백, 탭, 줄바꿈, 쉼표로 테이블 이름을 구분한다. 다음은 테이블 목록 파일의 예로, t1부터 t10까지를 모두 일관성 확인 또는 복구를 위한 테이블로 인식한다.

```
t1 t2 t3,t4 t5
t6, t7 t8 t9

t10
```

### **--check-file-tracker**

파일 트랙커(tracker)의 모든 파일의 페이지에 검사를 수행한다.

### **--check-heap**

모든 힙 파일에 대해 검사를 수행한다.

### **--check-catalog**

카탈로그 정보에 대한 일관성 검사를 수행한다.

### **--check-btree**

모든 B-트리 인덱스에 대한 유효 검사를 수행한다.

### **--check-class-name**

클래스 이름 해시 테이블과 힙 파일로 부터 가져온 클래스 정보(class oid)의 일치 여부 검사를 수행한다.

### **--check-btree-entries**

모든 B-트리 영역(entry)의 일관성 검사를 수행한다.

### **-I, --index-name=INDEX\_NAME**

검사 대상 테이블에 대해 해당 옵션으로 명시한 인덱스가 유효한지 검사한다. 이 옵션을 사용하면 힙 파일에 대한 유효 검사는 하지 않는다. 이 옵션을 사용할 때는 하나의 테이블과 하나의 인덱스만을 허용하며, 테이블 이름을 입력하지 않거나 두 개 이상의 테이블 이름을 입력하면 에러를 반환한다.

## diagdb

**cubrid diagdb** 유틸리티를 이용해 다양한 데이터베이스 내부 정보를 확인할 수 있다. **cubrid diagdb** 유틸리티가 제공하는 정보들은 현재 데이터베이스의 상태를 진단하거나 문제를 파악하는데 도움이 된다.

```
cubrid diagdb [option] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **diagdb**: CUBRID에 저장되는 바이너리 형태의 파일 정보를 텍스트 형태로 출력하여 현재 데이터베이스의 저장 상태를 확인할 때 사용하는 명령어이다. 데이터베이스가 구동 정지 상태인 경우에만 정상적으로 수행된다. 전체를 확인하거나 옵션을 사용하여 파일 테이블, 파일 용량, 힙 용량, 클래스 이름, 디스크 비트맵을 선택해 확인할 수 있다.
- *database\_name*: 내부 정보를 확인하려는 데이터베이스 이름이다.

다음은 **cubrid diagdb** 에서 사용하는 [option]이다.

```
-d, --dump-type=TYPE 출력 범위 지정
-o, --output-file=FILE 출력 메시지를 재지정할 파일
--emergency 비상시 재시작
```

### -d, --dump-type=TYPE

데이터베이스의 전체 파일에 대한 기록 상태를 출력할 때 출력 범위를 지정한다. 생략하면 기본값인 -1이 지정된다.

```
cubrid diagdb -d 1 demodb
```

**-d** 옵션에 적용되는 타입은 모두 9가지로, 그 종류는 다음과 같다.

| 타입 | 설명 |
|----|----|
|----|----|

|    |                     |
|----|---------------------|
| -1 | 전체 데이터베이스 정보를 출력한다. |
|----|---------------------|

| 타입 | 설명                |
|----|-------------------|
| 1  | 파일 테이블 정보를 출력한다.  |
| 2  | 파일 용량 정보를 출력한다.   |
| 3  | 힙 용량 정보를 출력한다.    |
| 4  | 인덱스 용량 정보를 출력한다.  |
| 5  | 클래스 이름 정보를 출력한다.  |
| 6  | 디스크 비트맵 정보를 출력한다. |
| 7  | 카탈로그 정보를 출력한다.    |
| 8  | 로그 정보를 출력한다.      |
| 9  | 힙(heap) 정보를 출력한다. |

### **-o, --output-file=FILE**

**-o** 옵션은 데이터베이스의 서버/클라이언트 프로세스에서 사용되는 파라미터 정보를 지정된 파일로 저장하는 데 사용된다. 이 파일은 현재 디렉터리에 생성된다. **-o** 옵션을 지정하지 않으면 콘솔 화면에 메시지가 표시된다.

```
cubrid diagdb -d8 -o logdump_output demodb
```

### **--emergency**

복구를 생략(suppression)하려면 **--emergency** 옵션을 사용한다. 이 옵션은 디버깅에만 사용해야 한다. 복구 이슈가 있는 경우 이 옵션을 사용하기 전에 데이터베이스를 백업할 것을 권장한다.

## paramdump

**cubrid paramdump** 유틸리티는 서버/클라이언트 프로세스에서 사용하는 파라미터 정보를 출력한다.

```
cubrid paramdump [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **paramdump**: 서버/클라이언트 프로세스에서 사용하는 파라미터 정보를 출력하는 명령이다.
- *database\_name*: 파라미터 정보를 출력할 데이터베이스 이름이다.

다음은 **cubrid paramdump** 에서 사용하는 [options]이다.

|                               |                                |
|-------------------------------|--------------------------------|
| <b>-o, --output-file=FILE</b> | 출력 메시지를 재지정할 파일                |
| <b>-b, --both</b>             | 서버/클라이언트 프로세스에 사용하는 파라미터 정보 출력 |
| <b>-S, --SA-mode</b>          | 독립 모드 실행                       |
| <b>-C, --CS-mode</b>          | 클라이언트 서버 모드 실행                 |

### **-o, --output-file=FILE**

데이터베이스의 서버/클라이언트 프로세스에서 사용하는 파라미터 정보를 지정된 파일에 저장하는 옵션이며, 파일은 현재 디렉터리에 생성된다. **-o** 옵션이 지정되지 않으면 메시지는 콘솔 화면에 출력한다.

```
cubrid paramdump -o db_output demodb
```

### **-b, --both**

데이터베이스의 서버/클라이언트 프로세스에서 사용하는 파라미터 정보를 콘솔 화면에 출력하는 옵션이며, **-b** 옵션을 사용하지 않으면 서버 프로세스의 파라미터 정보만 출력한다.

```
cubrid paramdump -b demodb
```

### **-S, --SA-mode**

독립 모드에서 서버 프로세스의 파라미터 정보를 출력한다.

```
cubrid paramdump -S demodb
```

### **-C, --CS-mode**

클라이언트-서버 모드에서 서버 프로세스의 파라미터 정보를 출력한다.

```
cubrid paramdump -C demodb
```

## tde

**cubrid tde** 유틸리티는 데이터베이스의 TDE 암호화를 관리하기 위하여 사용되며, **DBA** 사용자만 수행할 수 있다. **cubrid tde** 유틸리티를 사용하면 데이터베이스에 등록된 키를 변경할 수 있고, 키 파일에 새로운 키를 안정적으로 추가하고 제거할 수 있다. 또한, 지금까지 키 파일에 추가된 키들과 데이터베이스에 등록된 키가 무엇인지 조회할 수 있다. 자세한 내용은 [TDE \(Transparent Data Encryption\)](#) 을 참고한다.

```
cubrid tde <operation> [option] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티
- **tde**: 대상 데이터베이스에 적용된 TDE 암호화에 대한 관리 도구
- *operation*: 도구를 통한 수행할 작업을 지정한다. 키 조회, 키 추가, 키 제거, 키 변경 네 가지 종류가 있으며 하나의 작업이 주어져야 한다.
- *database\_name*: TDE 관리 작업을 수행하려는 데이터베이스 이름

다음은 **cubrid tde** 에 대한 오퍼레이션 (operation)에 대한 설명이다. 다음 중 하나의 작업이 주어져야 한다.

|                                   |                                                            |
|-----------------------------------|------------------------------------------------------------|
| <b>-s, --show-keys</b>            | 데이터베이스에 등록된 키, 키 파일 ( <code>_keys</code> )내의 키들에 대한 정보를 출력 |
| <b>-n, --generate-new-key</b>     | 키 파일 ( <code>_keys</code> )에 새로운 키를 추가; 최대 개수: 128         |
| <b>-d, --delete-key=KEY_INDEX</b> | 키 파일 ( <code>_keys</code> )의 한 키를 제거                       |
| <b>-c, --change-key=KEY_INDEX</b> | 데이터베이스에 등록된 키를 키 파일 ( <code>_keys</code> )에 존재하는 다른 키로 변경  |

### -s, --show-keys

데이터베이스에 등록된 키와 키 파일 (`_keys`)내의 키들에 대한 정보를 출력한다.

```
$ cubrid tde -s testdb
Key File: /home/usr/CUBRID/databases/testdb/testdb_keys

The current key set on testdb:
Key Index: 2
Created on Fri Nov 27 11:14:54 2020
Set on Fri Nov 27 11:15:30 2020

Keys Information:
```



```
Key Index: 0 created on Fri Nov 27 11:11:27 2020
Key Index: 1 created on Fri Nov 27 11:14:47 2020
Key Index: 2 created on Fri Nov 27 11:14:54 2020
Key Index: 3 created on Fri Nov 27 11:14:55 2020
```

```
The number of keys: 4
```

**The current key** 정보는 현재 데이터베이스에 등록된 키의 정보이다. 키 파일내에서의 인덱스와 키의 생성시간, 등록된 시간을 출력해준다. 키 인덱스와 생성시간으로 등록된 키를 식별할 수 있고, 등록된 시간을 통해 키 변경 계획을 수립할 수 있다.

**Keys Information** 은 키 파일 내에서 생성되어 관리되고 있는 키들을 보여준다. 키 인덱스와 생성시간을 확인할 수 있다.

#### **-n, --generate-new-key**

키 파일 (\_keys)에 새로운 키를 추가한다 (최대 128개). 성공할 경우 추가된 키의 인덱스를 출력해주며, 이 인덱스는 이후 키를 변경하거나 제거할 때에 키를 식별하기 위하여 사용된다. 추가된 키들의 정보는 **--show-keys** 를 통해 확인할 수 있다.

```
$ cubrid tde -n testdb
Key File: /home/usr/CUBRID/databases/testdb/testdb_keys

SUCCESS: A new key has been generated - key index: 1 created on
Tue Dec 1 11:30:47 2020
```

#### **-d, --delete-key=KEY\_INDEX**

키 파일 (\_keys)에서 인덱스로 지정된 키 하나를 제거한다. 현재 데이터베이스에 등록된 키는 제거할 수 없다.

```
$ cubrid tde -d 1 testdb
Key File: /home/usr/CUBRID/databases/testdb/testdb_keys

SUCCESS: The key (index: 1) has been deleted
```

#### **-c, --change-key=KEY\_INDEX**

데이터베이스에 등록된 키를 키 파일 (\_keys)에 존재하는 다른 키로 변경한다. 변경 시에 이전에 등록된 키와 새로 등록하려는 키가 모두 존재해야 한다.

```
$ cubrid tde -c 2 testdb
Key File: /home/usr/CUBRID/databases/testdb/testdb_keys

Trying to change the key from the key (index: 0) to the key
(index: 2)..
SUCCESS: The key has been changed from the key (index: 0) to the
key (index: 2)
```

데이터베이스에 등록된 키를 변경하기 위해서는 먼저 **--generate-new-key** 를 통해 등록할 키를 생성해야 한다. 사용자는 키 변경을 위해 새로운 키를 생성할 수 있고, 미리 여러 키를 생성해 두고 보안 계획에 따라 키를 변경할 수 있다.

다음은 **cubrid tde** 에서 사용하는 [options]이다.

#### **-p, --dba-password=PASSWORD**

이 옵션 뒤에 오는 값은 **DBA** 의 암호이며 생략하면 프롬프트에서 입력해야 한다.

## vacuumdb

**cubrid vacuumdb** 유틸리티는 삭제된 레코드와 불필요한 mvcc 관련 정보를 정리하기 위해 vacuum을 수행하거나, vacuum 관련 상태 정보를 얻기 위해 사용된다.

```
cubrid vacuumdb [options] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
- **vacuumdb**: 데이터베이스의 레코드에서 삭제된 레코드와 불필요한 mvcc 관련 정보 정리 및 관련 정보를 확인하는 유틸리티이다.
- *database\_name*: vacuum을 수행할 데이터베이스 이름이다.

다음은 **cubrid vacuumdb**에서 사용할 수 있는 옵션들이다.

|                               |                 |
|-------------------------------|-----------------|
| <b>-o, --output-file=FILE</b> | 출력 메시지를 재지정할 파일 |
| <b>--dump</b>                 | 현재 vacuum 상태 덤프 |
| <b>-S, --SA-mode</b>          | 독립 모드 실행        |
| <b>-C, --CS-mode</b>          | 클라이언트 서버 모드 실행  |

#### **-o, --output-file=FILE**

이 옵션은 vacuum에 대한 상태 정보를 저장할 파일명을 지정한다. 파일은 현재 디렉터리에 생성된다. **-o** 옵션을 지정하지 않으면 콘솔 화면에 메시지가 표시된다.

```
cubrid vacuumdb -o db_output demodb
```

#### **--dump**

이 옵션은 vacuum에 대한 상태 정보를 표시하는 데 사용된다. 현재는 vacuum에서 참조하는 첫 번째 로그 페이지 ID와 로그 볼륨 이름을 표시한다. 사용자는 첫 번째 로그 페이지 ID의 변경을 통해 vacuum 작업의 진행 상황을 확인할 수 있으며, vacuum 작업을 위해 유지해야 하는 최소 로그 볼륨을 확인할 수 있다. 독립형 모드와 클라이언트/서버 모드 모두에서 사용할 수 있다. **-C** 옵션과 함께 지정되면 인덱스 vacuum시 인덱스 단말 노드 (leaf node) 의 임계값 이상

의 **오버플로우 페이지** 읽기가 발생한 인덱스에 대한 정보를 함께 표시 한다. 이와 관련된 시스템 파라미터는 **vacuum\_ovfp\_check\_duration**과 **vacuum\_ovfp\_check\_threshold**가 있다.

```
cubrid vacuumdb --dump demodb
```

```
$ cubrid vacuumdb -C --dump demodb

*** Vacuum Dump ***
First log page ID referenced = 85653 (in demodb_lgar013)

*** Exceeding read threshold (1000 pages) for OID overflow pages
(OVFP), Since 09/04/23 08:53:50 ***
 Class name Index name Count Num
of OVFP recent read Max num of OVFP read
=====
=====
 dba.tm v_idx 272451
1089 (09/04/23 10:49:52) 1478 (09/04/23 10:28:46)
 dba.tbl idx 322728
1023 (09/04/23 09:53:47) 1475 (09/04/23 09:29:29)

$
```

- Exceeding read threshold (**1000** pages) : vacuum\_ovfp\_check\_threshold 설정값을 보여준다.
- Since **09/04/23 08:53:50** : OVFP 모니터링이 시작된 시각 정보를 보여준다.
- Class name: 클래스 이름(테이블 이름), 소유자 이름을 포함합니다.
- Index name: 인덱스 이름
- Count: 인덱스 베akup(vacuum) 처리시 임계값(vacuum\_ovfp\_check\_threshold)을 초과해서 인덱스 오버플로우 페이지를 읽은 횟수
- Num of OVFP recent read: 인덱스 베akup 처리시 임계값(vacuum\_ovfp\_check\_threshold) 이상의 인덱스 오버플로우 페이지를 읽은 가장 최근의 기록
- Max num of OVFP read: 인덱스 베akup 작업시 임계값(vacuum\_ovfp\_check\_threshold) 이상의 인덱스 오버플로우 페이지를 최대로 읽은 시점의 기록

### **-S, --SA-mode**

이 옵션은 독립 실행형 모드인 데이터베이스에서 vacuumdb 동작을 수행하도록 한다.

```
cubrid vacuumdb -S demodb
```

**-C, --CS-mode**

이 옵션은 클라이언트/서버 모드에서 vacuumdb 동작을 수행하도록 한다. 이 모드에서는 vacuum은 수행할 수 없고 -dump 옵션과 함께 사용해야 한다.

```
cubrid vacuumdb -C demodb
```

## flashback

**cubrid flashback** 유틸리티는 커밋된 트랜잭션을 되돌릴 수 있는 SQL 구문을 제공하며, DBA 사용자만 수행할 수 있다. **cubrid flashback** 을 수행하기 위해서는 시스템 파라미터 **supplemental\_log** 를 반드시 0보다 큰 값으로 설정해야 하며, **supplemental\_log** 가 설정된 후에 실행되는 DML에 대해서만 지원한다. 이 유틸리티는 클라이언트/서버 모드에서만 동작하고 동시에 두 개 이상 수행할 수 없다::

```
cubrid flashback [options] database_name owner_name.class_name1
[owner_name.class_name2, ...]
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티
- **flashback**: 커밋된 특정 트랜잭션을 되돌릴 수 있도록 SQL 구문을 제공하는 도구
- **database\_name**: 되돌리고자 하는 트랜잭션이 수행된 데이터베이스 이름
- **owner\_name.class\_name**: 되돌리고자 하는 테이블명 리스트. 테이블명을 명시할 때는 해당 테이블의 소유자명도 함께 명시해야 한다.

다음 예제에서는 사용자가 “cubrid flashback demodb dba.tbl”를 수행했을 때 출력되는 결과를 보여준다.

```
$ csql -u dba demodb

csql> CREATE TABLE tbl (a INT);
csql> INSERT INTO tbl VALUES (10);

$ cubrid flashback demodb dba.tbl

Flashback Summary
Number of Transaction: 1
Start date - End date: 03-05-2022:10:52:56 - 03-05-2022:11:02:56
Transaction id User name Start
time End time Num_insert Num_update
Num_delete Table
 135 DBA
03-05-2022:11:02:40 03-05-2022:11:02:41 1
0 0 dba.tbl
```

```
Enter transaction id (press -1 to quit): 135
delete from [dba.tbl] where [a] = 10 limit 1;
```

위의 예에서 **cubrid flashback** 을 실행하면 지정된 기간 내에 수행된 트랜잭션에 대한 정보를 표시한다. 사용자가 기간을 지정하지 않으면 현재 시각으로부터 10분 전에 수행된 트랜잭션들부터 표시된다. 사용자가 트랜잭션 식별자를 선택하면, 선택된 트랜잭션이 실행한 DML을 되돌릴 수 있도록 SQL 구문을 제공한다. 사용자는 300초 이내에 트랜잭션 식별자를 선택해야 하며, 해당 제한 시간은 시스템 파라미터 **flashback\_timeout** 을 통해 조절할 수 있다.

**Flashback Summary** 에 표시된 각 칼럼의 의미는 다음과 같다.

- Transaction id : 트랜잭션 식별자
- User name : 트랜잭션을 수행한 사용자
- Start time : 트랜잭션을 시작한 시간 (추정시간)
- End time : 트랜잭션이 완료된 시간
- Num\_insert : 트랜잭션 내에서 수행된 삽입(insert) 연산 횟수
- Num\_update : 트랜잭션 내에서 수행된 수정(update) 연산 횟수
- Num\_delete : 트랜잭션 내에서 수행된 삭제(delete) 연산 횟수
- Table : 트랜잭션 내에서 DML이 수행된 테이블명 리스트

#### 참고

**cubrid flashback** 을 통해 되돌리려는 테이블이 **트리거 이벤트 대상** 일 경우, 사용자가 의도하지 않은 결과를 얻을 수 있다. **cubrid flashback** 을 통해 테이블에 대한 변경 사항을 되돌리기 전에 트리거를 비활성화하는 것이 좋다. 자세한 내용은 **ALTER TRIGGER** 을 참조한다.

#### 참고

**cubrid flashback** 은 복합 데이터 타입인 SET, MULTiset, LIST, JSON을 지원하지 않는다. **cubrid flashback** 수행 시 대상 클래스가 미지원 데이터 타입을 컬럼으로 갖는 경우 해당 컬럼값은 null로 출력된다. 또한, LOB 타입의 컬럼은 데이터가 아닌 해당 파일의 위치 정보(LOB locator)만 출력된다.

다음은 **cubrid flashback** 에서 사용하는 [options]이다.

```

-o, --output-file=FILE 출력 메시지를 FILE로 리디렉션합니다.
-u, --user=ID 플래시백할 SQL 문을 실행한 사용자를 지정합니다
-p, --dba-password DBA 사용자의 비밀번호
-s, --start-date=DATE 플래시백의 시작 날짜를 지정합니다.
-e, --end-date=DATE 플래시백의 종료 날짜를 지정합니다.
--detail 각 SQL 문에 대한 자세한 정보를 출력합니다.
--oldest 시작 시간부터 종료 시간까지 수행된 SQL문에 대하
여 시간 순으로 플래시백을 수행합니다.

```

### **-o, --output-file=FILE**

특정 트랜잭션을 되돌리기 위한 SQL 구문들을 지정된 파일에 저장하는 옵션이다. **-o** 옵션을 지정하지 않으면 콘솔 화면에 메시지가 표시된다.

```
cubrid flashback -o db_output demodb dba.tbl
```

### **-u, --user=ID**

트랜잭션을 수행한 사용자를 지정하며, 해당 사용자가 수행한 트랜잭션에 대해서만 **Flashback Summary** 에 표시된다. **-u** 옵션을 지정하지 않으면 모든 사용자에 의해 수행된 트랜잭션들이 표시된다.

```

$ csql -u public demodb

csql> CREATE TABLE tbl (a int);
csql> INSERT INTO tbl VALUES (10);

csql> CALL login ('dba', '') ON CLASS db_user;

csql> INSERT INTO public.tbl VALUES (20);

$ cubrid flashback -u public demodb dba.tbl

Flashback Summary
Number of Transaction: 1
Start date - End date: 03-05-2022:10:52:56 - 03-05-2022:11:02:56
Transaction id User name Start
time End time Num_insert Num_update
Num_delete Table
 135 PUBLIC
03-05-2022:11:02:40 03-05-2022:11:02:41 1
0 0 public.tbl
Enter transaction id (press -1 to quit): 135

delete from [public.tbl] where [a] = 10 limit 1;

```

### **-p, --dba-password=PASSWORD**

이 옵션 뒤에 오는 값은 **DBA** 의 암호이며 생략하면 프롬프트에서 입력해야 한다.

**-s, --start-date=DATE**

dd-mm-yyyy:hh:mi:ss (e.g. 28-04-2022:14:10:00) 형식으로 시작 날짜를 지정한다. 커밋된 트랜잭션 중에서 지정된 날짜 이후에 시작했거나 진행중인 트랜잭션들을 찾기 위해 사용된다.

```
cubrid flashback -s 28-04-2022:14:10:00 demodb dba.tbl
```

**-e, --end-date=DATE**

dd-mm-yyyy:hh:mi:ss (e.g. 28-04-2022:14:10:00) 형식으로 종료 날짜를 지정한다. 지정된 날짜 이전에 커밋된 트랜잭션들을 찾기 위해 사용된다.

시작 날짜를 지정하지 않으면 시작 날짜는 종료 날짜로부터 10분 전 시간으로 설정된다. 또한 종료 날짜가 지정되지 않으면 종료 날짜는 시작 날짜 10분 후 시간으로 설정된다. 만약 시작 날짜 및 종료 날짜가 지정되지 않으면, 10분 전부터 현재 시각까지 수행된 트랜잭션들이 표시된다.

```
cubrid flashback -e 28-04-2022:14:10:00 demodb dba.tbl
```

**--detail**

트랜잭션을 되돌리기 위한 각 SQL 구문들에 대해 자세한 정보를 보여준다. 트랜잭션 식별자, 트랜잭션을 수행한 사용자, 원본 SQL을 함께 제공한다. 원본 SQL은 사용자가 실행한 정확한 구문이 아니고, 단순히 행 단위로 실행되는 삽입/수정/삭제 문으로 사용자가 수행했던 구문을 나타낸 것이다. 예를 들어, INSERT ... SELECT 문은 여러 개의 삽입 문으로 표현된다. **--detail** 옵션을 지정하지 않으면, 트랜잭션을 되돌리기 위해 사용할 SQL 구문만 표시된다.

```
cubrid flashback --detail demodb dba.tbl

Flashback Summary
Number of Transaction: 1
Start date - End date: 03-05-2022:10:52:56 - 03-05-2022:11:02:56
Transaction id User name Start
time End time Num_insert Num_update
Num_delete Table
 135 DBA
02:40 03-05-2022:11:02:41 1 0
0 dba.tbl
Enter transaction id (press -1 to quit): 135

[TRANSACTION ID] 135
[USER] DBA
[ORIGINAL] insert into [dba.tbl] values (10);
[FLASHBACK] delete from [dba.tbl] where [a] = 10 limit 1;
```

**--oldest**

지정된 트랜잭션 내에서 수행된 SQL 구문들을 시간 순서로 표시한다. **--oldest** 옵션을 지정하지 않으면, 트랜잭션 내에서 수행된 SQL 구문들을 시간 역순으로 표시한다.

```
cubrid flashback --oldest demodb dba.tbl
```

## memmon

**cubrid memmon** 유틸리티는 현재 서버 프로세스에 할당된 힙 메모리 사용량을 출력한다. 시스템 파라미터 **enable\_memory\_monitoring** 가 yes로 설정된 경우, 서버 메모리 모니터링 모듈은 힙 메모리 총 사용량과 메모리 할당이 발생한 큐브리드 소스 코드 및 라인 정보를 기준으로 세부적인 메모리 할당 정보를 추적 관리한다. 이를 통해 유틸리티를 실행하는 시점의 서버 힙 메모리 사용 현황을 확인할 수 있다.

```
cubrid memmon [option] database_name
```

- **cubrid**: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티
- **memmon**: 서버의 힙 메모리 사용 현황을 출력하는 유틸리티
- *database\_name*: 힙 메모리 사용 현황을 확인할 데이터베이스 이름

다음 예제에서는 사용자가 “cubrid memmon demodb”를 수행했을 때 출력되는 결과를 보여준다.

```
=====cubrid memmon=====
Server Name: demodb
Total Memory Usage: 935240660 Bytes (for meta info: 470300 Bytes)

File
Name
| Memory Usage(Ratio)
| storage/page_buffer.c:
4977
| 525315320 Bytes(56%)
| transaction/log_page_buffer.c:
584
| 262147560 Bytes(28%)
| ...
```

**cubrid memmon** 유틸리티를 실행하면 데이터베이스 이름, 힙 메모리 총 사용량, 할당된 힙 메모리를 추적 관리하기 위해 서버 내부적으로 유지되는 메타 정보의 메모리 사용량이 출력된다. 세부 항목으로는 힙 메모리 할당이 발생한 모든 파일과 라인 별[파일:라인] 메모리 사용량과 점유율이 사용량 기준으로 정렬되어 출력된다. 시스템 파라미터 **enable\_memory\_monitoring** 가 no로 설정된 경우 또는 -disable-force 옵션을 이용하여 메모리 모니터링 기능을 강제로 중지시킨 경우, 에러 메시지가 출력된다.



유틸리티가 출력하는 정보는 다음과 같은 의미를 갖는다.

- Server Name : 데이터베이스 이름
- Total Memory Usage : 힙 메모리 총 사용량
  - for meta info : 힙 메모리 총 사용량 중 메모리 사용량을 추적 관리하기 위해 사용되는 메타 정보의 메모리 사용량
- File Name : 힙 메모리 할당이 발생한 큐브리드 소스 코드 파일명과 라인 정보
- Memory Usage(Ratio) : 힙 메모리 사용량 (총 사용량 대비 점유율)

#### 참고

힙 메모리 할당이 발생했던 이력을 남기기 위해 힙 메모리 할당 해제가 완료되어 현재 메모리 사용량이 0이 되는 지점 또한 **cubrid memmon**의 출력 대상이 된다.

다음은 **cubrid memmon**에서 사용하는 [option]이다.

#### **-o, --output-file=FILE**

유틸리티 수행 결과를 지정한 파일에 저장하기 위한 옵션이다. **-o** 옵션을 지정하지 않으면 콘솔 화면에 메시지가 출력된다.

```
cubrid memmon -o memory_usage_output demodb
```

#### **--disable-force**

메모리 모니터링 기능을 강제로 중지하는 옵션이다. 이 옵션을 이용해 메모리 모니터링 기능을 강제로 중지시키면 **cubrid memmon** 유틸리티를 실행해도 서버의 힙 메모리 사용량은 출력되지 않는다. 메모리 사용량을 다시 추적하기 위해서는 서버를 반드시 재구동해야 한다.

```
cubrid memmon --disable-force demodb
```

## HA 명령어

**cubrid changemode** 유틸리티는 서버의 HA 모드 출력하거나 변경하는 유틸리티이다.

**cubrid applyinfo** 유틸리티는 HA 환경에서 트랜잭션 로그 반영 정보를 확인하는 유틸리티이다.

자세한 사용법은 [cubrid service에 HA 등록](#)을 참고한다.

## 로캘 명령어

**cubrid genlocale** 유틸리티는 사용하고자 하는 로캘(locale) 정보를 컴파일하는 유틸리티이다. 이 유틸리티는 **make\_locale.sh** (Windows는 **.bat**) 스크립트 내에서 실행된다.

**cubrid dumplocale** 유틸리티는 컴파일된 바이너리 로캘(CUBRID 로케일 라이브러리) 파일을 사용자가 읽을 수 있는 형식으로 콘솔에 출력한다. 출력 정보는 리다이렉션(redirection)을 사용해서 저장해 두는 것이 좋다.

**cubrid synccolldb** 유틸리티는 데이터베이스와 로캘 라이브러리 사이의 콜레이션 불일치 여부를 체크하고, 불일치하는 경우 동기화한다.

자세한 사용법은 [로캘 설정](#) 을 참고한다.

## 타임존 명령어

**cubrid gen\_tz** 유틸리티는 다음과 같이 두 가지 모드가 있다.

- **new** 모드는 tzdata 폴더에 저장된 IANA 타임존 데이터를 C 소스 코드 파일로 컴파일할 때 사용한다. 이후 이 파일은 **make\_tz.sh** (Linux) / **make\_tz.bat** (Windows) 스크립트를 통해 Linux용 .so 공유 라이브러리나 Windows용 .dll 라이브러리로 변환된다.
- **extend** 모드는 new 모드와 비슷하지만 타임존 데이터를 다른 버전으로 갱신하고 기존 데이터와의 호환성을 유지하려고 할 때 사용하며, 항상 데이터베이스명 인자와 함께 사용한다. 단순히 두 가지 버전의 타임존 데이터를 병합하여 이전 데이터와의 호환성을 유지할 수 없는 경우에는 데이터베이스 테이블의 데이터를 갱신하기 위해 이 모드를 사용한다. Linux에서는 **make\_tz.sh -g extend** 를 사용하고 Windows에서는 **make\_tz.bat /extend** 를 사용한다.

**cubrid dump\_tz** 유틸리티는 컴파일된 CUBRID 타임존 라이브러리 파일을 콘솔에서 사용자가 읽을 수 있는 형식으로 출력한다. 출력 정보는 리다이렉션(redirection)을 사용해서 저장해 두는 것이 좋다.

## 시스템 설정

이 장에서는 시스템의 전체적인 성능과 동작을 결정하는 시스템 파라미터의 설정 정보에 대해 설명하며, 데이터베이스 서버 및 브로커에 적용할 설정 파일의 사용법과 개별 파라미터의 의미에 대해 설명한다.

이 장에서 설명하는 주요 내용은 다음과 같다.

- 데이터베이스 서버 설정
- 브로커 설정
- HA 설정

## · SHARD 설정

# 데이터베이스 서버 설정

## 데이터베이스 서버 설정이 미치는 범위

CUBRID는 데이터베이스 서버, 브로커, CUBRID 매니저로 구성되며, 각 구성요소에 대한 설정 파일이 존재한다. 데이터베이스 서버의 시스템 파라미터 설정 파일은 **cubrid.conf**이며 **\$CUBRID/conf** 디렉터리에 위치한다. **cubrid.conf**에 설정되는 시스템 파라미터는 데이터베이스 시스템의 전체적인 성능과 동작에 영향을 준다. 따라서, 데이터베이스 서버 설정을 이해하는 작업은 매우 중요하다.

CUBRID 데이터베이스 서버는 클라이언트/서버 구조로 구성된다. 구체적으로 서버 라이브러리와 링크되어 있는 데이터베이스 서버 프로세스와 클라이언트 라이브러리와 링크되어 있는 브로커 프로세스로 나뉜다. 서버 프로세스는 데이터베이스 저장 구조를 관리하고 동시성과 트랜잭션 기능을 제공하며, 클라이언트 프로세스는 질의 실행을 위한 준비 단계와 객체 관리 및 스키마 관리 기능을 수행한다.

**cubrid.conf** 파일에서 설정할 수 있는 데이터베이스 서버의 시스템 파라미터는 적용되는 범위에 따라 클라이언트 파라미터, 서버 파라미터, 클라이언트/서버 파라미터로 구분할 수 있다. 클라이언트 파라미터는 브로커와 같은 클라이언트 프로세스에만 적용되는 파라미터이며, 서버 파라미터는 서버 프로세스의 동작에 영향을 주는 파라미터이다. 클라이언트/서버 파라미터는 서버와 클라이언트에 적용된다.

### 참고

#### **cubrid.conf** 파일의 위치와 적용

**cubrid.conf** 파일은 **\$CUBRID/conf** 디렉터리에 위치하며, 데이터베이스 별 설정은 **cubrid.conf** 파일 내에서 섹션으로 구분한다.

## 데이터베이스 서버 설정값 변경

### 환경 설정 파일 편집

**\$CUBRID/conf** 디렉터리에 있는 시스템 파라미터 설정 파일( **cubrid.conf** )을 직접 편집하여 시스템 파라미터를 추가 및 삭제할 수 있으며, 파라미터의 설정값을 변경할 수 있다.

설정 파일에서 파라미터를 설정할 때 파라미터 구문 규칙은 다음과 같다.

- 파라미터 이름은 대/소문자를 구분하지 않는다.

- 파라미터 이름과 설정값은 동일한 라인에 입력되어야 한다.
- 파라미터 값을 설정하기 위해 등호 기호(=)를 사용하며, 등호 기호의 양 옆에는 공백 문자를 사용할 수 있다.
- 파라미터 값이 문자열인 경우 따옴표 없이 문자열만 입력한다. 다만, 해당 문자열에 공백 문자가 포함된 경우에는 따옴표를 사용한다.

## SQL 문을 사용

SQL 문을 이용하여 CSQL 인터프리터나 CUBRID 매니저의 질의 편집기에서 시스템 파라미터의 값을 설정할 수 있다. 단, 갱신할 수 있는 파라미터는 한정되어 있으므로 주의한다. 갱신할 수 있는 파라미터는 [cubrid.conf 설정 파일과 기본 제공 파라미터](#)를 참고한다.

```
SET SYSTEM PARAMETERS 'parameter_name=value [{; name=value}...]'
```

*parameter\_name* 은 설정값 변경이 가능한 클라이언트 파라미터 이름이고, *value* 는 해당 파라미터의 값을 의미한다. 세미콜론(;)으로 구분하여 여러 개의 파라미터 값을 변경할 수 있다.

다음은 인덱스 스캔 작업의 결과를 OID 순으로 가져오고, CSQL 인터프리터에서 히스토리 내역으로 저장하는 질의 개수를 70개로 설정하는 예제이다.

```
SET SYSTEM PARAMETERS 'index_scan_in_oid_order=1;
csql_history_num=70';
```

*value* 에 **DEFAULT** 값을 주면 **call\_stack\_dump\_activation\_list** 파라미터를 제외하고 해당 파라미터를 기본값으로 재설정한다.

```
SET SYSTEM PARAMETERS 'lock_timeout=DEFAULT';
```

## CSQL 인터프리터의 세션 명령어 사용

CSQL 인터프리터 내에서 세션 명령어(**SET**)를 이용하여 시스템 파라미터의 값을 설정할 수 있다. 단, 갱신할 수 있는 파라미터는 한정되어 있으므로 주의한다. 갱신할 수 있는 파라미터는 [cubrid.conf 설정 파일과 기본 제공 파라미터](#)를 참고한다.

다음은 데이터 정의문 수행이 허용되지 않도록 **block\_ddl\_statement** 파라미터를 yes로 설정하는 예제이다.

```
csql> ;set block_ddl_statement=yes
=== Set Param Input ===
block_ddl_statement=yes
```

## cubrid.conf 설정 파일과 기본 제공 파라미터

CUBRID는 데이터베이스 서버, 브로커, CUBRID 매니저로 구성된다. 각 구성 요소를 수행하기 위한 설정 파일 이름은 다음과 같고, 모두 **\$CUBRID/conf** 디렉터리에 위치한다.

- 데이터베이스 서버 설정 파일 : **cubrid.conf**
- 브로커 설정 파일 : **cubrid\_broker.conf**
- CUBRID 매니저 서버 설정 파일 : **cm.conf**

**cubrid.conf** 파일은 CUBRID 데이터베이스 서버에 대한 시스템 파라미터를 지정하는 설정 파일로 데이터베이스 시스템의 전체적인 성능과 동작을 결정한다. **cubrid.conf** 파일은 시스템 설치에 필요한 몇 가지 중요한 파라미터가 기본으로 설정된 상태로 제공된다.

## 데이터베이스 서버 시스템 파라미터

다음은 **cubrid.conf** 설정 파일에 사용 가능한 데이터베이스 서버 시스템 파라미터이다. 아래 표에서 “적용 구분”의 “클라이언트 파라미터”는 브로커 응용서버(CAS), CSQL, **cubrid** 유틸리티에 적용되는 파라미터를 의미하며, “서버 파라미터”는 DB 서버(cub\_server 프로세스)에 적용되는 파라미터를 의미한다. 자세한 의미는 **데이터베이스 서버 설정이 미치는 범위**를 참조한다.

**SET SYSTEM PARAMETERS** 구문이나 CSQL 인터프리터의 세션 명령인 **;set**을 통해 DB 구동 중 동적으로 설정값 변경이 가능한 파라미터를 변경할 수 있다. 자세한 의미는 **질의 실행 계획 보기**를 참조한다. DB 사용자의 권한이 DBA인 경우 적용 구분에 상관없이 파라미터 값의 변경이 가능하며, DBA가 아닌 경우 “세션” 파라미터(아래 표에서 “세션” 항목의 값이 0인 파라미터)의 값만 변경이 가능하다.

아래 표에서 “적용 구분” 항목이 “서버”인 파라미터는 cub\_server 프로세스에 영향을 끼치며, “클라이언트”인 파라미터는 CAS, CSQL 또는 클라이언트/서버 모드(--CS-mode)로 실행하는 “cubrid” 유틸리티에 영향을 끼친다. “클라이언트/서버”인 파라미터는 cub\_server 프로세스와 CAS, CSQL, “cubrid” 유틸리티에 모두 영향을 끼친다.

아래 표에는 “동적 변경”과 “세션” 파라미터 여부가 표시되어 있다. “동적 변경”이 “가능”한 파라미터는 “적용 구분”과 “세션” 파라미터 여부에 따라 적용 범위가 다음과 같이 달라진다.

- “동적 변경”이 “가능”한 파라미터 중 “적용 구분”이 “서버”이면 변경된 파라미터 값이 DB 서버에 적용되어, 이후에 접속하는 응용 프로그램들은 변경된 값을 사용하며, DB를 재구동하기 전까지는 변경된 값을 유지한다.
- “동적 변경”이 “가능”한 파라미터 중 “적용 구분”이 “클라이언트”이면 “세션” 파라미터에 해당하며, DB 세션 당 변경된 값이 유지된다. 즉, 변경을 요청한 응용 프로그램에만 변경된 값이 적용된다. 예를 들어, **block\_ddl\_statement** 파라미터의 값이 **yes**로 변경되면 변경을 요청한 응용 프로그램에서만 DDL 문을 사용할 수 없게 된다.
- “동적 변경”이 “가능”한 파라미터 중 “적용 구분”이 “클라이언트/서버”이고
  - “세션” 파라미터에 해당하면 DB 세션 당 변경된 값이 유지된다. 즉, 변경을 요청한 응용 프로그램에만 변경된 값이 적용되며, 서버에는 영향을 미치지 않는다. 예를 들어,

**add\_column\_update\_hard\_default** 파라미터의 값이 **yes**로 변경되면, 변경을 요청한 응용 프로그램만이 NOT NULL 제약조건으로 새로 추가한 칼럼이 고정 기본값(hard default)을 갖게 한다.

- “세션” 파라미터에 해당하지 않으면 “클라이언트” 단과 “서버” 단의 값이 변경된다. 예를 들어, **error\_log\_level** 파라미터는 “서버” 단과 “클라이언트” 단에 각각 적용되는 파라미터로, 이 값이 “ERROR”에서 “WARNING”으로 변경되면 “서버”(cub\_server 프로세스)와 해당 변경을 요청한 “클라이언트”(CAS 또는 CSQL)에만 WARNING이 적용되고, 나머지 “클라이언트”에는 “ERROR”가 유지된다.

#### 참고

파라미터의 값을 영구히 변경하려면 cubrid.conf의 설정값을 변경한 후 DB 서버와 브로커 모두 재구동해야 한다.

| 용도 구분         | 파라미터 이름          | 적용 구분    | 세션 | 타입     | 기본값  | 동적 변경 |
|---------------|------------------|----------|----|--------|------|-------|
| 접속 관련<br>파라미터 | cubrid_port_id   | 클라이언트/서버 |    | int    | 1523 |       |
|               | check_peer_alive | 클라이언트/서버 | 0  | string | both | 가능    |
|               | db_hosts         | 클라이언트    | 0  | string | NULL | 가능    |
|               | max_clients      | 서버       |    | int    | 100  |       |
|               | tcp_keepalive    | 클라이언트/서버 |    | bool   | yes  |       |
|               | use_user_hosts   | 클라이언트/서버 |    | bool   | off  |       |
|               |                  | 서버       |    | byte   |      |       |

| 용도 구분          | 파라미터 이름                    | 적용 구분    | 세션 | 타입   | 기본값                      | 동적 변경   |
|----------------|----------------------------|----------|----|------|--------------------------|---------|
| 메모리 관련<br>파라미터 | data_buffer_size           |          |    |      | 32,768 *<br>db_page_size |         |
|                | index_scan_oid_buffer_size | 서버       |    | byte | 4 *<br>db_page_size      | DBA만 가능 |
|                | max_agg_hash_size          | 서버       |    | byte | 2,097,152(2M)            |         |
|                | max_hash_list_scan_size    | 클라이언트/서버 | O  | byte | 8,388,608(8M)            | 가능      |
|                | max_parallel_workers       | 서버       |    | int  | 100                      |         |
|                | parallelism                | 서버       |    | int  | 4                        |         |
|                | max_subquery_cache_size    | 서버       |    | byte | 2,097,152(2M)            | DBA만 가능 |
|                | memoize_memory_limit       | 클라이언트/서버 | O  | byte | 2,097,152(2M)            | 가능      |
|                | sort_buffer_size           | 서버       |    | byte | 128 *<br>db_page_size    | DBA만 가능 |
|                |                            | 서버       |    | int  | 4                        |         |

| 용도 구분          | 파라미터<br>이름                             | 적용 구분 | 세션 | 타입     | 기본값       | 동적 변경       |
|----------------|----------------------------------------|-------|----|--------|-----------|-------------|
|                | temp_file_<br>memory_<br>size_in_pages |       |    |        |           |             |
|                | thread_stacksize                       | 서버    |    | byte   | 1,048,576 |             |
| 디스크 관련<br>파라미터 | db_volume_size                         | 서버    |    | byte   | 512M      |             |
|                | dont_reuse_<br>heap_file               | 서버    |    | bool   | no        | DBA만 가<br>능 |
|                | log_volume_<br>size                    | 서버    |    | byte   | 512M      |             |
|                | temp_file_<br>max_size_in_<br>pages    | 서버    |    | int    | -1        |             |
|                | temp_volume_<br>path                   | 서버    |    | string | NULL      |             |
|                | unfill_factor                          | 서버    |    | float  | 0.1       | DBA만 가<br>능 |
|                | volume_ext<br>ension_<br>path          | 서버    |    | string | NULL      | DBA만 가<br>능 |



| 용도 구분          | 파라미터 이름                           | 적용 구분    | 세션 | 타입     | 기본값                               | 동적 변경   |
|----------------|-----------------------------------|----------|----|--------|-----------------------------------|---------|
| 오류 메시지 관련 파라미터 | double_write_buffer_size          | 서버       |    | byte   | 2,097,152(2M)                     | DBA만 가능 |
|                | data_file_os_advise               | 서버       |    | int    | 0                                 | DBA만 가능 |
|                | call_stack_dump_activation_list   | 클라이언트/서버 |    | string | DEFAULT                           | DBA만 가능 |
|                | call_stack_dump_deactivation_list | 클라이언트/서버 |    | string | NULL                              | DBA만 가능 |
|                | call_stack_dump_on_error          | 클라이언트/서버 |    | bool   | no                                | DBA만 가능 |
|                | error_log                         | 클라이언트/서버 |    | string | cub_client.err,<br>cub_server.err |         |
|                | error_log_level                   | 클라이언트/서버 |    | string | NOTIFICATION                      | DBA만 가능 |
|                | error_log_warning                 | 클라이언트/서버 |    | bool   | no                                | DBA만 가능 |

| 용도 구분          | 파라미터 이름                             | 적용 구분    | 세션 | 타입    | 기본값                        | 동적 변경   |
|----------------|-------------------------------------|----------|----|-------|----------------------------|---------|
|                | error_log_size                      | 클라이언트/서버 |    | int   | 512M                       | DBA만 가능 |
| 동시성/잠금<br>파라미터 | deadlock_detection_interval_in_secs | 서버       |    | float | 1.0                        | DBA만 가능 |
|                | isolation_level                     | 클라이언트    | 0  | int   | 4                          | 가능      |
|                | lock_escalation                     | 서버       |    | int   | 100,000                    | DBA만 가능 |
|                | lock_timeout                        | 클라이언트    | 0  | msec  | -1                         | 가능      |
|                | rollback_on_lock_escalation         | 서버       |    | bool  | no                         | DBA만 가능 |
| 로깅 관련<br>파라미터  | adaptive_flush_control              | 서버       |    | bool  | yes                        | DBA만 가능 |
|                | background_archiving                | 서버       |    | bool  | yes                        | DBA만 가능 |
|                | checkpoint_every_size               | 서버       |    | byte  | 100,000 *<br>log_page_size | DBA만 가능 |

| 용도 구분 | 파라미터 이름                             | 적용 구분 | 세션 | 타입   | 기본값                      | 동적 변경   |
|-------|-------------------------------------|-------|----|------|--------------------------|---------|
|       | checkpoint_interval                 | 서버    |    | msec | 6min                     | DBA만 가능 |
|       | checkpoint_sleep_msecs              | 서버    |    | msec | 1                        | DBA만 가능 |
|       | force_remove_log_archives           | 서버    |    | bool | yes                      | DBA만 가능 |
|       | log_buffer_size                     | 서버    |    | byte | 16k *<br>log_page_size   |         |
|       | log_max_archives                    | 서버    |    | int  | INT_MAX                  | DBA만 가능 |
|       | log_trace_flush_time                | 서버    |    | msec | 0                        | DBA만 가능 |
|       | max_flush_size_per_second           | 서버    |    | byte | 10,000 *<br>db_page_size | DBA만 가능 |
|       | remove_log_archive_interval_in_secs | 서버    |    | sec  | 0                        | DBA만 가능 |

| 용도 구분           | 파라미터 이름                        | 적용 구분    | 세션 | 타입   | 기본값                | 동적 변경   |
|-----------------|--------------------------------|----------|----|------|--------------------|---------|
| 트랜잭션 처리 관련 파라미터 | sync_on_flush_size             | 서버       |    | byte | 200 * db_page_size | DBA만 가능 |
|                 | ddl_audit_log                  | 클라이언트    |    | bool | no                 |         |
|                 | ddl_audit_log_size             | 클라이언트    |    | byte | 10M                |         |
|                 | async_commit                   | 서버       |    | bool | no                 | DBA만 가능 |
|                 | group_commit_interval_in_msecs | 서버       |    | msec | 0                  | DBA만 가능 |
| 구문/타입 관련 파라미터   | dblink_auto_commit             | 서버       |    | bool | yes                |         |
|                 | add_column_update_hard_default | 클라이언트/서버 | 0  | bool | no                 | 가능      |
|                 | alter_table_change_type_strict | 클라이언트/서버 | 0  | bool | yes                | 가능      |
|                 | allow_truncated_string         | 클라이언트/서버 | 0  | bool | no                 | 가능      |

| 용도 구분 | 파라미터 이름                       | 적용 구분    | 세션 | 타입     | 기본값   | 동적 변경   |
|-------|-------------------------------|----------|----|--------|-------|---------|
|       | ansi_quotes                   | 클라이언트    |    | bool   | yes   |         |
|       | block_ddl_statement           | 클라이언트    | 0  | bool   | no    | 가능      |
|       | block_nowhere_statement       | 클라이언트    | 0  | bool   | no    | 가능      |
|       | compat_numeric_division_scale | 클라이언트/서버 | 0  | bool   | no    | 가능      |
|       | create_table_reuseoid         | 클라이언트    | 0  | bool   | yes   | 가능      |
|       | cte_max_recursions            | 클라이언트/서버 | 0  | int    | 2000  | 가능      |
|       | default_week_format           | 클라이언트/서버 | 0  | int    | 0     | 가능      |
|       | group_concat_max_len          | 서버       |    | byte   | 1,024 | DBA만 가능 |
|       | intl_check_input_string       | 클라이언트    |    | bool   | no    |         |
|       | intl_collation                | 클라이언트    | 0  | string |       | 가능      |

| 용도 구분 | 파라미터 이름                       | 적용 구분    | 세션 | 타입     | 기본값    | 동적 변경   |
|-------|-------------------------------|----------|----|--------|--------|---------|
|       | intl_date_lang                | 클라이언트    | 0  | string |        | 가능      |
|       | intl_number_lang              | 클라이언트    | 0  | string |        | 가능      |
|       | json_max_array_idx            | 서버       |    | int    | 65,536 | DBA만 가능 |
|       | max_query_per_tran            | 서버       |    | int    | 100    |         |
|       | no_backslash_escapes          | 클라이언트/서버 | 0  | bool   | yes    | 가능      |
|       | only_full_group_by            | 클라이언트    | 0  | bool   | no     | 가능      |
|       | oracle_compat_number_behavior | 클라이언트/서버 |    | bool   | no     |         |
|       | oracle_style_empty_string     | 클라이언트/서버 |    | bool   | no     |         |
|       | pipes_as_concat               | 클라이언트    |    | bool   | yes    |         |
|       | plus_as_concat                | 클라이언트    |    | bool   | yes    |         |

| 용도 구분          | 파라미터 이름                              | 적용 구분    | 세션 | 타입   | 기본값       | 동적 변경 |
|----------------|--------------------------------------|----------|----|------|-----------|-------|
|                | require_like_escape_character        | 클라이언트    |    | bool | no        |       |
|                | return_null_on_function_errors       | 클라이언트/서버 | O  | bool | no        | 가능    |
|                | stored_procedure_return_numeric_size | 클라이언트/서버 |    | list | 38,15     |       |
|                | string_max_size_bytes                | 클라이언트/서버 | O  | byte | 1,048,576 | 가능    |
|                | unicode_input_normalization          | 클라이언트    |    | bool | no        |       |
|                | unicode_output_normalization         | 클라이언트    |    | bool | no        |       |
|                | update_use_attribute_references      | 클라이언트    | O  | bool | no        | 가능    |
| 스레드 관련<br>파라미터 | thread_connection_pooling            | 서버       |    | bool | yes       |       |
|                |                                      | 서버       |    | int  | 300       |       |

| 용도 구분            | 파라미터 이름                           | 적용 구분    | 세션 | 타입     | 기본값                | 동적 변경   |
|------------------|-----------------------------------|----------|----|--------|--------------------|---------|
|                  | thread_connection_timeout_seconds |          |    |        |                    |         |
|                  | thread_worker_pooling             | 서버       |    | bool   | yes                |         |
|                  | thread_core_count                 | 서버       |    | int    | 시스템 코어 수           |         |
|                  | thread_worker_timeout_seconds     | 서버       |    | int    | 300                |         |
|                  | loaddb_worker_count               | 서버       |    | int    | 8                  | DBA만 가능 |
| 타임존 파라미터         | server_timezone                   | 클라이언트/서버 |    | string | OS의 타임존            |         |
|                  | timezone                          | 클라이언트/서버 | 0  | string | server_timezone의 값 | 가능      |
|                  | tz_leap_second_support            | 클라이언트/서버 |    | bool   | no                 |         |
| 질의 계획 캐시 관련 파라미터 | max_plan_cache_entries            | 클라이언트/서버 |    | int    | 1,000              |         |



| 용도 구분            | 파라미터 이름                              | 적용 구분    | 세션 | 타입   | 기본값   | 동적 변경 |
|------------------|--------------------------------------|----------|----|------|-------|-------|
| 질의 캐쉬<br>관련 파라미터 | max_plan_cache_clones                | 서버       |    | int  | 1,000 |       |
|                  | xasl_cache_time_threshold_in_minutes | 클라이언트/서버 |    | int  | 360   |       |
|                  | max_filter_pred_cache_entries        | 클라이언트/서버 |    | int  | 1,000 |       |
|                  | max_query_cache_entries              | 서버       |    | int  | 0     |       |
| 유틸리티 관련<br>파라미터  | query_cache_size_in_pages            | 서버       |    | int  | 0     |       |
|                  | backup_volume_max_size_bytes         | 서버       |    | byte | 0     |       |
|                  | communication_histogram              | 클라이언트    | 0  | bool | no    | 가능    |
|                  | compactdb_page_reclaim_only          | 서버       |    | int  | 0     |       |

| 용도 구분      | 파라미터 이름                   | 적용 구분    | 세션 | 타입     | 기본값  | 동적 변경   |
|------------|---------------------------|----------|----|--------|------|---------|
|            | csql_history_num          | 클라이언트    | 0  | int    | 50   | 가능      |
| HA 관련 파라미터 | ha_mode                   | 클라이언트/서버 |    | string | off  |         |
| 기타 파라미터    | access_ip_control         | 서버       |    | bool   | no   | DBA만 가능 |
|            | access_ip_control_file    | 서버       |    | string |      | DBA만 가능 |
|            | agg_hash_respect_order    | 클라이언트    |    | bool   | yes  | DBA만 가능 |
|            | auto_restart_server       | 서버       |    | bool   | yes  |         |
|            | enable_string_compression | 클라이언트/서버 |    | bool   | yes  |         |
|            | index_scan_in_oid_order   | 클라이언트    | 0  | bool   | no   | 가능      |
|            | index_unfill_factor       | 서버       |    | float  | 0.05 | DBA만 가능 |
|            | stored_procedure          | 서버       |    | bool   | yes  |         |

| 용도 구분 | 파라미터 이름                        | 적용 구분 | 세션 | 타입     | 기본값    | 동적 변경   |
|-------|--------------------------------|-------|----|--------|--------|---------|
|       | stored_procedure_uds           | 서버    |    | bool   | yes    |         |
|       | stored_procedure_port          | 서버    |    | int    | 0      |         |
|       | stored_procedure_vm_options    | 서버    |    | string |        |         |
|       | multi_range_optimization_limit | 클라이언트 | 0  | int    | 100    | 가능      |
|       | optimizer_enable_merge_join    | 클라이언트 |    | bool   | no     | 가능      |
|       | use_stat_estimation            | 서버    |    | bool   | no     | DBA만 가능 |
|       | server                         | 서버    |    | string |        |         |
|       | service                        | 서버    |    | string |        |         |
|       | session_state_timeout          | 서버    |    | sec    | 21,600 |         |
|       | sort_limit_max_count           | 클라이언트 | 0  | int    | 1,000  | 가능      |

| 용도 구분 | 파라미터<br>이름                          | 적용 구분    | 세션 | 타입     | 기본값                     | 동적 변경   |
|-------|-------------------------------------|----------|----|--------|-------------------------|---------|
|       | sql_trace_slow                      | 서버       |    | msec   | -1                      | DBA만 가능 |
|       | sql_trace_execution_plan            | 서버       |    | bool   | no                      | DBA만 가능 |
|       | use_orderby_sort_limit              | 서버       |    | bool   | yes                     | DBA만 가능 |
|       | vacuum_pre_fetch_log_mode           | 서버       |    | int    | 1                       |         |
|       | vacuum_pre_fetch_log_buffer_size    | 서버       |    | int    | 3200 *<br>log_page_size |         |
|       | data_buffer_neighbor_flush_pages    | 서버       |    | int    | 8                       | DBA만 가능 |
|       | data_buffer_neighbor_flush_nondirty | 서버       |    | bool   | no                      | DBA만 가능 |
|       | tde_keys_file_path                  | 클라이언트/서버 |    | string | NULL                    |         |
|       | tde_default_algorithm               | 클라이언트/서버 |    | string | AES                     |         |

| 용도 구분 | 파라미터<br>이름                         | 적용 구분    | 세션 | 타입     | 기본값   | 동적 변경 |
|-------|------------------------------------|----------|----|--------|-------|-------|
|       | recovery_progress_logging_interval | 서버       |    | int    | 0     |       |
|       | supplemental_log                   | 클라이언트/서버 |    | int    | 0     |       |
|       | regexp_engine                      | 클라이언트/서버 | 0  | string | re2   | 가능    |
|       | vacuum_overflow_check_threshold    | 서버       |    | int    | 1000  |       |
|       | vacuum_overflow_check_duration     | 서버       |    | int    | 45000 |       |
|       | deduplicate_key_level              | 클라이언트/서버 | 0  | int    | -1    | 가능    |
|       | print_index_detail                 | 클라이언트/서버 | 0  | bool   | no    | 가능    |
|       | enable_memory_monitoring           | 클라이언트/서버 |    | bool   | no    |       |

| 용도 구분 | 파라미터 이름           | 적용 구분 | 세션 | 타입  | 기본값 | 동적 변경 |
|-------|-------------------|-------|----|-----|-----|-------|
|       | flashback_timeout | 클라이언트 |    | int | 300 |       |

- **log\_page\_size**: 데이터베이스 생성 시 **--log-page-size** 옵션으로 지정한 로그 볼륨 페이지 크기. 기본값: 16KB. 관련 파라미터의 설정 값은 페이지 단위로 버림된다. 예를 들어 checkpoint\_every\_size 의 값은 16KB로 나누어 소수점 이하를 버림한 값에 16KB를 곱한 값이 된다.
- **db\_page\_size**: 데이터베이스 생성 시 **--db-page-size** 옵션으로 지정한 DB 볼륨 페이지 크기. 기본값: 16KB. 관련 파라미터의 설정 값은 페이지 단위로 버림된다. 예를 들어 data\_buffer\_size 의 값은 16KB로 나누어 소수점 이하를 버림한 값에 16KB를 곱한 값이 된다.

## 파라미터의 섹션별 분류

**cubrid.conf**에 지정된 파라미터는 다음과 같이 네 가지 섹션으로 제공된다.

- CUBRID 서비스를 시작할 때 사용 : [service] 섹션
- 전체 데이터베이스에 공통으로 적용 : [common] 섹션
- 각 데이터베이스에 개별적으로 적용 : [@<database>] 섹션
- cubrid 유틸리티가 독립 모드(stand-alone, --SA-mode)로 구동할 때만 사용 : [standalone] 섹션

여기서 <database>는 파라미터를 개별적으로 적용할 데이터베이스 이름이며, [common]에 설정된 파라미터가 [@<database>]에 설정된 파라미터와 동일한 경우 [@<database>]에 설정된 파라미터가 최종 적용된다.

```
.....
[common]
.....
sort_buffer_size=2M
.....
[standalone]

sort_buffer_size=256M
.....
```

[standalone] 섹션에 정의된 설정은 "cubrid"로 시작하는 cubrid 유틸리티들이 독립 모드로 구동할 때만 사용된다. 예를 들어, 위와 같이 설정한 상태에서 --CS-mode(기본값)으로 DB를 구동(cubrid database start db\_name)하면 "sort\_buffer\_size=2M"가 적용된다. 하지만 DB를 정지하고 "cubrid loadddb --SA-mode"를 실행할 때는 "sort\_buffer\_size=256M"가 적용된다. "cubrid loadddb --SA-

mode”를 실행할 때 인덱스 생성 과정에서 정렬 버퍼(sort buffer)를 더 많이 사용할 것으로 예상되므로 이를 늘려주는 것이 “loaddb” 수행 성능에 도움이 된다.

## 기본 제공 파라미터

CUBRID 설치 시 생성되는 기본 데이터베이스 환경 설정 파일( **cubrid.conf**)에는 데이터베이스 서버 파라미터 중 반드시 변경해야 할 일부 파라미터가 기본적으로 포함된다. 기본으로 포함되지 않는 파라미터의 설정값을 변경하기 원할 경우 직접 추가/편집해서 사용하면 된다.

다음은 **cubrid.conf** 파일 내용이다.

```
Copyright (C) 2008 Search Solution Corporation. All rights
reserved by Search Solution.
#
Id
#
cubrid.conf#

For complete information on parameters, see the CUBRID
Database Administration Guide chapter on System Parameters

Service section - a section for 'cubrid service' command
[service]

The list of processes to be started automatically by 'cubrid
service start' command
Any combinations are available with server, broker and manager.
service=server,broker,manager

The list of database servers in all by 'cubrid service start'
command.
This property is effective only when the above 'service' property
contains 'server' keyword.
#server=demodb,testdb

Common section - properties for all databases
This section will be applied before other database specific
sections.
[common]

Read the manual for detailed description of system parameters
Manual > System Configuration > Database Server Configuration >
Default Parameters

Size of data buffer are using K, M, G, T unit
data_buffer_size=512M

Size of log buffer are using K, M, G, T unit
log_buffer_size=256M
```

```
Size of sort buffer are using K, M, G, T unit
The sort buffer should be allocated per thread.
So, the max size of the sort buffer is sort_buffer_size *
max_clients.
sort_buffer_size=2M

The maximum number of concurrent client connections the server
will accept.
This value also means the total # of concurrent transactions.
max_clients=100

TCP port id for the CUBRID programs (used by all clients).
cubrid_port_id=1523
```

testdb 만 data\_buffer\_size를 128M로, max\_clients를 10으로 설정하고 싶은 경우 다음과 같이 설정한다.

```
[service]

service=server,broker,manager

[common]

data_buffer_size=512M
log_buffer_size=256M
sort_buffer_size=2M
max_clients=100

TCP port id for the CUBRID programs (used by all clients).
cubrid_port_id=1523

[@testdb]
data_buffer_size=128M
max_clients=10
```

### 접속 관련 파라미터

다음은 데이터베이스 서버와 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름        | 타입  | 기본값  | 최소값 | 최대값 |
|----------------|-----|------|-----|-----|
| cubrid_port_id | int | 1523 | 1   |     |



| 파라미터 이름          | 타입     | 기본값  | 최소값 | 최대값  |
|------------------|--------|------|-----|------|
| check_peer_alive | string | both |     |      |
| db_hosts         | string | NULL |     |      |
| max_clients      | int    | 100  | 10  | 4000 |
| tcp_keepalive    | bool   | yes  |     |      |
| use_user_hosts   | bool   | off  |     |      |

### cubrid\_port\_id

**cubrid\_port\_id**는 마스터 프로세스가 사용하는 포트를 설정하기 위한 파라미터로 기본값은 **1523**이다. CUBRID를 설치한 서버에서 이미 1523 포트를 사용하고 있거나, 방화벽에 의해 1523 포트가 차단된 경우에는 마스터 프로세스가 정상적으로 구동할 수 없으므로, 마스터 서버와 연결할 수 없다는 에러 메시지가 나타날 수 있다. 이와 같이 포트 충돌이 발생하는 경우, 관리자는 서버 환경을 고려하여 **cubrid\_port\_id**의 설정값을 변경해야 한다.

### check\_peer\_alive

**check\_peer\_alive**는 클라이언트 프로세스와 서버 프로세스가 정상 동작하는지 각각 확인하는 과정의 수행 여부를 결정하는 파라미터이다. 기본값은 **both**이다.

서버 프로세스와 접속하는 클라이언트 프로세스에는 브로커 응용 서버(cub\_cas) 프로세스, 복제 로그 반영 프로세스(copylogdb), 복제 로그 복사 프로세스(applylogdb), CSQL 인터프리터(csql) 등이 있다. 서버 프로세스와 클라이언트 프로세스는 접속이 이루어진 후 네트워크를 통해 데이터를 기다리는 중 오랫동안(예: 5초 이상) 응답을 받지 못하면 설정에 따라 상대방이 정상 동작하는지 확인하는 과정을 거친다. 서로 확인하는 과정에서 정상 동작하지 않는다고 판단되면 연결된 접속을 강제 종료한다.

값의 종류 및 동작 방식은 다음과 같다.

- **both**: 서버 프로세스는 클라이언트 프로세스의 ECHO(7) 포트에 주기적으로 접속하여 클라이언트 프로세스가 정상 동작하는지 확인하고, 클라이언트 프로세스는 서버 프로세스의 ECHO(7) 포트에 주기적으로 접속하여 서버 프로세스가 정상 동작하는지 확인한다(기본값).
- **server\_only**: 서버 프로세스만 클라이언트 프로세스가 정상 동작하는지 확인한다.

- **client\_only**: 클라이언트 프로세스만 서버 프로세스가 정상 동작하는지 확인한다.
- **none**: 클라이언트 프로세스와 서버 프로세스 둘 다 상대방이 정상 동작하는지 확인하지 않는다.

특히, ECHO(7) 포트가 방화벽(firewall) 설정으로 막혀있으면 서버 프로세스 또는 클라이언트 프로세스가 각각 서로의 상태를 확인할 때 상대방 프로세스가 종료된 것으로 오인할 수 있으므로, none으로 설정하여 이 문제를 회피해야 한다.

## db\_hosts

**db\_hosts**는 클라이언트에서 연결할 수 있는 데이터베이스 서버 호스트의 목록 및 연결 순서를 지정하기 위한 파라미터이다. 서버 호스트 목록은 한 개 이상의 서버 호스트 이름을 나열하며, 각 호스트는 이름 사이에 공백 또는 콜론(:) 기호를 사용하여 구분한다. 이 때, 중복되거나 존재하지 않는 호스트 이름은 무시된다.

다음은 **db\_hosts** 파라미터의 설정값을 보여주는 예제로 **host1, host2, host3**의 순서대로 연결이 시도된다.

```
db_hosts="hosts1:hosts2:hosts3"
```

한편, 클라이언트는 서버 연결을 위하여 데이터베이스 위치 정보 파일(**databases.txt**)을 참조하여 지정된 서버 호스트에 1차적으로 연결을 시도한다. 연결이 실패하면 데이터베이스 설정 파일(**cubrid.conf**)의 **db\_hosts** 파라미터의 설정값을 참조하여 2차적으로 지정된 서버 호스트 중 첫 번째 서버 호스트에 연결을 시도한다.

## max\_clients

**max\_clients**는 데이터베이스 서버에 동시 연결을 허용하는 클라이언트(일반적으로 브로커 응용 서버(CAS))의 최대 개수를 지정하기 위한 파라미터이다. 즉, **max\_clients** 파라미터는 데이터베이스 서버 프로세스 하나 당 동시에 접속할 수 있는 클라이언트의 최대 개수를 의미한다. 이 파라미터의 기본값은 **100**이다.

CUBRID 환경에서 동시 사용자 수를 증가시키기 위해서는 질의 성능을 고려하여 **max\_clients** 파라미터(**cubrid.conf**) 및 **MAX\_NUM\_APPL\_SERVER** 파라미터(**cubrid\_broker.conf**)를 적절한 값으로 설정해야 한다. 즉, **max\_clients** 파라미터를 통해 데이터베이스 서버가 허용하는 동시 접속 개수를 설정하고, **MAX\_NUM\_APPL\_SERVER** 파라미터를 통해 해당 브로커가 허용하는 동시 접속 개수를 설정한다.

예를 들어, **cubrid\_broker.conf** 파일에서 [%query\_editor]의 **MAX\_NUM\_APPL\_SERVER** 값이 50이고 [%BROKER1]의 **MAX\_NUM\_APPL\_SERVER** 값이 50인 브로커 노드 2개가 하

나의 데이터베이스 서버에 접속하는 경우, 데이터베이스 서버가 허용하는 동시 접속 개수인 **max\_clients**의 값은 다음과 같이 설정할 수 있다.

- (각 브로커 노드 당 최대 50개) \* (브로커 노드 2개) + (CSQL 인터프리터의 데이터베이스 서버 접속, HA 로그 복사 프로세스와 같은 CUBRID 내부 프로세스의 데이터베이스 서버 접속 등에 대한 여유분 10개) = 110

특히, HA 환경에서는 failover 등으로 인해 여러 브로커 노드 접속이 하나의 데이터베이스 서버에 집중될 수 있으므로, 같은 데이터베이스에 접속하는 모든 브로커 노드의 **MAX\_NUM\_APPL\_SERVER** 값을 합한 값 보다 크게 설정해야 한다.

클라이언트의 데이터베이스 접속 여부에 관계 없이 **max\_clients**의 개수를 크게 설정할 수록 메모리 사용량이 증가하므로 주의한다.

#### 참고

Linux 시스템에서 max\_clients 파라미터는 “ulimit -n” 명령과 관련이 있는데, “ulimit -n” 명령은 프로세스 하나가 사용할 수 있는 file descriptor의 최대 개수를 지정한다. file descriptor는 파일 뿐 아니라 네트워크 소켓도 포함하므로, “ulimit -n”의 개수는 max\_clients의 개수보다 크게 설정해야 한다.

## tcp\_keepalive

**tcp\_keepalive**는 TCP 네트워크 프로토콜에 SO\_KEEPAIVE 옵션을 적용할지 여부를 지정하는 파라미터로, 기본값은 **yes**이다. 이 값이 **\*\*no**이면 마스터 노드와 슬레이브 노드 간 방화벽이 설정되어 있는 환경에서 장시간 동안 트랜잭션 로그가 복사되지 않을 때 DB 서버 쪽 연결이 종료되는 현상이 발생할 수 있다.

## use\_user\_hosts

**use\_user\_hosts** 시스템 파라미터는 CUBRID 서비스에서 필요한 **호스트명**, **IP 주소** 검색 기능을 아래 서비스 중에서 선택하기 위해 사용한다. 기본값은 **OFF** 이다.

- 시스템 에서 제공하는 호스트, IP 주소 검색 라이브러리 (Linux의 경우 **glibc**)
- **CUBRID** 에서 제공하는 **호스트/IP 주소 검색 라이브러리**

use\_user\_hosts=off (기본값)

- 시스템 라이브러리를 사용하여 IP 주소, 호스트명을 검색한다.
- 일반적으로 /etc/hosts 파일 검색, DNS Query 등이 사용된다.

use\_user\_hosts=on

- CUBRID 호스트 검색 라이브러리를 이용하여 IP 주소/호스트명을 검색한다.

- `$CUBRID/conf/cubrid_hosts.conf` 파일을 이용하여, 호스트명과 IP 주소를 검색한다
- 시스템의 `/etc/hosts`, `/etc/nsswitch.conf` 파일에 대한 읽기 권한 등과 관계 없이 동작한다.

### ⚠ 경고

`use_user_hosts` 파라미터는 서비스 운영 중 변경되면 정상적으로 서비스가 종료되지 않을 수 있으므로, 반드시 CUBRID 서비스가 종료된 상태에서 변경해야 한다.

- `$CUBRID/conf/cubrid_hosts.conf` 파일의 형식은 `/etc/hosts` 와 동일하나 다음과 같은 몇 가지 제약이 있다.

- **IPv4** 주소 만을 허용한다 (**IPv6** 는 허용하지 않는다).
- **alias** 는 허용하지 않는다.

```
172.31.0.1 host1 alias1 alias2
```

- 같은 ip 주소가 여러 개 있는 경우 맨 위 줄에 있는 {ip, hostname} 정보만 사용되며, 나머지는 무시된다.

```
172.31.0.1 host1
172.31.0.1 alias1
172.31.0.1 alias2
```

- 두 개 이상의 IP에 동일한 호스트명을 사용할 경우 맨 위 줄에 있는 IP만 적용된다.

```
172.31.0.1 host1
178.31.0.2 host1
```

- 다음은 `$CUBRID/conf/cubrid_hosts.conf`의 예시이다.

```
#
hosts file for CUBRID user specific host service
#
127.0.0.1 localhost
172.31.0.1 node1
192.168.0.31 node4.kr # Seoul
192.168.2.31 node5.gov.or.kr # Daejeon
```

**⚠ 경고**

\$CUBRID/conf/cubrid\_hosts.conf는 반드시 모든 CUBRID 프로세스를 종료시킨 후 변경해야 하며, **재 구동 후 변경된 사항이 적용된다.** 또한, **localhost** 와 **'hostname'** (Linux 명령어 중 hostname에 의해 출력되는 호스트명)을 반드시 cubrid\_hosts.conf에 포함해야 한다.

**⚠ 경고**

\$CUBRID/conf/cubrid\_hosts.conf 파일에서 호스트 이름은 다음과 같은 형식을 따른다 (Linux 호스트명 작성 규칙).

- 호스트 이름으로는 오직 영문자, 숫자(0 ~ 9), 하이픈('-'), 점('.') 문자만 사용할 수 있다.
- 호스트 이름의 첫 문자는 영문자이어야 한다.
- 호스트 이름의 마지막 문자는 영문자와 숫자이어야 한다.
- FQDN (Fully Qualified Domain Name) 형태의 호스트명이 사용 가능하다 (예: www.cubrid.com).

다음과 같은 호스트 이름은 허용한다.

```
cubrid-dev1
CUB2.dev
```

**메모리 관련 파라미터**

다음은 데이터베이스 서버 또는 클라이언트에서 사용하는 메모리와 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름          | 타입   | 기본값                    | 최소값                       | 최대값                                              |
|------------------|------|------------------------|---------------------------|--------------------------------------------------|
| data_buffer_size | byte | 32,768<br>db_page_size | * 1,024<br>* db_page_size | 2G(32비트),<br>INT_MAX *<br>db_page_size<br>(64비트) |

| 파라미터 이름                            | 타입   | 기본값                   | 최소값                    | 최대값                                              |
|------------------------------------|------|-----------------------|------------------------|--------------------------------------------------|
| index_scan_oid_<br>buffer_size     | byte | 4 *<br>db_page_size   | 0.05 *<br>db_page_size | 16 *<br>db_page_size                             |
| max_agg_hash_si<br>ze              | byte | 2,097,152(2M)         | 32,768(32K)            | 134,217,728(128M<br>B)                           |
| max_hash_list_sc<br>an_size        | byte | 8,388,608(8M)         | 0                      | 128MB                                            |
| max_parallel_wor<br>kers           | int  | 100                   | 0                      | 1000                                             |
| parallelism                        | int  | 4                     | 0                      | MIN(32, 시스템<br>코어 수)                             |
| max_subquery_c<br>ache_size        | byte | 2,097,152(2M)         | 0                      | 16,777,216(16M)                                  |
| memoize_memor<br>y_limit           | byte | 2,097,152(2M)         | 0                      | 없음                                               |
| sort_buffer_size                   | byte | 128 *<br>db_page_size | 1 *<br>db_page_size    | 2G(32비트),<br>INT_MAX *<br>db_page_size<br>(64비트) |
| temp_file_memor<br>y_size_in_pages | int  | 4                     | 0                      | 20                                               |
| thread_stacksize                   | byte | 1,048,576             | 65,536                 |                                                  |

**data\_buffer\_size**

**data\_buffer\_size**는 데이터베이스 서버가 메모리 내에 캐시하는 데이터 버퍼의 크기를 설정하기 위한 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $32,768 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **512M**) 이고, 최소값은  $1,024 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **16M**)이다. CUBRID 64비트 버전에서는 최대값이  $INT\_MAX * db\_page\_size$ 이다. CUBRID 32비트 버전에서는 최대값이 **2G**임에 주의한다.

**data\_buffer\_size** 파라미터의 값이 클수록 버퍼에 캐시되는 데이터 페이지가 많아지므로 디스크 I/O 비용을 줄일 수 있다는 장점이 있다. 반면, 이 파라미터의 값을 너무 크게 설정하면 과도하게 시스템 메모리가 점유되므로 운영체제에 의해 버퍼 풀이 스와핑(swapping)되는 현상이 발생할 수 있다. **data\_buffer\_size** 파라미터는 필요한 메모리 크기가 시스템 메모리의 2/3 이내가 되도록 설정할 것을 권장한다.

· 필요한 메모리 크기 = 데이터 버퍼 크기(**data\_buffer\_size**)

### index\_scan\_oid\_buffer\_size

**index\_scan\_oid\_buffer\_size**는 인덱스 스캔을 수행할 때 OID 리스트의 임시 저장을 위한 버퍼의 크기를 설정하기 위한 파라미터이다. K 단위를 설정할 수 있으며, KB(kilobytes)를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $4 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **64K**)이다. 최소값은  $0.05 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 약 **1K**)이고, 최대값은  $16 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **256K**)이다.

**index\_scan\_oid\_buffer\_size** 파라미터 값과 데이터베이스 생성 시 설정한 단위 페이지의 크기에 비례하여 OID 버퍼의 크기가 결정되고, 이러한 OID버퍼의 크기가 클수록 인덱스 스캔 비용이 증가하는 경향을 보인다. 이를 고려하여 **index\_scan\_oid\_buffer\_size** 파라미터 값을 조정할 수 있다.

### max\_agg\_hash\_size

**max\_agg\_hash\_size**는 집계를 포함하는 질의에서 튜플 그룹을 해싱하기 위해 할당된 트랜잭션 당 최대 메모리 크기를 설정하는 파라미터이다. 기본값은 **2,097,152(2M)**, 최소값은 32,768(32K), 그리고 최대값은 134,217,728(128MB)이다.

**NO\_HASH\_AGGREGATE** 힌트가 명시되면, 집계 작업 시 해싱 방식이 사용되지 않을 것이다. **agg\_hash\_respect\_order**를 참고한다.

### max\_hash\_list\_scan\_size

**max\_hash\_list\_scan\_size**는 부질의를 포함하는 질의에서 해시 테이블을 빌드하기 위해 할당된 트랜잭션 당 최대 메모리 크기를 설정하는 파라미터이다. 기본값은 **8,388,608(8M)**, 최소값은 0, 그리고 최대값은 134,217,728(128MB)이다.

**max\_hash\_list\_scan\_size**이 0으로 설정되거나, **NO\_HASH\_LIST\_SCAN** 힌트가 명시되면, 조희 작업 시 해싱 방식이 사용되지 않을 것이다.

## max\_parallel\_workers

**max\_parallel\_workers**는 서버 전체에서 동시에 실행할 수 있는 병렬 질의 워커(parallel query worker) 스레드의 최대 개수를 설정하는 파라미터이다. 기본값은 **100**이며, 최소값은 **0**, 최대값은 **1000**이다.

이 파라미터가 **0**으로 설정되면 병렬 질의 기능이 비활성화된다. **2 이상**으로 설정하면 병렬 스캔(Parallel Scan; 힙/리스트/인덱스), 병렬 부질의 실행(Parallel Subquery Execution), 병렬 해시 조인(Parallel Hash Join), 병렬 정렬(Parallel Sort) 등 다양한 병렬 처리 기능을 사용할 수 있다.

서버는 **전역 워커 풀(worker pool)**을 통해 병렬 질의 실행 작업을 관리한다. 다수의 세션이 동시에 병렬 질의를 요청하거나 단일 세션 내에서 복합적인 병렬 연산이 수행되더라도 서버 전체에서 활성화된 병렬 워커의 총합은 **max\_parallel\_workers** 값을 초과할 수 없다.

각 서버 스레드는 질의 실행 전에 **병렬 질의 워커 매니저(parallel query worker manager)**를 통해 필요한 워커 수를 미리 예약(reserve)하며, 작업 완료되면 즉시 해제(release)한다. 가용 워커가 부족하여 요청한 수를 모두 예약하지 못하면 실제 예약된 워커 수만큼만 병렬로 실행되며, 예약된 워커가 하나도 없으면 해당 질의는 일반적인 단일 스레드 실행 방식으로 수행된다.

이 파라미터는 `cubrid.conf`에 설정한 후 서버를 재시작해야 적용되며, **SET SYSTEM PARAMETERS** 문으로는 변경할 수 없다.

병렬 질의 실행에 대한 자세한 내용은 [병렬 실행](#)를 참고한다.

다음은 최대 200개의 병렬 워커를 허용하는 예제이다.

```
max_parallel_workers=200
```

### 참고

병렬 질의는 충분한 CPU 코어와 메모리가 확보된 환경에서 효과적이다. **max\_parallel\_workers** 값이 실제 물리적 코어 수에 비해 지나치게 크면 시스템 리소스 경합으로 인해 오히려 성능이 저하될 수 있다.

### 참고



병렬 질의의 각 워커 스레드는 독립적인 자원(메모리 등)을 사용하므로, 사용하는 병렬 워커 수가 늘어날수록 서버의 전체 메모리 사용량이 증가한다.

#### 참고

모든 쿼리를 병렬로 실행할 경우 서버 자원의 과도한 점유로 인해 시스템 전체의 성능 저하가 발생할 수 있습니다. 이를 방지하기 위해 옵티마이저는 병렬 처리 효율이 높은 쿼리를 선별 허용하는 **처리량 규칙**을 적용한다.

## parallelism

**parallelism**은 단일 병렬 연산에 할당할 수 있는 **병렬 처리 수준(degree of parallelism)**의 상한값을 설정하는 파라미터이다. 기본값은 **4**이며, 최소값은 **0**, 최대값은 **MIN(32, 시스템 CPU 코어 수)**이다.

이 파라미터가 **0**이나 **1**로 설정하면 **PARALLEL** 힌트가 없는 모든 질의는 단일 스레드로 실행된다. **2 이상**으로 설정하면 옵티마이저가 판단하기에 병렬 처리가 효율적인 연산(scan, hash join, sort, subquery 등)에 대해 자동으로 병렬 실행을 적용한다.

실제 쿼리 실행시 적용되는 병렬 처리 수준은 다음과 같은 규칙을 따른다.

**PARALLEL(N)** 힌트가 사용되면 해당 파라미터 값을 무시하고 N 만큼의 워커를 할당 시도한다. 단 최대 허용값(32 또는 시스템 코어 수 중 작은 값)은 초과할 수 없다.

힌트가 없는 경우, 개별 병렬 연산에 할당되는 워커의 수는 이 파라미터 값을 초과할 수 없다. 하지만 하나의 전체 질의가 여러 개의 병렬 연산을 동시에 수행하는 구조인 경우에는 해당 질의에 사용되는 총 병렬 워커수는 이 파라미터 값을 초과할 수 있다. 단, 서버 전체에서 수행되는 병렬 워커 수는 **max\_parallel\_workers** 값을 초과할 수는 없다.

**PARALLEL ( degree )** 힌트를 사용하여 질의별로 병렬 처리 수준을 재정의할 수 있다. 자세한 내용은 **병렬 실행**를 참고한다.

이 파라미터는 **cubrid.conf**에 설정한 후 서버를 재시작해야 적용되며, **SET SYSTEM PARAMETERS** 문으로는 변경할 수 없다. 서버 기동 시에는 다음 순서로 값이 조정된다.

- 시스템의 CPU 코어 수보다 크면 코어 수로 낮춘다.
- 그 결과가 **max\_parallel\_workers** 값보다 크면 그 값으로 다시 낮춘다.

다음은 기본 병렬 처리 수준을 8로 설정하는 예제이다.

```
parallelism=8
```

**참고**

**parallelism** 파라미터는 개별 병렬 연산의 상한값을 설정하지만, 하나의 전체 질의의 실제 병렬 실행 여부와 수준은 처리량 규칙, 테이블 크기, **max\_parallel\_workers** 설정 등에 따라 결정된다.

**참고**

병렬 처리 수준을 높게 설정하면 개별 질의의 성능은 향상될 수 있지만, 과도하게 높은 값은 리소스 경쟁으로 인해 시스템 전체의 성능이 오히려 저하될 수 있다.

**참고**

이 파라미터의 최대값은 물리 코어 수와 32 중 작은 값으로 자동으로 제한되어, 개별 병렬 연산이 서버 리소스를 과도하게 점유하는 것을 방지하여, 시스템 전체의 안정성을 보장한다.

**max\_subquery\_cache\_size**

**max\_subquery\_cache\_size**는 서브 쿼리 캐시의 크기를 설정하기 위한 파라미터이다. 값 뒤에 B, K, M으로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **2,097,152** (2M) 바이트, 최소값은 **0**, 그리고 최대값은 **16,777,216** (16M) 바이트이다. 서브 쿼리 캐시는 질의의 서브 쿼리 개수만큼 할당되며, 주질의가 종료될 때 할당 해제된다.

**max\_subquery\_cache\_size**이 0으로 설정되거나, **NO\_SUBQUERY\_CACHE** 힌트가 명시되거나, 저장공간이 부족한 경우 서브 쿼리 캐시가 사용되지 않는다.

**memoize\_memory\_limit**

**memoize\_memory\_limit**는 **MEMOIZE** 최적화가 캐시 하나에 사용할 수 있는 메모리 크기를 설정하기 위한 파라미터이다. 값 뒤에 B, K, M으로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **2,097,152** (2M) 바이트이며, 최소값은 **0**이다.

이 파라미터가 **0**으로 설정되면 **MEMOIZE** 최적화가 비활성화된다. **MEMOIZE** 최적화에는 힌트가 없으므로 이 파라미터가 유일한 제어 수단이며, 다음과 같이 세션 단위로도 변경할 수 있다.

```
SET SYSTEM PARAMETERS 'memoize_memory_limit=0';
```

이 값은 캐시 하나에 대한 한도이다. 캐시는 MEMOIZE가 적용되는 조인마다, 병렬 실행 시에는 워커 스레드마다 별도로 할당되므로 하나의 질의가 사용하는 전체 캐시 메모리는 이 값보다 커질 수 있다.

#### 참고

이 파라미터의 값은 질의 계획을 캐시하는 키에 포함되므로, 값을 변경하면 같은 질의라도 별도의 질의 계획이 생성된다.

**sort\_buffer\_size**는 정렬을 수행하는 질의에서 사용되는 버퍼의 크기를 설정하기 위한 파라미터이다. 서버는 각 클라이언트의 정렬 요청마다 하나의 정렬 버퍼를 할당하며, 정렬을 완료한 후에는 할당되었던 버퍼 메모리를 해제한다. 정렬을 수행하는 질의로는 SELECT 정렬 질의 뿐만 아니라 인덱스 생성 질의도 포함된다.

값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $128 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **2M**)이고, 최소값은  $1 * db\_page\_size$  ( $db\_page\_size$ 가 16K일 때 **16K**)이다.

### temp\_file\_memory\_size\_in\_pages

**temp\_file\_memory\_size\_in\_pages**는 질의에 관한 임시 결과를 캐시하는 버퍼 페이지 개수를 설정하기 위한 파라미터로 기본값은 **4**이며, 최대값은 20까지 허용된다.

- 필요한 메모리 크기 = 임시 메모리 버퍼 페이지 수 (**temp\_file\_memory\_size\_in\_pages**) \* 데이터베이스 페이지 크기(page size)
- 임시 메모리 버퍼 페이지 수 = **temp\_file\_memory\_size\_in\_pages** 파라미터 설정값
- 데이터베이스 페이지 크기 = 데이터베이스 생성 시 **cubrid createdb** 유틸리티의 **-s** 옵션에 의해 지정된 페이지 크기 값

임시 결과를 저장하는 공간은 다음과 같다.

- 임시 결과 캐시 버퍼(**temp\_file\_memory\_size\_in\_pages** 시스템 파라미터에 의해 확보된 메모리)
- 일시적 데이터 저장을 위한 영구적 볼륨
- 일시적 볼륨

기존의 저장 공간이 소진되면 일시적 결과 저장을 위해 캐시 버퍼 -> 영구적 볼륨 -> 일시적 볼륨의 순서로 저장 공간이 사용된다.

### thread\_stacksize

**thread\_stacksize**는 스레드의 스택 크기를 설정하기 위한 파라미터로 기본값은 **1048576** 바이트이다. **thread\_stacksize** 파라미터의 설정값은 운영체제가 허용하는 스택 크기를 초과할 수 없다.

### 디스크 관련 파라미터

다음은 데이터베이스 볼륨 정의 및 파일 저장을 위한 디스크 관련 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                     | 타입     | 기본값            | 최소값 | 최대값              |
|-----------------------------|--------|----------------|-----|------------------|
| db_volume_size              | byte   | 512M           | 0   | 20G              |
| dont_reuse_heap_file        | bool   | no             |     |                  |
| log_volume_size             | byte   | 512M           | 20M | 4G               |
| temp_file_max_size_in_pages | int    | -1             |     |                  |
| temp_volume_path            | string | NULL           |     |                  |
| unfill_factor               | float  | 0.1            | 0.0 | 0.3              |
| volume_extension_path       | string | NULL           |     |                  |
| double_write_buffer_size    | byte   | 2,097,152 (2M) | 0   | 33,554,432 (32M) |

| 파라미터 이름             | 타입  | 기본값 | 최소값 | 최대값 |
|---------------------|-----|-----|-----|-----|
| data_file_os_advice | int | 0   | 0   | 6   |

## db\_volume\_size

**db\_volume\_size**는 다음과 같은 값을 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **512M**이다.

- **cubrid createdb**와 **cubrid addvoldb** 유틸리티에서 **--db-volume-size** 옵션을 생략했을 때 생성되는 데이터베이스 볼륨의 기본 크기
- 데이터베이스 볼륨 공간을 모두 사용하면 자동으로 추가되는 볼륨의 최대 크기 (자동 추가 볼륨은 필요량만큼 점진적으로 증가되며 db\_volume\_size에 초과한 경우 다른 추가 볼륨으로 생성됨)

### 참고

실제 볼륨 크기는 항상 64개 디스크 섹터의 배수로 올림된다. 섹터 크기는 페이지의 크기에 따라 달라지므로, 64개 섹터 크기는 페이지 크기 4k, 8k 또는 16k 각각에 대해 16M, 32M 또는 64M이다.

## dont\_reuse\_heap\_file

**dont\_reuse\_heap\_file** 은 테이블 삭제(**DROP TABLE**)로 인해 삭제된 힙 파일을 새로운 테이블 생성(**CREATE TABLE**) 시 재사용하지 않도록 설정하는 파라미터로, no로 설정되면 삭제된 힙 파일을 재사용하고, yes로 설정되면 삭제된 힙 파일을 재사용하지 않는다. 기본값은 **no** 이다.

## log\_volume\_size

**log\_volume\_size**는 **cubrid createdb** 유틸리티에서 **--log-volume-size** 옵션이 생략되었을 때 로그 볼륨 파일의 기본 크기를 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **512M** 이다.

## temp\_file\_max\_size\_in\_pages

**temp\_file\_max\_size\_in\_pages** 는 일시적 볼륨을 확장할 수 있는 최대 페이지 수를 설정하는 파라미터이다. 기본값은 **-1**이며 일시적 볼륨이 무제한 디스크 공간을 차지할 수 있

음을 뜻한다. 제한을 두기 위해서는 양수 값으로 설정할 수 있으며 설정된 값을 초과하면 오류가 표시되고 일부 큰 질의가 취소될 수 있다.

이 파라미터를 **0**으로 설정하면 일시적 볼륨이 자동으로 생성되지 않으며 관리자가 **cubrid addvoldb** 유틸리티를 사용해 일시적 데이터를 저장하기 위한 용도로 영구적 볼륨을 생성해야 한다.

자세한 사항은 다음을 참고하도록 한다. **일시적 볼륨(Temporary Volume)**

### temp\_volume\_path

**temp\_volume\_path**는 복잡한 질의문이나 정렬 수행을 위하여 자동으로 생성되는 일시적 임시 볼륨(temporary temp volume)의 디렉터리를 지정하는 파라미터로 기본값은 데이터베이스 생성 시에 설정된 볼륨 위치이다.

### unfill\_factor

**unfill\_factor**는 데이터 갱신에 대비하여 힙(heap) 페이지로 할당되는 디스크 공간의 비율을 정의하기 위한 파라미터로 기본값은 **0.1**로 10%의 여유 공간이 설정된다. 원칙적으로, 테이블의 데이터는 물리적인 순서대로 삽입되지만, 데이터가 원래 크기보다 큰 데이터로 갱신되어 해당 페이지의 저장 공간이 부족하면 갱신된 데이터는 다른 페이지에 재배치되어야 하므로 성능이 저하될 수 있다. 이를 방지하기 위하여 **unfill\_factor** 파라미터를 통해 힙 페이지 공간 비율을 설정할 수 있고, 최대값은 0.3(30%)까지 허용된다. 한편, 데이터 갱신이 거의 발생하지 않는 데이터베이스에서는 이 파라미터를 0.0으로 설정하여 데이터 갱신을 위한 힙 페이지 공간을 할당하지 않을 수 있고, **unfill\_factor** 파라미터의 값이 음수거나 최대값보다 크게 설정되는 경우에는 기본값(**0.1**)이 적용된다.

### volume\_extension\_path

**volume\_extension\_path**는 **cubrid addvoldb** 유틸리티로 추가 볼륨을 생성할 때 추가 볼륨의 경로를 지정하는 **-F** 옵션을 생략하면 기본 경로로 사용할 경로를 지정하는 파라미터이다. 기본값은 데이터베이스 생성 시에 설정된 볼륨 위치이다.

### double\_write\_buffer\_size

**double\_write\_buffer\_size** 는 이중 쓰기 버퍼 (Double Write Buffer, DWB)의 메모리와 디스크 공간을 설정할 수 있는 파라미터이다. 이 크기를 0으로 설정함으로써 Partial I/O를 방지하기 위한 DWB를 사용하지 않을 수 있다. 기본적으로 DWB는 활성화되어 있으며, 기본값은 **2M** 이다. 해당 값은 **바이트 단위(byte)**로 동작하므로, 최소 2,097,152 바이트에서 최대 33,554,432 바이트 사이의 값으로 설정해야 한다.

### data\_file\_os\_advise

**data\_file\_os\_advise**는 I/O 성능을 향상시키기 위해 사용되는 유닉스 전용 파라미터이다. 파라미터 설정값은 *posix\_fadvise()* 플래그로 변환된다. (플래그에 대한 자세한 내용은 [여기](#)를 참고하자.)

| 파라미터 값 | posix_fadvise 플래그     |
|--------|-----------------------|
| 0      | 0                     |
| 1      | POSIX_FADV_NORMAL     |
| 2      | POSIX_FADV_SEQUENTIAL |
| 3      | POSIX_FADV_RANDOM     |
| 4      | POSIX_FADV_NOREUSE    |
| 5      | POSIX_FADV_WILLNEED   |
| 6      | POSIX_FADV_DONTNEED   |

#### ⚠ 경고

posix\_fadvise 플래그와 데이터 액세스 방법을 완벽히 이해해야 한다. 파라미터 설정은 성능 향상을 도울 수도 있지만 잘못 사용할 경우 성능을 하락시킬 수도 있다. 대부분의 시나리오에서 디폴트 값을 사용하는 것이 가장 좋다.

### 오류 메시지 관련 파라미터

다음은 CUBRID에 의해 기록되는 오류 메시지의 처리에 관한 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                          | 타입     | 기본값     |
|----------------------------------|--------|---------|
| call_stack_dump_activation_level | string | DEFAULT |

| 파라미터 이름                           | 타입     | 기본값                            |
|-----------------------------------|--------|--------------------------------|
| call_stack_dump_deactivation_list | string | NULL                           |
| call_stack_dump_on_error          | bool   | no                             |
| error_log                         | string | cub_client.err, cub_server.err |
| error_log_level                   | string | NOTIFICATION                   |
| error_log_warning                 | bool   | no                             |
| error_log_size                    | int    | 512M                           |

### call\_stack\_dump\_activation\_list

**call\_stack\_dump\_activation\_list**는 모든 오류에 대해 콜-스택을 서버 에러 로그 파일 (\$CUBRID/log/server 디렉터리에 위치)에 덤프하지 않기로 설정한 상태에서, 예외적으로 콜-스택을 덤프할 특정 오류 번호를 지정하기 위한 파라미터이다. 따라서, **call\_stack\_dump\_activation\_list** 파라미터는 **call\_stack\_dump\_on\_error**의 값이 **no**인 경우에만 효력이 있다.

값을 설정하지 않을 경우 기본값은 “DEFAULT” 키워드이며, 다음 오류들을 포함한다. “DEFAULT” 키워드는 다른 오류 번호와 함께 사용될 수 있다.

| 오류 번호 | 오류 메시지                                                                            |
|-------|-----------------------------------------------------------------------------------|
| -2    | Internal system failure: no more specific information is available.               |
| -7    | Trying to format disk volume xxx with an incorrect value xxx for number of pages. |



| 오류 번호 | 오류 메시지                                                                                                       |
|-------|--------------------------------------------------------------------------------------------------------------|
| -13   | An I/O error occurred while reading page xxx of volume xxx.                                                  |
| -14   | An I/O error occurred while writing page xxx of volume xxx.                                                  |
| -17   | Internal error: fetching deallocated pageid xxx of volume xxx.                                               |
| -19   | Internal error: pageptr = xxx of page xxx of volume xxx is not fixed.                                        |
| -21   | Internal error: unknown sector xxx of volume xxx.                                                            |
| -22   | Internal error: unknown page xxx of volume xxx.                                                              |
| -45   | Slot xxx on page xxx of volume xxx is allocated to an anchored record. A new record cannot be inserted here. |
| -46   | Internal error: slot xxx on page xxx of volume xxx is not allocated.                                         |
| -48   | Accessing deleted object xxx xxx xxx.                                                                        |
| -50   | Internal error: relocation record of object xxx xxx xxx may be corrupted.                                    |
| -51   | Internal error: object xxx xxx xxx may be corrupted.                                                         |

| 오류 번호 | 오류 메시지                                                                                                                                                |
|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| -52   | Internal error: object overflow address xxx xxx xxx may be corrupted.                                                                                 |
| -76   | Your transaction (index xxx, <b>xxx@xxx xxx</b> ) timed out waiting on xxx on page xxx xxx. You are waiting for user(s) xxx to release the page lock. |
| -78   | Internal error: an I/O error occurred while reading logical log page xxx (physical page xxx) of xxx.                                                  |
| -79   | Internal error: an I/O error occurred while writing logical log page xxx (physical page xxx) of xxx.                                                  |
| -81   | Internal error: logical log page xxx may be corrupted.                                                                                                |
| -90   | Redo logging is always a page level logging operation. A data page pointer must be given as part of the address.                                      |
| -96   | Media recovery may be needed on volume xxx.                                                                                                           |
| -97   | Internal error: unable to find log page xxx in log archives.                                                                                          |
| -313  | Object buffer underflow while reading.                                                                                                                |
| -314  | Object buffer overflow while writing.                                                                                                                 |

| 오류 번호 | 오류 메시지                                                                                                               |
|-------|----------------------------------------------------------------------------------------------------------------------|
| -407  | Unknown key xxx referenced in B+tree index {vfid: (xxx, xxx), rt_pgid: xxx, key_type: xxx}.                          |
| -414  | Unknown class identifier: xxx xxx xxx.                                                                               |
| -415  | Invalid class identifier: xxx xxx xxx.                                                                               |
| -416  | Unknown representation identifier: xxx.                                                                              |
| -417  | Invalid representation identifier: xxx.                                                                              |
| -583  | Trying to allocate an invalid number (xxx) of pages.                                                                 |
| -603  | Internal Error: Sector/page table of file VFID xxx xxx seems corrupted.                                              |
| -836  | LATCH ON PAGE(xxx xxx) TIMEDOUT                                                                                      |
| -859  | LATCH ON PAGE(xxx xxx) ABORTED                                                                                       |
| -890  | Partition failed.                                                                                                    |
| -891  | Appropriate partition does not exist.                                                                                |
| -976  | Internal error: Table size overflow (allocated size: xxx, accessed size: xxx) at file table page xxx xxx(volume xxx) |

| 오류 번호 | 오류 메시지                                                                                                                      |
|-------|-----------------------------------------------------------------------------------------------------------------------------|
| -1040 | HA generic: xxx.                                                                                                            |
| -1075 | Descending index scan aborted because of lower priority on B+tree with index identifier: (vfid = (xxx, xxx), rt_pgid: xxx). |

다음은 -115, -116번의 오류 번호에 대해서만 콜-스택 덤프가 수행되도록 파라미터를 설정한 예제이다.

```
call_stack_dump_on_error= no
call_stack_dump_activation_list=-115,-116
```

다음은 -115, -116번의 오류 번호와 “DEFAULT” 오류 번호에 대해 콜-스택 덤프가 수행되도록 파라미터를 설정한 예제이다.

```
call_stack_dump_on_error= no
call_stack_dump_activation_list=-115,-116, DEFAULT
```

### call\_stack\_dump\_deactivation\_list

**call\_stack\_dump\_deactivation\_list**는 모든 오류에 대해 콜-스택 덤프를 설정한 상태에서, 예외적으로 콜-스택을 덤프하지 않는 특정 오류 번호를 지정하기 위한 파라미터이다. 따라서, **call\_stack\_dump\_deactivation\_list** 파라미터는 **call\_stack\_dump\_on\_error**의 값이 **yes**인 경우에만 효력이 있다.

다음은 -115, -116번의 오류 번호를 제외한 나머지 오류에 대해서 콜-스택 덤프를 수행하기 위해 파라미터를 설정한 예제이다.

```
call_stack_dump_on_error= yes
call_stack_dump_deactivation_list=-115,-116
```

### call\_stack\_dump\_on\_error

**call\_stack\_dump\_on\_error**는 데이터베이스 서버에서 오류가 발생했을 때 콜-스택을 덤프할지 결정하기 위한 파라미터이다. “no”로 설정되면 모든 오류에 대해서 콜-스택을 덤프하지 않고, “yes”로 설정되면 모든 오류에 대해서 콜-스택을 덤프한다. 기본값은 **no**이다.

## error\_log

**error\_log** 는 데이터베이스 서버에 오류가 발생하는 경우, 에러 로그가 저장되는 파일 이름을 지정하기 위한 서버/클라이언트 파라미터이다. 에러 로그가 저장되는 파일명의 작성 규칙은 `<database_name>_<date>_<time>.err` 이다. 한편 시스템이 데이터베이스 서버 정보를 찾을 수 없는 오류에 대해서는 에러 로그 파일명의 작성 규칙을 따를 수 없다. 따라서, **cubrid.err** 파일에 오류 로그를 기록한다. **cubrid.err** 에러 로그 파일은 **\$CUBRID/log/server** 디렉터리에 저장된다.

## error\_log\_level

**error\_log\_level** 은 에러 심각성(severity) 수준에 따라 에러 로그 파일에 저장할 에러 메시지를 지정할 수 있는 서버 파라미터이다. 에러 심각성 수준은 가장 낮은 수준인 **WARNING** 부터 가장 심각한 수준인 **FATAL** 까지 총 5단계로 구성되며, 그에 따른 에러 메시지 포함 관계는 **FATAL < ERROR < SYNTAX < NOTIFICATION < WARNING** 이다. 기본값은 **NOTIFICATION** 이며, 이 경우 **FATAL** , **ERROR** , **SYNTAX** , **NOTIFICATION** 에 해당하는 에러 메시지가 에러 로그 파일에 기록된다.

## error\_log\_warning

**error\_log\_warning** 은 에러 심각성(severity) 수준이 **WARNING** 인 에러 메시지의 출력 여부를 설정할 수 있는 서버 파라미터이다. 기본값은 no이다. **WARNING** 메시지가 에러 로그 파일에 저장되도록 하려면, **error\_log\_warning** 의 값을 **yes** 로 설정해야 한다.

## error\_log\_size

**error\_log\_size** 는 에러 로그 파일에서 기록되는 파일의 크기를 지정하는 파라미터로 기본값은 **512M** 이다. 에러 로그 파일의 크기가 이 파라미터의 설정값에 도달하면 `<database_name>_<date>_<time>.err.bak` 파일이 생성된다.

## 동시성/잠금 파라미터

다음은 데이터베이스 서버의 동시성 제어 및 잠금에 관한 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                             | 타입    | 기본값 | 최소값 | 최대값 |
|-------------------------------------|-------|-----|-----|-----|
| deadlock_detection_interval_in_secs | float | 1.0 | 0.1 |     |
| isolation_level                     | int   | 4   | 4   | 6   |

| 파라미터 이름                     | 타입   | 기본값     | 최소값     | 최대값     |
|-----------------------------|------|---------|---------|---------|
| lock_escalation             | int  | 100,000 | 5       |         |
| lock_timeout                | msec | -1(무제한) | 0(대기안함) | INT_MAX |
| rollback_on_lock_escalation | bool | no      |         |         |

### deadlock\_detection\_interval\_in\_secs

**deadlock\_detection\_interval\_in\_secs**는 중단된 트랜잭션에 대해 교착 상태 여부를 탐지하는 주기를 초 단위로 설정하기 위한 파라미터이다. CUBRID 시스템은 교착 상태에 있는 트랜잭션 중 하나를 롤백시켜 교착 상태를 해결한다. 기본값은 1초이며, 최소값은 0.1초이다. 이 값은 0.1초 단위로 올림하여 동작한다. 즉, 입력값이 0.12초이면 0.2초를 입력한 것과 같이 동작한다. 탐지 주기가 길면 오랜 시간동안 교착 상태를 탐지할 수 없으므로 주의한다.

### isolation\_level

**isolation\_level**은 트랜잭션의 격리 수준을 설정하기 위한 파라미터로 격리 수준이 높을수록 트랜잭션의 동시성이 적고 다른 동시성 트랜잭션에 의해 간섭받지 않는다.

**isolation\_level** 파라미터는 격리 수준을 의미하는 4에서 6까지의 정수값 또는 문자열로 설정하며, 기본값은 **READ COMMITTED**이다. 각 격리 수준 및 파라미터 설정값에 대한 자세한 내용은 **트랜잭션 격리 수준** 과 다음 표를 참조한다.

| 격리 수준           | isolation_level 파라미터 설정값                                 |
|-----------------|----------------------------------------------------------|
| SERIALIZABLE    | "TRAN_SERIALIZABLE" or 6                                 |
| REPEATABLE READ | "TRAN_REP_CLASS_REP_INSTANCE" or<br>"TRAN_REP_READ" or 5 |

## 격리 수준

## isolation\_level 파라미터 설정값

READ COMMITTED

“TRAN\_REP\_CLASS\_COMMIT\_INSTANCE”  
or “TRAN\_READ\_COMMITTED” or  
“TRAN\_CURSOR\_STABILITY” or 4

- **TRAN\_SERIALIZABLE** : 가장 높은 수준의 일관성을 보장하는 격리 수준이며, **SERIALIZABLE 격리 수준** 을 참고한다.
- **TRAN\_REP\_READ** : snapshot isolation 때문에 더티 읽기, 반복 불가능한 읽기 및 유령 읽기가 발생하지 않는 수준이며, **REPEATABLE READ 격리 수준** 을 참고한다.
- **TRAN\_READ\_COMMITTED** : 반복 불가능한 읽기(unrepeatable read)가 발생할 수 있는 격리 수준이며, **READ COMMITTED 격리 수준** 을 참고한다.

 참고

9.3 이하 버전에서는 다음의 격리 수준을 추가로 지원한다. 10.0부터는 다량의 동시 트랜잭션 처리 시 MVCC 기법을 적용하여 격리 수준을 낮추지 않고도 동시성을 더 잘 보장할 수 있게 되었기 때문에, 아래의 낮은 격리 수준을 더 이상 사용하지 않게 되었다.

- **TRAN\_REP\_CLASS\_UNCOMMIT\_INSTANCE** : 더티 읽기(dirty read)가 발생될 수 있는 격리 수준이다.
- **TRAN\_COMMIT\_CLASS\_COMMIT\_INSTANCE** : 반복 불가능한 읽기(unrepeatable read)가 발생될 수 있고, 데이터 조회 중에 다른 트랜잭션에 의한 테이블 스키마의 변경이 허용되는 격리 수준이다.
- **TRAN\_COMMIT\_CLASS\_UNCOMMIT\_INSTANCE** : 더티 읽기(dirty read)가 발생될 수 있고, 데이터 조회 중에 다른 트랜잭션에 의한 테이블 스키마의 변경이 허용되는 격리 수준이다.

**lock\_escalation**

**lock\_escalation** 은 행에 대한 잠금이 테이블 잠금으로 확대되기 전에 개별 행에 허용되는 최대 잠금의 개수를 설정하기 위한 파라미터로 기본값은 **100,000** 이다. **lock\_escalation** 파라미터의 설정값이 작으면, 메모리 잠금 관리에 의한 오버헤드가 적은 반면 동시성은 줄어든다. 반대로 설정값이 크면 메모리 잠금 관리에 의한 오버헤드가 큰 반면 동시성이 향상된다.

## lock\_timeout

**lock\_timeout** 은 잠금 대기 시간을 지정하기 위한 클라이언트 파라미터로 지정된 시간 이내에 잠금이 허용되지 않으면 해당 트랜잭션이 취소되고 오류가 반환된다. 기본값인 **-1** 로 설정하면 잠금이 허용될 때까지의 대기 시간이 무제한이고, 0으로 설정하면 잠금을 대기하지 않는다.

s, min, h 단위를 지정할 수 있으며 각각 seconds, minutes, hours를 의미한다. 단위 생략 시 기본 단위는 밀리초(ms)이며, 밀리초로 설정한 값은 초 단위로 올림된다. 예를 들어, 1ms는 1s가 되며, 1001ms는 2s가 된다.

## rollback\_on\_lock\_escalation

잠금 에스컬레이션 발생 시 트랜잭션의 롤백 여부를 지정한다. 기본값은 **no**이다.

이 파라미터가 **yes**로 설정되면, 잠금 에스컬레이션 발생 시점에 에스컬레이션 없이 에러 로그를 기록하고, 해당 잠금 요청은 실패하면서 트랜잭션을 롤백한다. no로 설정되면 잠금 에스컬레이션이 수행되고 트랜잭션을 계속 진행한다.

잠금 에스컬레이션이 발생하면 레코드 잠금이 테이블 잠금으로 전환되고, 잠금(lock)을 해제하는 시간이 오래 걸리면서 해당 테이블에 대한 다른 트랜잭션의 접근이 불가하게 되는 상황이 발생할 수 있다. 그렇다고 잠금 에스컬레이션이 발생하는 레코드 잠금 개수를 지정하는 **lock\_escalation** 파라미터 값을 크게 하면 메모리 자원을 많이 사용하는 문제가 발생할 수 있다.

## 로깅 관련 파라미터

다음은 CUBRID 데이터베이스의 백업과 복구에 이용되는 로그에 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                | 타입   | 기본값                     | 최소값                   | 최대값             |
|------------------------|------|-------------------------|-----------------------|-----------------|
| adaptive_flush_control | bool | yes                     |                       |                 |
| background_archiving   | bool | yes                     |                       |                 |
| checkpoint_every_size  | byte | 10,000<br>log_page_size | * 10<br>log_page_size | * log_page_size |



| 파라미터 이름                             | 타입   | 기본값                      | 최소값                    | 최대값                        |
|-------------------------------------|------|--------------------------|------------------------|----------------------------|
| checkpoint_interval                 | msec | 6min                     | 1min                   | 35,791,394min              |
| checkpoint_sleep_msecs              | msec | 1                        | 0                      |                            |
| force_remove_log_archives           | bool | yes                      |                        |                            |
| log_buffer_size                     | byte | 16k<br>log_page_size *   | 128<br>log_page_size * | INT_MAX<br>log_page_size * |
| log_max_archives                    | int  | INT_MAX                  | 0                      | INT_MAX                    |
| log_trace_flush_time                | int  | 0                        | 0                      | INT_MAX                    |
| max_flush_size_per_second           | byte | 10,000<br>db_page_size * | 1<br>db_page_size *    | INT_MAX<br>db_page_size *  |
| remove_log_archive_interval_in_secs | sec  | 0                        | 0                      |                            |
| sync_on_flush_size                  | byte | 200<br>db_page_size *    | 1<br>db_page_size *    | INT_MAX<br>db_page_size *  |
| ddl_audit_log                       | bool | no                       |                        |                            |
| ddl_audit_log_size                  | byte | 10M                      | 10M                    | 2G                         |

## adaptive\_flush\_control

**adaptive\_flush\_control**는 내려쓰기(flush) 작업 중에 50ms마다 작업 상태에 따라 내려쓰기할 용량(flush capacity)을 자동 조정하는 파라미터이며, 기본값은 **yes**이다. 즉, 특정 시점에 **INSERT** 또는 **UPDATE** 연산이 집중되어 내려쓰기한 페이지 수가 **max\_flush\_pages\_per\_second** 파라미터 값에 도달하면 이 용량을 증가시키고, 이에 도달하지 못하면 이 용량을 감소시킨다. 이처럼 워크로드에 따라 주기적으로 내려쓰기 용량을 조정하여 I/O 부하를 분산할 수 있다.

## background\_archiving

**background\_archiving**은 특정 시점마다 주기적으로 임시 보관 로그를 생성하도록 하는 파라미터로서, 보관 로그 작업으로 인한 디스크 I/O 부하를 분산시키고자 할 때 유용하다. 기본값은 **yes**이다.

## checkpoint\_every\_size

**checkpoint\_every\_size**는 체크포인트가 수행되는 주기를 로그 페이지 단위로 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $10,000 * \text{log\_page\_size}$  (log\_page\_size가 16K이면 **156.25M**)이다.

특정 시간대에 **INSERT** / **UPDATE** 가 집중되는 서비스 환경에서는 **checkpoint\_every\_size** 파라미터의 설정값을 작게 설정하여 체크포인트 시점에 I/O 부하를 분산할 수 있다.

체크포인트는 특정 시점에 데이터 버퍼에 있는 페이지 중 임시 페이지를 제외한 모든 수정된 페이지를 데이터베이스 볼륨(디스크)에 기록하는 작업이다. 체크포인트는 데이터베이스 장애 발생 이후 복구 시 체크포인트가 발생하기 전의 트랜잭션 로그를 반영할 필요가 없게 하여 복구 시간을 단축시키는 효과가 있다. 다만, 체크포인트 작업으로 인해 디스크 I/O가 발생하여 DB 운영에 영향을 끼칠 수 있으므로 체크포인트 주기를 적절하게 설정해야 한다.

### 참고

CUBRID에서 체크포인트는 4가지 방법으로 제공되는데, 다음의 2가지는 cubrid.conf의 설정에 의해 제공된다.

- **checkpoint\_interval**: 체크포인트가 완료된 이후 이 파라미터의 설정값이 경과된 시간마다 체크포인트 작업이 주기적으로 수행된다.
- **checkpoint\_every\_size**: 트랜잭션 로그 파일의 크기가 이 파라미터의 설정값에 도달하는 시점마다 체크포인트 작업이 주기적으로 수행된다.

이상 두 개의 파라미터 조건 중 하나만 만족하면 체크포인트가 수행된다.

다음의 2가지는 사용자의 명령으로 제공된다.

- CSQL 인터프리터를 “DBA” 사용자로 실행한 이후 “;checkpoint” 명령을 수행하면 체크포인트가 수행된다.

참고로 체크포인트 수행 도중에 백업을 수행하면, 체크포인트가 종료될 때까지 백업 명령은 대기 상태가 된다.

## checkpoint\_interval

**checkpoint\_interval**은 체크포인트가 수행되는 주기를 설정하는 파라미터이다. ms, s, min, h 단위를 지정할 수 있으며 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위 생략 시 기본 단위는 밀리초(ms)이며, 밀리초로 설정한 값은 초 단위로 올림된다. 예를 들어, 1ms는 1s가 되며, 1001ms는 2s가 된다. 기본값은 **6min**이고, 최소값은 1min이다. 최대값은 35791394min이다.

## checkpoint\_sleep\_msecs

체크포인트가 발생할 때 버퍼의 데이터를 디스크에 플러시하는 작업을 천천히 진행하게 하는 파라미터이다. 기본값은 **1** (밀리초)이다.

## force\_remove\_log\_archives

**force\_remove\_log\_archives**는 **log\_max\_archives**로 지정한 개수의 최근 보관 로그(log archive) 파일을 제외한 나머지 파일의 삭제 허용 여부를 지정하는 파라미터로서, 기본값은 **yes**이다.

파라미터 값을 **yes** 로 설정하면, **log\_max\_archives**로 지정한 개수의 최근 보관 로그 파일을 제외한 나머지 파일이 삭제된다.

파라미터 값을 **no** 로 설정하면, 보관 로그 파일이 삭제되지 않지만, 예외적으로 **ha\_mode**를 on으로 설정하면 HA 관련 프로세스에 필요한 보관 로그 파일과 **log\_max\_archives**로 지정한 개수의 최근 보관 로그 파일을 제외한 나머지 파일이 삭제된다.

CUBRID HA 환경을 구축하고자 하는 사용자는 **환경 설정**을 참고한다.

## log\_buffer\_size

**log\_buffer\_size** 는 메모리에 캐시되는 로그 버퍼의 크기를 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $128 * \text{log\_page\_size}$  (log\_page\_size가 16K이면 **2M**) 이다.

**log\_buffer\_size** 파라미터의 설정값이 크면 데이터베이스 수정 연산이 많고, 수행 시간이 긴 트랜잭션 많은 환경에서는 디스크 I/O가 감소되어 성능이 향상될 수 있다. CUBRID

의 MVCC 시스템은 이전 버전의 값에 접근하는 것과 데이터베이스에서 삭제된 값을 회수하기 위해 로그에 의존한다. 따라서 CUBRID가 설치된 시스템의 메모리 크기 및 작업 연산의 크기를 고려하여 적당한 값으로 설정할 것을 권장한다.

- 필요한 메모리 크기 = 로그 버퍼 크기(**log\_buffer\_size**)

## log\_max\_archives

**log\_max\_archives** 는 보존할 보관 로그 파일의 최대 개수를 설정하는 파라미터이다. 최소값은 0이며, 기본값은 **INT\_MAX** (2,147,483,647)이다. 이 파라미터의 동작 방식은 **force\_remove\_log\_archives** 설정 여부에 따라 달라질 수 있다. 예를 들어, cubrid.conf 파일에서 **log\_max\_archives** = 3이고 **force\_remove\_log\_archives** = yes로 설정되어 있을 경우, 가장 최근의 아카이브 로그 파일 3개만 유지된다. 즉, 네 번째 아카이브 로그 파일이 생성되면, 가장 오래된 아카이브 로그 파일이 자동으로 삭제된다. 삭제된 아카이브 로그에 대한 정보는 \*\_lginf 파일에 기록된다. 사용자가 **ha\_mode=off**, **force\_remove\_log\_archives=n** 으로 설정하면, 시스템은 **log\_max\_archives**에 사용자가 입력한 값 대신, **INT\_MAX(2,147,483,647)**값을 자동으로 할당한다.

하지만 활성화된 트랜잭션이 기존 보관 로그 파일을 여전히 참조하고 있다면, 해당 보관 로그 파일은 삭제되지 않는다. 즉, 어떤 트랜잭션이 첫 번째 보관 로그 파일이 생성되는 시점에서 시작되어 다섯 번째 보관 로그 파일이 생성되는 시점까지도 종료되지 않았다면 첫 번째 보관 로그 파일은 삭제되지 않는다.

그리고 보관 로그 파일의 정보가 데이터베이스 볼륨에 아직 반영되어 있지 않은 경우에도 해당 보관 로그 파일은 삭제되지 않는다. (체크포인트가 발생한 이후의 보관 로그는 데이터 버퍼의 수정된 페이지 정보를 가지고 있으므로 데이터베이스 복구를 위해 필요하다.)

DB 운영 중에 **log\_max\_archives**의 값을 동적으로 변경하는 경우, 변경된 값은 새로운 보관 로그 파일이 생성될 때 적용된다. 예를 들어, 해당 값을 10에서 5로 변경한 경우, 새로운 보관 로그 파일이 생성되는 시점에 오래된 파일 5개를 삭제한다.

CUBRID HA 환경을 구축하고자 하는 사용자는 **환경 설정**을 참고한다.

### 참고

2008 R4.3 이하 버전과 9.1 버전에서 **log\_max\_archives**는 HA 환경에서 복제 로그 파일의 최대 보존 개수를 지정할 때도 사용되었으나, 2008 R4.4와 9.2 이상 버전에서는 cubrid\_ha.conf의 **ha\_copy\_log\_max\_archives** 파라미터가 그 역할을 대신하게 되었다.

## log\_trace\_flush\_time

이 파라미터에 설정한 시간보다 로그 플러싱 시간이 오래 걸리는 경우 해당 이벤트가 데이터베이스 서버 로그에 기록된다.

기록되는 정보의 예는 다음과 같다.

```
03/18/14 10:20:45.889 - LOG_FLUSH_THREAD_WAIT
total flush count: 1 page(s)
total flush time: 310 ms
time waiting for log writer: 308 ms
last log writer info
 client: DBA@cdb037.cub|copylogdb(15312)
 time spent by log writer: 308 ms
```

- LOG\_FLUSH\_THREAD\_WAIT: 이벤트 이름
- total flush count: 이벤트 발생 당시 플러시(flush)한 페이지 수
- total flush time: 총 플러시 소요 시간
- time waiting for log writer: LFT(Log Flushing Thread)가 LWT(Log Writer Thread)를 대기한 시간
- last log writer info
  - DBA@cdb037.cub|copylogdb(15312): LFT를 대기하게 한 LWT와 관련된 copylogdb 정보 <사용자@호스트명|클라이언트명(pid)>
  - time spent by log writer: LWT에서 측정한 LWT 소요 시간(일반적으로 time waiting for log writer와 동일)

## max\_flush\_size\_per\_second

**max\_flush\_size\_per\_second**는 버퍼로부터 디스크로 내려쓰기(flush) 작업을 수행할 때, 내려쓰기할 최대 용량 (flush capacity)을 설정하기 위한 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $10,000 * db\_page\_size$  ( $db\_page\_size$ 가 16K이면 **156.25M**)이다. 즉, 이 파라미터 설정을 통해 1초당 내려쓰기할 최대 용량을 제어하여, 특정 시점에 I/O 부하가 집중되는 현상을 방지할 수 있다.

만약, 특정 시점에 **INSERT** 또는 **UPDATE** 연산이 집중되어 이 파라미터에 의해 설정된 최대 용량에 도달하면, 로그 페이지만 내려쓰기를 수행하고 데이터 페이지는 더 이상 디스크로 내려쓰지 않는다. 따라서, 이 파라미터는 서비스 환경의 워크로드를 고려하여 적절한 값을 설정해야 한다.

## remove\_log\_archive\_interval\_in\_secs

**log\_max\_archives**에서 지정한 개수를 초과한 보관 로그는 체크포인트가 발생하는 시점에 삭제하게 되는데, 데이터 마이그레이션이나 대량 배치와 같은 작업이 수행되는 경우

많은 양의 보관 로그가 쌓였다가 일시에 지워지는 일이 자주 발생한다. 이렇게 파일 삭제가 한꺼번에 발생한다면 데이터베이스 서버의 I/O 부하가 급격히 증가하므로, 이 부담을 줄일 필요가 있다.

**remove\_log\_archive\_interval\_in\_secs** 파라미터는 이러한 부담을 줄이고자 보관 로그의 삭제를 천천히 진행하게 한다. 기본값은 0(초)이다. 대량 배치와 같은 작업이 자주 발생하는 상황에서, 디스크 공간이 충분하다면 삭제 간격을 60초 정도로 줄 것을 권장한다.

### sync\_on\_flush\_size

**sync\_on\_flush\_size**는 버퍼로부터 데이터 페이지 및 로그 페이지를 내려쓰기한 후, 운영 시스템의 FILE I/O와 동기화를 수행하는 주기를 페이지 단위로 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은  $200 * db\_page\_size$  ( $db\_page\_size$ 가 16K이면 **3.125M**)이다. 즉, 200페이지만큼 내려쓰기 작업이 수행될 때마다 CUBRID 서버는 운영 체제의 FILE I/O와 동기화를 수행한다. I/O 부하와 관련된 파라미터이다.

### ddl\_audit\_log

**ddl\_audit\_log** 로깅 여부를 지정하는 파라미터이다. 기본값은 no 이다. 이 값이 yes로 설정되면, 모든 DDL이 수행 될때 로그 파일에 저장된다. 로그 파일 저장 경로는 \$CUBRID/log/ddl\_audit 이며, 각각의 DDL AUDIT 로그 파일 이름은 **DDL Audit Log** 를 참조 한다.

### ddl\_audit\_log\_size

**ddl\_audit\_log\_size** 는 DDL AUDIT 로그 파일의 최대 크기를 지정한다. 로그 파일이 지정한 크기보다 커지면 DDL AUDIT 로그 파일에 .bak 를 붙인 형식의 이름으로 백업된 후 새 파일에 로그가 기록된다. 크기 설정 값 뒤에 B, K, M, G의 크기 단위를 사용할 수 있으며, 각각 Bytes, Kilobytes, Megabytes 그리고 Gigabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 10M 이고, 최대 2G까지 설정가능하다.

### 트랜잭션 처리 관련 파라미터

다음은 트랜잭션의 커밋 성능 향상을 위한 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름      | 타입   | 기본값 | 최소값 | 최대값 |
|--------------|------|-----|-----|-----|
| async_commit | bool | no  |     |     |

| 파라미터 이름                        | 타입   | 기본값 | 최소값 | 최대값 |
|--------------------------------|------|-----|-----|-----|
| group_commit_interval_in_msecs | msec | 0   | 0   |     |
| dblink_auto_commit             | bool | yes | 0   |     |

### async\_commit

**async\_commit**은 비동기식 커밋 기능을 활성화시키는 파라미터로 기본값인 **no**로 설정하면 비동기식 커밋을 수행하지 않고, **yes**로 설정하면 비동기식 커밋을 수행한다. 비동기식 커밋이란 커밋 로그가 디스크에 플러시되기 전에 클라이언트에게 커밋을 완료 처리하고, 로그 플러시 스레드(LFT)가 로그 플러시를 백그라운드에서 수행하여 커밋 작업의 성능을 향상시키는 기능이다. 로그 플러시가 수행되기 전에 데이터베이스 서버에 장애가 발생하면 이미 커밋 완료된 트랜잭션을 복구할 수 없으므로 주의한다.

### group\_commit\_interval\_in\_msecs

**group\_commit\_interval\_in\_msecs**은 그룹 커밋을 수행하는 간격을 밀리초(msec) 단위로 지정하는 파라미터로 기본값인 **0**으로 설정되면 그룹 커밋을 수행하지 않는다. 그룹 커밋이란 지정된 시간동안 발생한 여러 번의 커밋을 그룹으로 취합하여 커밋 로그가 동시에 디스크에 플러시되도록 하여 커밋 작업의 성능을 향상시키는 기능이다.

### dblink\_auto\_commit

**dblink\_auto\_commit**은 분산 트랜잭션 처리 시, **dblink**를 통한 질의의 커밋 방식에 영향을 주는 설정으로, 기본 값은 **yes**이다.

- 이 파라미터 값을 **yes**로 설정하면, dblink 질의는 실행 즉시 자동으로 커밋된다.
- 반대로 **no**로 설정하면, dblink를 포함한 전체 트랜잭션이 커밋될 때 dblink를 통해 연결된 원격 서버의 질의도 함께 커밋된다.

분산 트랜잭션에서는 dblink 질의부터 먼저 커밋을 시도한다. 만약 이 과정에서 하나라도 커밋에 실패하는 경우 전체 트랜잭션은 취소(rollback)된다.

#### 경고

- 일부 dblink 질의가 이미 커밋된 이후에 전체 트랜잭션 커밋이 실패하더라도, 해당 커밋은 되돌릴 수 없다.

• dblink 트랜잭션에서 savepoint는 지원하지 않는다.

## 구문/타입 관련 파라미터

다음은 CUBRID에서 지원하는 SQL 구문 및 데이터 타입에 관한 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                        | 타입   | 기본값 | 최소값 | 최대값 |
|--------------------------------|------|-----|-----|-----|
| add_column_update_hard_default | bool | no  |     |     |
| alter_table_change_type_strict | bool | yes |     |     |
| allow_truncated_string         | bool | no  |     |     |
| ansi_quotes                    | bool | yes |     |     |
| block_ddl_statement            | bool | no  |     |     |
| block_nowhere_statement        | bool | no  |     |     |
| compat_numeric_division_scale  | bool | no  |     |     |
| create_table_reuseoid          | bool | yes |     |     |



| 파라미터 이름                           | 타입     | 기본값    | 최소값   | 최대값       |
|-----------------------------------|--------|--------|-------|-----------|
| cte_max_recur<br>sions            | int    | 2,000  | 2     | 1,000,000 |
| default_week_for<br>mat           | int    | 0      |       |           |
| group_concat_ma<br>x_len          | byte   | 1,024  | 4     | INT_MAX   |
| intl_check_input_<br>string       | bool   | no     |       |           |
| intl_collation                    | string |        |       |           |
| intl_date_lang                    | string |        |       |           |
| intl_number_lang                  | string |        |       |           |
| json_max_array_i<br>dx            | int    | 65,536 | 1,024 | 1,048,576 |
| max_query_tran                    | int    | 100    | 1     | 32,767    |
| no_backslash_es<br>capes          | bool   | yes    |       |           |
| only_full_group_<br>by            | bool   | no     |       |           |
| oracle_compat_n<br>umber_behavior | bool   | no     |       |           |

| 파라미터 이름                              | 타입   | 기본값       | 최소값 | 최대값        |
|--------------------------------------|------|-----------|-----|------------|
| oracle_style_empt_string             | bool | no        |     |            |
| pipes_as_concat                      | bool | yes       |     |            |
| plus_as_concat                       | bool | yes       |     |            |
| require_like_escape_character        | bool | no        |     |            |
| return_null_on_function_errors       | bool | no        |     |            |
| stored_procedure_return_numeric_size | list | 38,15     | 1,0 | 38,38      |
| string_max_size_bytes                | byte | 1,048,576 | 64  | 33,554,432 |
| unicode_input_normalization          | bool | no        |     |            |
| unicode_output_normalization         | bool | no        |     |            |
| update_use_attribute_references      | bool | no        |     |            |

**add\_column\_update\_hard\_default**

**add\_column\_update\_hard\_default**는 **ALTER TABLE ... ADD COLUMN** 절로 새로운 칼럼을 추가할 때 이 칼럼에 입력할 값을 고정 기본값(hard\_default)으로 제공할지 여부를 설정하는 파라미터로서, 기본값은 **no**이다.

**NOT NULL** 제약 조건이 있고 **DEFAULT** 제약 조건이 없을 때 이 파라미터 값이 **yes**이면 칼럼의 새로운 입력값을 고정 기본값(hard default value)으로 입력하며, **no**이면 에러를 반환한다. 이 파라미터의 값이 **yes**일 때 추가하려는 칼럼의 타입에 고정 기본값이 없으면 에러를 반환한다. 각 타입별 고정 기본값에 대해서는 **ALTER TABLE** 문의 **CHANGE/MODIFY** 절을 참고한다.

```
SET SYSTEM PARAMETERS 'add_column_update_hard_default=yes';

CREATE TABLE tbl (i int);
INSERT INTO tbl VALUES (1),(2);
ALTER TABLE tbl ADD COLUMN j INT NOT NULL;

SELECT * FROM tbl;
```

| i | j |
|---|---|
| 1 | 0 |
| 2 | 0 |

```
SET SYSTEM PARAMETERS 'add_column_update_hard_default=no';

CREATE TABLE tbl (i INT);
INSERT INTO tbl VALUES (1),(2);
ALTER TABLE tbl ADD COLUMN j INT NOT NULL;
```

```
ERROR: Cannot add NOT NULL constraint for attribute "j":
there are existing NULL values for this attribute.
```

### alter\_table\_change\_type\_strict

**alter\_table\_change\_type\_strict**는 타입 변경에 따른 해당 칼럼 값들의 변환 허용 여부를 지정하는 파라미터로서, 기본값은 **yes**이다. 이 파라미터 값이 **no**이면 칼럼의 타입 변경이나 **NOT NULL** 제약 조건을 추가할 때 값의 변경이 발생할 수 있으며, **yes**이면 값의 변경이 발생하지 않는다. 자세한 내용은 **ALTER TABLE** 문의 **CHANGE/MODIFY** 절을 참고한다.

### allow\_truncated\_string

**allow\_truncated\_string**은 INSERT 또는 UPDATE 질의에 사용되는 문자열 작업에 따른 문자열 값의 자르기를 허용할지 여부를 설정하는 매개 파라미터이며 기본값은 **no**이다.

이 파라미터 값이 **no**로 설정되면 INSERT 또는 UPDATE 질의와 관련된 모든 문자열에 대해 문자열 값이 잘리는 것을 허용하지 않는다. 다만, SELECT 질의와 관련된 문자열은 이 설정과 상관없이 문자열 값이 잘릴 수 있다. 이 파라미터 값이 **yes**로 설정되면 (INSERT/UPDATE/SELECT) 질의 유형에 관계없이 문자열 값이 잘리는 것을 허용한다.

### ansi\_quotes

**ansi\_quotes** 는 식별자 처리를 위한 기호 또는 문자열을 감싸는 기호에 관한 파라미터로 기본값은 **yes** 이다. 이 파라미터 값이 yes이면 큰따옴표는 식별자 처리 기호로 해석되고, 작은따옴표는 문자열 처리 기호로 해석된다. 이 값이 no이면 큰 따옴표와 작은 따옴표 모두 문자열 처리 기호로 해석된다.

### block\_ddl\_statement

**block\_ddl\_statement** 는 클라이언트가 수행하는 데이터 정의문(Data Definition Language, DDL)을 제한하기 위한 파라미터로 no로 설정하면 해당 클라이언트의 데이터 정의문 수행을 허용하며, yes로 설정하면 해당 클라이언트의 데이터 정의문 수행을 허용하지 않는다. yes로 설정된 경우 통계 정보 갱신 구문(update statistics)도 제한될 수 있음을 유의하여야 한다. 기본값은 **no** 이다.

### block\_nowhere\_statement

**block\_nowhere\_statement** 는 클라이언트가 수행하는 조건절( **WHERE**)이 없는 **UPDATE** / **DELETE** 문을 제한하기 위한 파라미터로 **no** 로 설정하면 해당 클라이언트의 조건절이 없는 **UPDATE** / **DELETE** 문을 허용하며, yes로 설정하면 해당 클라이언트의 조건절이 없는 **UPDATE** / **DELETE** 문의 수행을 허용하지 않는다. 기본값은 **no** 이다.

### compat\_numeric\_division\_scale

**compat\_numeric\_division\_scale** 은 나눗셈 연산의 결과 값(몫)에 대하여 소수점 이하 자릿수를 몇 자리까지 표시할 것인가를 지정하기 위한 파라미터로 **no** 로 설정하면 몫의 소수점 이하 자릿수가 9개가 되고, **yes** 로 설정하면 몫의 소수점 이하 자릿수가 피연산자의 소수점 이하 자릿수에 따라 결정된다. 기본값은 **no** 이다.

### create\_table\_reuseoid

**create\_table\_reuseoid** 는 테이블 생성시 테이블 옵션을 생략할 경우 **REUSE\_OID** 또는 **DONT\_REUSE\_OID** 옵션을 사용할지를 지정하기 위한 파라미터로 **yes** 로 설정하면 **REUSE\_OID** 테이블 옵션을 가지고 테이블을 생성한다. 기본값은 **yes** 이다.

**REUSE\_OID**과 **DONT\_REUSE\_OID**에 대한 자세한 설명은 **REUSE\_OID** 과 **DONT\_REUSE\_OID** 를 참고한다.

### cte\_max\_recursions

**cte\_max\_recursions** 는 CTE(Common Table Expressions) 질의문의 반복되는 부분을 수행할 때 최대 반복 횟수를 제한하는 파라미터이다. 이 파라미터는 무한 반복과 임시 결과의 크기로 인해 발생할 수 있는 이슈를 예방한다.

## default\_week\_format

**default\_week\_format**은 `WEEK()` 함수 *mode* 인자의 기본값을 설정한다. 기본값은 **0**이다. 자세한 내용은 `WEEK()` 함수를 참고한다.

## group\_concat\_max\_len

**group\_concat\_max\_len** 은 `GROUP_CONCAT()` 함수의 리턴 값의 크기를 제한하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **1,024** 바이트이며, 최소값은 4 바이트, 최대값은 `INT_MAX` 바이트(약2G)이다. `GROUP_CONCAT()` 함수의 결과가 제한을 넘으면 오류가 반환된다.

이 함수는 **string\_max\_size\_bytes** 파라미터의 영향을 받으며, **group\_concat\_max\_len** 값을 **string\_max\_size\_bytes** 보다 크게 설정한 경우 `GROUP_CONCAT()` 결과가 **string\_max\_size\_bytes** 값을 초과하면 오류가 발생한다.

## intl\_check\_input\_string

**intl\_check\_input\_string**은 입력되는 문자열이 사용하는 문자셋에 맞게 입력되는지에 대한 검사 여부를 설정하는 파라미터이다. 기본값은 **no** 이다. 예를 들어, 이 값이 **no**이고 문자셋이 UTF-8일 때 UTF-8 바이트 순서(byte sequence)에 맞지 않는 데이터가 들어오는 경우 비정상적인 동작을 보이거나 심하면 데이터베이스 서버 혹은 응용 프로그램이 비정상 종료될 수도 있다. 하지만 이러한 문제가 없다는 것이 보장된다면 검사하지 않는 것이 성능상 좀더 유리하다.

UTF-8과 EUC-KR만이 검사 대상이며, ISO-8859-1은 한 바이트 인코딩이고 모든 바이트가 유효하므로 검사할 필요가 없다.

## intl\_collation

**intl\_collation** 은 특정 응용 클라이언트에 대해 콜레이션 이름을 지정하는 파라미터로서, 이 파라미터를 지정하면 “SET NAMES” 문을 통해 응용 클라이언트의 콜레이션을 변경하는 것과 같은 동작을 수행한다. 콜레이션 지정은 문자셋을 포함한다.

아래의 두 문장은 같은 동작을 수행한다.

```
SET NAMES utf8;
SET SYSTEM PARAMETERS 'intl_collation=utf8_bin';
```

**intl\_collation** 의 값으로 사용할 수 있는 값은 **콜레이션** 을 참고한다.

## intl\_date\_lang

**intl\_date\_lang** 은 **TIME**, **DATE**, **DATETIME**, **TIMESTAMP** 타입의 값을 입력 또는 출력하는 함수의 인자로 언어 이름이 생략되는 경우, 문자열의 지역화된(localized) 캘린더(월 이름과 요일 이름, 오전/오후 이름) 형식을 지정하는 파라미터이다.

사용할 수 있는 값은 다음과 같다. 단, 이 값들을 모두 사용하려면 내장된 로캘(locale)을 제외한 나머지 로캘에 대해서는 원하는 로캘 라이브러리를 설정해야 한다. 로캘 설정에 대해서는 [로캘 설정](#) 을 참고한다.

| 언어    | 언어의 로캘 이름 |
|-------|-----------|
| 영어    | en_US     |
| 독일어   | de_DE     |
| 스페인어  | es_ES     |
| 프랑스어  | fr_FR     |
| 이탈리아어 | it_IT     |
| 일본어   | ja_JP     |
| 캄보디아어 | km_KH     |
| 한국어   | ko_KR     |
| 터키어   | tr_TR     |
| 베트남어  | vi_VN     |
| 중국어   | zh_CN     |
| 루마니아어 | ro_RO     |

지정된 언어의 캘린더 형식에 따라 입력 문자열을 인식하는 함수는 다음과 같다.

- `TO_DATE()`
- `TO_TIME()`
- `TO_DATETIME()`
- `TO_TIMESTAMP()`
- `STR_TO_DATE()`

지정된 언어의 캘린더 형식에 따라 문자열을 출력하는 함수는 다음과 같다.

- `TO_CHAR()`
- `DATE_FORMAT()`
- `TIME_FORMAT()`

## intl\_number\_lang

**intl\_number\_lang** 은 문자열을 숫자로, 또는 숫자를 문자열로 변환하는 함수들에서 입력되거나 출력되는 문자열에 숫자 형식을 부여할 때 적용할 로캘을 지정하는 파라미터이다. 숫자에 대해 지역화되는 것들은 자릿수 구분 기호와 소수점 기호이다. 일반적으로는 쉼표(.)와 마침표(.)가 쓰이는데, 로캘에 따라 서로 바뀔 수 있다. 예를 들어, 숫자 1000.12(천 소수점 이하 일)은 대부분의 로캘에서는 1,000.12로 쓰이는 반면, tr\_TR 로캘에서는 1.000,12로 쓰인다.

지정된 언어의 숫자 형식에 따라 입력 문자열을 인식하는 함수는 다음과 같다.

- `TO_NUMBER()`

지정된 언어의 숫자 형식에 따라 문자열을 출력하는 함수는 다음과 같다.

- `FORMAT()`
- `TO_CHAR()`

## json\_max\_array\_idx

**json\_max\_array\_idx** 는 JSON 함수가 배열의 크기를 변경할 때 크기의 제한을 설정하는데 사용되는 파라미터이다.

만약 매우 큰 인덱스 값을 **JSON\_ARRAY\_INSERT** 와 같은 함수에 잘못 입력한다면, JSON 문서는 많은 메모리와 디스크 공간을 차지할 것이다. 이 파라미터는 이러한 위험을 제한하기 위해 사용한다.

문자열로부터 생성된 배열에는 이 제한이 적용되지 않는다.

## max\_query\_per\_tran

**max\_query\_per\_tran** 은 트랜잭션 당 최대 보유할 수 있는 질의(close되지 않은 질의) 개수를 설정시 사용되는 파라미터이다.

기본 설정 값은 100이며, 최소값은 1, 그리고 최대값은 32,767이다. 이 값을 너무 크게 설정하면, 사용하는 메모리가 증가하여 전체 시스템 성능에 영향을 줄 수 있으므로 주의해야 한다.

## no\_backslash\_escapes

**no\_backslash\_escapes** 은 이스케이프 문자로 백슬래시(\) 사용 여부에 관한 파라미터로서, 기본값은 **yes** 이다. 이 파라미터 값이 **no** 이면 백슬래시(\)가 이스케이프 문자로 사용되며, **yes** 이면 백슬래시는 일반 문자로 사용된다. 예를 들어, 이 값이 **no** 일 때 “\n”은 개행(new line) 문자를 의미한다. 그러나 이 값이 **yes** 이면 “\n”은 “\”과 “n” 두 개의 문자를 의미한다. 백슬래시가 이스케이프 문자로 사용되는 경우에 대한 자세한 설명은 **특수 문자 이스케이프** 를 참고한다.

## only\_full\_group\_by

**only\_full\_group\_by**는 **GROUP BY** 절 사용에 관한 확장된 문법의 사용 여부를 설정하는 파라미터이다.

이 파라미터 값이 no이면 확장된 문법이 적용되므로 **GROUP BY** 절에 명시되지 않은 칼럼을 **SELECT** 칼럼 리스트에 명시할 수 있고, 이 값이 yes이면 **GROUP BY** 절에 명시된 칼럼만 **SELECT** 칼럼 리스트에 명시할 수 있다.

기본값은 **no** 이므로, SQL 표준에 따라 질의를 수행하려면 **only\_full\_group\_by** 파라미터 값을 yes로 설정한다. 이 경우에는 확장된 문법이 적용되지 않으므로 실행 결과로 아래와 같은 에러가 출력된다.

```
ERROR: Attributes exposed in aggregate queries must also
appear in the group by clause.
```

## oracle\_compat\_number\_behavior

**oracle\_compat\_number\_behavior**는 다른 DBMS(Database Management System)와의 호환성 향상을 위한 파라미터로, NUMERIC, DOUBLE 과 FLOAT 타입에 대해 소숫점 이하 0을 출력하지 않도록 하며, DOUBLE과 FLOAT인 경우 지수 형식으로 표시하지 않는다. 예를 들면, 이 파라미터 설정 값이 **no**인 경우, DOUBLE 타입으로 구성된 a\_double 테이블을 조회하는 쿼리의 결과는 아래와 같이 지수 형태로 표시되지만, 파라미터 설정 값이 **yes**인 경우 소수점으로만 표시된다.

```
csql> ;get oracle_compat_number_behavior

=== Get Param Input ===
```



```
oracle_compat_number_behavior=n
```

```
SELECT a FROM a_double;
```

```

 a
=====
1.234567890123457e+19
1.234567890123457e-08
1.2300000000000000e-08
1.0000000000000000e+00
1.2000000000000000e+00
-4.9390303000000000e-03
-1.293934894993939e+18
-1.938943893939394e+16

```

```
csql> ;get oracle_compat_number_behavior
```

```
=== Get Param Input ===
```

```
oracle_compat_number_behavior=y
```

```
SELECT a FROM a_double;
```

```

 a
=====
12345678901234570000
0.000000001234567890123457
0.00000000123
1
1.2
-0.0049390303
-1293934894993939000
-19389438939393940

```

또한, **oracle\_compat\_number\_behavior**이 **yes**로 설정된 경우, 정수 타입의 나누기 연산 결과가 정수 타입이 아닌 실수 타입 (즉, NUMERIC 타입)이 된다. 아래의 예는 이 파라미터가 **yes**로 설정된 경우로, 적용되는 정수 타입은 INT, SHORT, BIGINT 이다.

```
csql> ;get oracle_compat_number_behavior
```

```
=== Get Param Input ===
```

```
oracle_compat_number_behavior=y
```

```
SELECT 1/2;
```

```
 1/2
=====
 0.5
```

#### 참고

oracle\_compat\_number\_behavior 설정값이 적용되는 JDBC/CCI 함수는 다음과 같다.

#### JDBC

**getString(int columnIndex), getString(String columnLabel)**

· 적용 데이터 타입: NUMERIC, DOUBLE, FLOAT

**getObject(int columnIndex), getObject(String columnLabel)**

· 적용 데이터 타입: NUMERIC

#### CCI

**cci\_get\_data(CCI\_A\_TYPE\_STR 타입을 사용한 경우), 예: cci\_get\_data(req, i, CCI\_A\_TYPE\_STR, &data, &ind)**

· 적용 데이터 타입: NUMERIC, DOUBLE, FLOAT

### oracle\_style\_empty\_string

**oracle\_style\_empty\_string** 은 다른 DBMS(Database Management System)와의 호환성을 향상시키기 위한 파라미터로, 빈 문자열(empty string)과 **NULL** 을 같은 값으로 처리하도록 지정한다. 기본값은 **no** 이다. **oracle\_style\_empty\_string** 파라미터를 **no** 로 설정하면 빈 문자열을 유효한 문자열로 처리하고, **yes** 로 설정하면 함수에 따라 빈 문자열을 **NULL** 로 처리하거나, **NULL** 을 빈 문자열로 처리한다.

#### 참고

이하에 언급된 함수들을 제외한 나머지 함수들은 **oracle\_style\_empty\_string** 파라미터의 영향을 받지 않는다.

· **oracle\_style\_empty\_string=yes** 일 때 빈 문자열과 NULL을 동일하게 NULL로 처리하는 함수

◦ ASCII()

- `CONCAT_WS()`
- `ELT()`
- `FIELD()`
- `FIND_IN_SET()`
- `FROM_BASE64()`
- `INSERT()`
- `INSTR()`
- `LOWER()`
- `LEFT()`
- `LOCATE()`
- `LPAD()`
- `LTRIM()`
- `MID()`
- `POSITION()`
- `REPEAT()`
- `REVERSE()`
- `RIGHT()`
- `RPAD()`
- `RTRIM()`
- `SPACE()`
- `STRCMP()`
- `SUBSTR()`
- `SUBSTRING()`
- `SUBSTRING_INDEX()`
- `TO_BASE64()`

- `TRANSLATE()`

- `TRIM()`

- `UPPER()`

- **oracle\_style\_empty\_string=yes** 일 때 빈 문자열과 NULL을 동일하게 빈 문자열로 처리하는 함수

- `CONCAT()`

- `REPLACE()`

### 참고

`REPLACE()` 함수는 10.0 미만에서 **oracle\_style\_empty\_string=yes** 일 때의 동작이 다르다.

```
SELECT REPLACE ('abc', 'a', '');
```

위의 질의에 대해 10.0 이상 버전에서는 빈 문자열 입력을 빈 문자열로 처리하여 'bc'를 출력하지만, 10.0 미만 버전에서는 빈 문자열 입력을 NULL로 처리하여 NULL을 출력한다.

## pipes\_as\_concat

**pipes\_as\_concat** 은 이중 파이프 기호(`||`)의 사용에 관한 파라미터로서, 기본값은 **yes** 이다. 이 파라미터 값이 **yes**이면 이중 파이프 기호가 문자열의 병합 연산자로 해석되고, **no** 이면 불리언(boolean) 연산자인 **OR** 로 해석된다.

## plus\_as\_concat

**plus\_as\_concat** 은 **+** 연산자의 사용에 관한 파라미터로서, 기본값은 **yes** 이다. 이 파라미터 값이 **yes** 이면 **+** 연산자가 문자열의 병합 연산자로 해석되고, **no** 이면 수치 연산자로 해석된다.

```
-- plus_as_concat = yes
SELECT '1'+'1';
```

```
'1'+'1'
=====
'11'
```

```
SELECT '1'+'a';
```

```
'1'+'a'
=====
'1a'
```

```
-- plus_as_concat = no
SELECT '1'+'1';
```

```
'1'+'1'
=====
2.0000000000000000e+000
```

```
SELECT '1'+'a';
```

```
ERROR: Cannot coerce 'a' to type double.
```

### require\_like\_escape\_character

**require\_like\_escape\_character** 는 **LIKE** 절의 이스케이프 문자 사용 여부에 관한 파라미터로서, 기본값은 **no** 이다. 이 파라미터 값이 **yes** 이고 **no\_backslash\_escapes** 가 **no** 이면 **LIKE** 절의 문자열에서 백슬래시(\)가 이스케이프 문자로 사용되며, 그렇지 않으면 **LIKE... ESCAPE** 절을 사용하여 이스케이프 문자를 명시해야 한다. 자세한 내용은 **LIKE** 을 참고한다.

### return\_null\_on\_function\_errors

**return\_null\_on\_function\_errors** 는 일부 SQL 함수에서 에러가 발생할 때의 동작을 정의하는 파라미터로서, 기본값은 **no** 이다. 이 파라미터 값이 **yes** 이면 함수에서 에러가 발생할 때 **NULL**을 반환하며, **no** 이면 함수에서 에러가 발생할 때 에러를 반환하고 관련 메시지를 출력한다.

다음 SQL 함수가 이 시스템 파라미터의 영향을 받는다.

## 날짜/시간 함수

- `ADDDATE()`
- `ADDTIME()`
- `DATEDIFF()`
- `DAY()`
- `DAYOFMONTH()`
- `DAYOFWEEK()`
- `DAYOFYEAR()`
- `FROM_DAYS()`
- `FROM_UNIXTIME()`
- `HOUR()`
- `LAST_DAY()`
- `MAKEDATE()`
- `MAKETIME()`
- `MINUTE()`
- `MONTH()`
- `QUARTER()`
- `SEC_TO_TIME()`
- `SECOND()`
- `TIME()`
- `TIME_TO_SEC()`
- `TIMEDIFF()`
- `TO_DAYS()`
- `WEEK()`
- `WEEKDAY()`
- `YEAR()`

## 문자열 함수

- `ASCII()`
- `BIN()`
- `BIT_LENGTH()`
- `CHR()`

## 수치 함수

- `ABS()`
- `ACOS()`
- `ASIN()`
- `ATAN()`
- `ATAN2()`
- `CEIL()`
- `CONV()`
- `COS()`
- `COT()`
- `DEGREES()`
- `EXP()`
- `FLOOR()`
- `LN()`
- `LOG2()`
- `LOG10()`
- `MOD()`
- `POW()`
- `RADIANS()`
- `SIGN()`
- `SIN()`

- `SQRT()`
- `TAN()`
- `TRUNC()`
- `WIDTH_BUCKET()`

```
SET SYSTEM PARAMETERS 'return_null_on_function_errors=no';
SELECT YEAR('12:34:56');
```

```
ERROR: Conversion error in time format.
```

```
SET SYSTEM PARAMETERS 'return_null_on_function_errors=yes';
SELECT YEAR('12:34:56');
```

```
year('12:34:56')
=====
NULL
```

### stored\_procedure\_return\_numeric\_size

**stored\_procedure\_return\_numeric\_size** 는 저장 프로시저에서 NUMERIC타입으로 리턴 되는 값의 정밀도와 스케일을 설정하는 파라미터이다. 설정 방식은 양의 정수 리스트로, 정밀도와 스케일을 각각 지정할 수 있다. 예를 들면, 정밀도를 15, 스케일을 10으로 설정하는 경우 15,10 으로 지정하면 된다. 기본 값은 38,15 이며, 값의 지정 범위는 정밀도의 경우 1 ~ 38, 스케일은 0 ~ 38이다. 그리고 정밀도 보다 큰 스케일 값은 지정할 수 없다.

### string\_max\_size\_bytes

**string\_max\_size\_bytes** 는 문자열 함수 또는 연산에서 문자열 인자로 사용할 수 있는 최대 바이트 크기를 정의하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 **1,048,576** 바이트(1M)이다. 최소값은 64 바이트, 최대값은 33,554,432 바이트(32M)이다. 최대값을 넘어서는 경우, REPEAT, SPACE 함수에서는 NULL이 출력되며, 그 외 함수에서는 에러를 출력한다.

이 파라미터에 영향을 받는 함수 및 연산식은 다음과 같다.

- `SPACE()`
- `CONCAT()`
- `CONCAT_WS()`



- **+**: 문자열이 피연산자
- **REPEAT()**
- **GROUP\_CONCAT()**: 이 함수는 **string\_max\_size\_bytes** 파라미터뿐만 아니라 **group\_concat\_max\_len** 파라미터의 영향도 받는다.
- **INSERT()** 함수

## unicode\_input\_normalization

시스템 수준에서 입력할 유니코드를 결합된 코드로 저장할지 여부를 설정하는 파라미터로 기본값은 **no** 이다.

일반적으로 유니코드 텍스트는 “완전히 결합된(fully composed)” 혹은 “완전히 분해된(fully decomposed)” 상태로 저장될 수 있다. 예를 들면 “완전히 결합된” 문자 ‘Ä’는 하나의 코드포인트인 00C4를 갖는데, 이는 UTF-8 인코딩으로 C3 84의 2바이트가 된다. 이와 달리 “완전히 분해된” 모드에서는 두 개의 코드포인트/문자 0041(문자 “A”)과 0308(COMBINING DIAERESIS)이 되며, UTF-8 인코딩으로는 41 CC 88의 3바이트가 된다.

CUBRID는 완전히 결합된 유니코드만 가지고 동작할 수 있다. 완전히 분해된 문자열을 가진 응용 클라이언트에 대해서는 **unicode\_input\_normalization**의 값을 **yes**로 설정하여 완전히 결합된 코드로 변환하여 동작한 후 **unicode\_output\_normalization**의 값을 **yes**로 설정하여 다시 완전히 분해된 텍스트로 되돌려줄 수 있다. 일반적으로 유니코드 정규화는 결합 이후 분해 시 역변환이 불가능하지만 CUBRID는 가능한 한 많은 문자들의 역변환을 가능하게 하기 위해 간략화한 유니코드 정규화 방식을 적용한다. CUBRID 정규화의 특징은 다음과 같다.

- 정규화는 언어 특징적인 요소가 아니며 로캘에 의존하지 않는다.

일단 하나 이상의 로캘을 사용할 수 있다면, 모든 CAS/CSQL 프로세스에서 이를 사용할 수 있다. 하지만 정규화는 CUBRID 서버 프로세스에 적용되는 것이 아니다.

**unicode\_input\_normalization** 시스템 파라미터는 시스템 수준에서 정규화에 의한 입력 코드의 결합 여부를 지정한다. **unicode\_output\_normalization** 시스템 파라미터는 시스템 수준에서 정규화에 의한 출력 코드의 분해 여부를 지정한다.

- 콜레이션과 정규화는 직접 관련이 없다.

**unicode\_input\_normalization** 값이 **no**임에도 불구하고, 확장이 있는 콜레이션(utf8\_de\_exp, utf8\_jap\_exp, utf8\_km\_exp) 문자열이 완전히 분해된 상태에서 제대로 정렬되지만 이는 의도한 사항이 아니며 UCA(Unicode Collation Algorithm)의 우연한 효과(side-effect)일 뿐이다. 확장이 있는 콜레이션은 오직 완전히 결합된 텍스트만 가지고 동작하도록 구현되었다.

- CUBRID에서 유니코드 정규화를 위한 결합(composition)과 분해(decomposition)는 별개로 동작하지 않는다.

일반적으로 **unicode\_input\_normalization** 와 **unicode\_output\_normalization** 의 값을 yes로 사용한다. 이 경우 클라이언트로부터 입력된 코드는 결합 모드로 저장되고 출력은 분해 모드로 저장된다.

클라이언트 응용 프로그램이 텍스트 데이터를 분해된 형태로 CUBRID에 보낸다면, **unicode\_input\_normalization** 을 yes 로 설정하여 CUBRID가 결합된 코드로 다루게 한다.

클라이언트 응용 프로그램이 분해된 형태로만 데이터를 다룰 수 있다면 **unicode\_output\_normalization** 을 yes 로 설정하여 CUBRID가 항상 분해된 코드로 데이터를 보내도록 한다.

클라이언트 응용 프로그램이 입력과 출력의 형태에 대해 모두 알고 있다면, **unicode\_input\_normalization** 과 **unicode\_output\_normalization** 을 no 인 상태로 둔다(기본 설정).

로캘과 콜레이션에 대한 자세한 설명은 [다국어 개요](#) 을 참고한다.

### **unicode\_output\_normalization**

시스템 수준에서 저장된 유니코드를 분해된 코드로 출력할 것인지 여부를 설정하는 파라미터로 기본값은 no 이다. 보다 자세한 설명은 위의 **unicode\_input\_normalization** 설명을 참고한다.

### **update\_use\_attribute\_references**

**update\_use\_attribute\_references** 는 UPDATE 문에서 갱신 대상 칼럼을 명시할 때 먼저 명시한 칼럼의 값이 질의문에서 해당 칼럼을 사용하는 또 다른 칼럼의 갱신에 영향을 줄 것인지 여부를 설정하는 파라미터로서, 기본값은 no 이다.

아래의 UPDATE 문 결과는 **update\_use\_attribute\_references** 파라미터의 값에 따라 달라진다.

```
CREATE TABLE tbl(a INT, b INT);
INSERT INTO tbl values (10, NULL);

UPDATE tbl SET a=1, b=a;
```

이 파라미터의 값이 yes이면, 위의 UPDATE 질의에서 갱신되는 b의 값은 “a=1”의 영향을 받아 1이 된다.

```
SELECT * FROM tbl;
```

```
1, 1
```

이 파라미터의 값이 no이면, 위의 UPDATE 질의에서 갱신되는 b의 값은 “a=1”의 영향을 받지 않고 해당 레코드에 저장되어 있는 a 값의 영향을 받아 NULL이 된다.

```
SELECT * FROM tbl;
```

```
1, NULL
```

## 스레드 관련 파라미터

스레드 파라미터 설정을 통해 스레드를 관리할 수 있다. 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                           | 타입   | 기본값      | 최소값 | 최대값  |
|-----------------------------------|------|----------|-----|------|
| thread_connection_pooling         | bool | true     |     |      |
| thread_connection_timeout_seconds | int  | 300      | -1  | 3600 |
| thread_worker_pooling             | bool | true     |     |      |
| thread_core_count                 | int  | 시스템 코어 수 | 1   | 1024 |
| thread_worker_timeout_seconds     | int  | 300      | -1  | 3600 |
| loaddb_worker_count               | bool | 8        | 2   | 64   |

## thread\_connection\_pooling

**thread\_connection\_pooling** 파라미터가 true이면, 서버 부팅 시 클라이언트 연결을 관리하는 모든 스레드가 풀링된다.

#### **thread\_connection\_timeout\_seconds**

**thread\_connection\_timeout\_seconds**는 연결을 관리하는 스레드가 종료하기 전에 대기하는 시간을 설정하는 파라미터이다. 스레드는 연결이 종료된 후 새로운 연결을 할당받기 위해 파라미터 설정값(초 단위) 동안 대기한다. 대기 시간을 초과할 때까지 새로운 연결을 할당받지 못하면, 스레드는 종료하게 된다. 이후 새로운 연결 요청이 발생하게 되면 다른 스레드가 시작될 것이다. 파라미터 값이 -1로 설정된 경우 스레드는 종료하지 않고, 새로운 연결이 할당될 때까지 정지한다.

#### **thread\_worker\_pooling**

**thread\_worker\_pooling** 파라미터가 true이면, 서버 부팅 시 클라이언트가 요청한 작업을 실행하는 모든 스레드가 풀링된다.

#### **thread\_core\_count**

**thread\_core\_count** 파라미터에 따라 풀링되는 스레드의 그룹 개수를 조정한다. 기본 값은 시스템 코어 수로 설정된다. 만약 설정된 파라미터 값에 의해 한 그룹의 스레드가 3개 이상 확보되지 않으면, 최소한 3개의 스레드가 각 그룹에 속하도록 시스템이 강제로 이 값을 조정한다.

#### **thread\_worker\_timeout\_seconds**

**thread\_worker\_timeout\_seconds**는 클라이언트가 요청한 작업을 처리하는 스레드가 종료하기 전에 대기하는 시간을 설정하는 파라미터이다. 스레드는 클라이언트가 요청한 작업을 실행한 후 새로운 작업을 할당받기 위해 파라미터 설정값(초 단위) 동안 대기한다. 대기 시간을 초과할 때까지 새로운 클라이언트 요청 작업을 할당받지 못하면, 스레드는 종료하게 된다. 이후 새로운 클라이언트 요청 작업이 발생하게 되면 다른 스레드가 시작될 것이다. 파라미터 값이 -1로 설정된 경우 스레드는 종료하지 않고, 새로운 작업이 할당될 때까지 정지한다.

#### **loaddb\_worker\_count**

**loaddb\_worker\_count**는 **loaddb** 세션에 전용으로 할당되는 최대 스레드 개수를 설정하는 파라미터이다. 단일 **loaddb** 세션이 수행되는 경우 모든 스레드가 이 작업을 수행할 수도 있다. 여러 **loaddb** 세션이 동시에 수행되는 경우 모든 세션에 할당된 스레드의 총 수는 이 파라미터 설정값을 초과할 수 없다.

#### 타임존 파라미터

다음은 타임존과 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                | 타입     | 기본값             | 최소값 | 최대값 |
|------------------------|--------|-----------------|-----|-----|
| server_timezone        | string | OS의 타임존         |     |     |
| timezone               | string | server_timezone |     |     |
| tz_leap_second_support | bool   | no              |     |     |

#### · **timezone**

세션에 대한 타임존을 설정한다. 기본값은 **server\_timezone**의 값이다. 타임존 오프셋(예: +01:00, +02) 또는 타임존 지역 이름(예: Asia/Seoul)으로 설정 가능하며, 데이터베이스 운영 중 변경이 가능하다.

#### · **server\_timezone**

서버에 대한 타임존을 설정한다. 기본값은 OS의 타임존이다. 변경된 값을 반영하려면 데이터베이스를 재시작해야 한다. 운영 체제의 타임존은 운영 체제와 운영 체제 설정 파일에 있는 정보에 따라 다를 수 있다.

- Windows의 경우, tzset() 함수와 tzname[0] 변수를 사용해 Windows 스타일 타임존 명을 검색한다. 이 이름은 CUBRID 매핑 데이터를 사용하는 IANA/CUBRID 스타일명으로 변환된다(매핑 파일은 %CUBRID%\timezones\tzdata\windowsZones.xml임).
- Linux의 경우, CUBRID에서 "/etc/sysconfig/clock" 파일을 읽고 구문을 분석하려 시도한다. 이 파일이 없는 경우 "/etc/localtime"의 값을 읽고 사용한다.
- AIX의 경우, "TZ" 운영 체제 환경 변수의 값을 사용한다.

모든 운영 체제에서 server\_timezone이 지정되지 않은 경우 "Asia/Seoul" 타임존을 서버 타임존으로 사용한다.

#### · **tz\_leap\_second\_support**

윤초(leap second)에 대한 지원 여부를 yes 또는 no로 설정한다. 기본값은 **no**이다.

윤초(閏秒)란 태양시 평균에 근접한 하루의 시간을 유지하기 위해 협정 세계시(UTC)에서 발생하는 오차를 보정하기 위해 가끔 추가하는 1초이다.

변경된 값을 반영하려면 데이터베이스를 재시작해야 한다.

## 질의 계획 캐시 관련 파라미터

다음은 질의 계획에 대한 캐시 기능과 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                              | 타입  | 기본값   | 최소값 | 최대값 |
|--------------------------------------|-----|-------|-----|-----|
| max_plan_cache_entries               | int | 1,000 |     |     |
| max_plan_cache_clones                | int | 1,000 |     |     |
| xasl_cache_time_threshold_in_minutes | int | 360   |     |     |
| max_filter_pred_cache_entries        | int | 1,000 |     |     |

### max\_plan\_cache\_entries

**max\_plan\_cache\_entries** 는 메모리에 캐시하는 질의 실행 계획의 최대 개수를 설정하는 파라미터이다. **max\_plan\_cache\_entries** 파라미터가 -1이나 0으로 설정되면 작성된 질의 실행 계획을 메모리 캐시에 저장하지 않는 것이며, 1 이상의 정수값이 설정되면 설정된 개수만큼의 질의 실행 계획을 메모리 캐시한다.

다음은 최대 1,000개의 질의에 대해 질의 실행 계획 캐시 기능을 수행하는 예제이다.

```
max_plan_cache_entries=1000
```

### max\_plan\_cache\_clones

플랜 캐시(plan cache)에는 질의에 대한 직렬화(serializable)된 xasl 저장되어 있는데, 질의 수행기는 플랜 캐시에 저장된 직렬화된 xasl를 역직렬화(deserializable)한 후 사용값과 함께 수행한다. 클론 캐시(clone cache)는 질의 수행기가 최소한의 역직렬화 수행하기 위해 역직렬화된 xasl을 저장하여 재 사용하기 위한 캐시 파라미터이다. 기본값은 1000이며, 하나의 플랜이 최대 가질 수 있는 캐시 값입니다. 플랜이 삭제되면 해당 플랜의 클론 캐시에 저장된 역직렬화된 xasl도 함께 삭제된다.

### xasl\_cache\_time\_threshold\_in\_minutes

캐시된 플랜을 재 수행(clean-up)할 것인지를 판단하는 파라미터로, 해당 파라미터는 플랜 캐시에 추가할 공간이 없는 경우도 제거할 대상을 찾기 위해 사용된다. 분단위로 설정 가능하며, 기본값은 360분이다.

### max\_filter\_pred\_cache\_entries

**max\_filter\_pred\_cache\_entries** 는 메모리에 캐시하는 필터링된 인덱스 표현식의 최대 개수를 설정하는 파라미터이다. 필터링된 인덱스 표현식은 컴파일된 상태로 저장되므로, 서버에서 즉시 사용할 수 있다. 캐시에 저장되어 있지 않을 경우, 필터링된 인덱스 표현식을 데이터베이스 스키마에서 가져와서 해석하는 과정이 필요하다.

### 질의 캐시 관련 파라미터

다음은 질의 결과에 대한 캐시 기능과 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                   | 타입  | 기본값 | 최소값 | 최대값     |
|---------------------------|-----|-----|-----|---------|
| max_query_cache_entries   | int | 0   | 0   | INT_MAX |
| query_cache_size_in_pages | int | 0   | 0   | INT_MAX |

두 파라미터 값 중 하나가 0인 경우, **QUERY\_CACHE** 힌트와 상관 없이 질의 결과는 캐시되지 않는다.

### max\_query\_cache\_entries

**max\_query\_cache\_entries** 는 최대 캐시할 수 있는 질의 개수에 대한 설정 값이다. 이 파라미터 값이 10이상으로 설정되면 설정된 수 만큼의 질의가 캐시된다.

다음 예는 최대 500개의 질의를 캐시하는 것을 보여주고 있다.

```
max_query_cache_entries=500
```

### query\_cache\_size\_in\_pages

**query\_cache\_size\_in\_pages** 는 최대 캐시할 수 있는 결과 페이지에 대한 설정 값이다. 이 파라미터 값이 10이상으로 설정되면 설정된 페이지 만큼의 결과가 캐시된다.

다음 예는 최대 4000 페이지의 결과를 캐시하는 것을 보여주고 있다.

```
query_cache_size_in_pages=4000
```

## 유틸리티 관련 파라미터

다음은 CUBRID에서 사용되는 유틸리티와 관련된 파라미터로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                      | 타입   | 기본값 | 최소값 | 최대값 |
|------------------------------|------|-----|-----|-----|
| backup_volume_max_size_bytes | byte | 0   | 32K |     |
| communication_histogram      | bool | no  |     |     |
| compactdb_page_reclaim_only  | int  | 0   |     |     |
| csql_history_num             | int  | 50  | 1   | 200 |

### backup\_volume\_max\_size\_bytes

**backup\_volume\_max\_size\_bytes** 는 **cubrid backupdb** 유틸리티에 의해 생성되는 백업 볼륨 파일의 분할 크기를 바이트 단위로 설정하는 파라미터이다. 값 뒤에 B, K, M, G, T로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes를 의미한다. 단위를 생략하면 바이트 단위가 적용된다. 기본값은 0이고, 최소값은 32K이다.

기본값인 **0** 으로 설정하면 생성되는 백업 볼륨이 분할되지 않으며, 0보다 큰 값을 설정하면 지정된 크기의 단위로 백업 볼륨 파일을 분할하여 생성한다.

### communication\_histogram

**communication\_histogram** 은 csql 인터프리터의 **세션 명령어** “**;.h**”와 관련된 파라미터이며, 기본값은 **no** 이다. **CSQL 실행 통계 정보 출력** 를 참고한다.

### compactdb\_page\_reclaim\_only

**compactdb\_page\_reclaim\_only** 는 **compactdb** 유틸리티와 관련된 파라미터로 이미 할당된 저장 영역의 OID를 재사용하기 위하여 이미 삭제된 객체의 저장 영역을 정리하는



유틸리티이다. **compactdb** 유틸리티에 의해 저장 영역이 재정렬되는 작업은 3단계로 구분할 수 있으며, **compactdb\_page\_reclaim\_only** 파라미터를 통해 재정렬 작업의 단위를 선택할 수 있다. 기본값인 **0** 으로 설정하면 1, 2, 3단계를 모두 수행하므로 데이터 단위, 테이블 단위, 파일 단위로 저장 영역을 재정렬한다. 1로 설정하면 1단계를 생략하므로 테이블 및 파일 단위로 저장 영역을 재정렬하고, 2로 설정하면 1, 2단계를 생략하므로 파일 단위로만 저장 영역을 재정렬한다.

- 1단계 : 데이터 단위로 저장 영역을 재정렬한다.
- 2단계 : 테이블 단위로 저장 영역을 재정렬한다.
- 3단계 : 파일 단위(heap file)로 저장 영역을 재정렬한다.

### csql\_history\_num

**csql\_history\_num** 은 CSQL 인터프리터와 관련된 파라미터로 CSQL 인터프리터 내에서 히스토리 내역으로 저장되는 SQL 문의 개수를 설정하는 파라미터이다. 기본값은 **50** 이다.

### HA 관련 파라미터

다음은 HA 관련 파라미터로, 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름 | 타입 | 기본값 |
|---------|----|-----|
|---------|----|-----|

|         |        |     |
|---------|--------|-----|
| ha_mode | string | off |
|---------|--------|-----|

### ha\_mode

**ha\_mode** 는 CUBRID HA 기능을 설정하기 위한 파라미터이며, 기본값은 **off** 이다.

- off : CUBRID HA 기능을 사용하지 않는다.
- on : 설정한 노드는 failover의 대상이 되는 노드로, CUBRID HA 기능을 사용한다.
- replica : 설정한 노드는 failover의 대상이 되지 않는 노드로, CUBRID HA 기능을 사용한다.

CUBRID HA 기능을 사용하려면 **ha\_mode** 파라미터를 설정하는 것 외에 **cubrid\_ha.conf** 파일에서 HA 관련 파라미터를 설정해야 한다. 자세한 내용은 **CUBRID HA** 를 참고한다.

## 기타 파라미터

다음은 기타 파라미터 정보로 각 파라미터의 타입과 설정 가능한 값의 범위는 다음과 같다.

| 파라미터 이름                   | 타입     | 기본값  | 최소값 | 최대값   |
|---------------------------|--------|------|-----|-------|
| access_ip_control         | bool   | no   |     |       |
| access_ip_control_file    | string |      |     |       |
| agg_hash_respect_order    | bool   | yes  |     |       |
| auto_restart_server       | bool   | yes  |     |       |
| enable_string_compression | bool   | yes  |     |       |
| index_scan_in_order       | bool   | no   |     |       |
| index_unfill_factor       | float  | 0.05 | 0   | 0.5   |
| stored_procedure          | bool   | yes  |     |       |
| stored_procedure_uds      | bool   | yes  |     |       |
| stored_procedure_port     | int    | 0    | 0   | 65535 |

| 파라미터 이름                        | 타입     | 기본값              | 최소값          | 최대값                   |
|--------------------------------|--------|------------------|--------------|-----------------------|
| stored_procedure_vm_options    | string |                  |              |                       |
| multi_range_optimization_limit | int    | 100              | 0            | 10,000                |
| optimizer_enable_merge_join    | bool   | no               |              |                       |
| use_stat_estimation            | bool   | no               |              |                       |
| server                         | string |                  |              |                       |
| service                        | string |                  |              |                       |
| session_state_timeout          | sec    | 21,600 (6 hours) | 60(1 minute) | 31,536,000 (1 year)   |
| sort_limit_max_count           | int    | 1000             | 0            | INT_MAX               |
| sql_trace_slow                 | msec   | -1(inf)          | 0            | 86,400,000 (24 hours) |
| sql_trace_execution_plan       | bool   | no               |              |                       |
| use_orderby_sort_limit         | bool   | yes              |              |                       |
|                                | int    | 1                | 0            | 1                     |

| 파라미터 이름                             | 타입     | 기본값     | 최소값 | 최대값     |
|-------------------------------------|--------|---------|-----|---------|
| vacuum_prefetch_log_mode            |        |         |     |         |
| vacuum_prefetch_log_buffer_size     | int    | 50M     | 25M | INT_MAX |
| data_buffer_neighbor_flush_pages    | int    | 8       | 0   | 32      |
| data_buffer_neighbor_flush_nondirty | bool   | no      |     |         |
| tde_keys_file_path                  | string |         |     |         |
| tde_default_algorithm               | string | AES     |     |         |
| recovery_progress_logging_interval  | int    | 0 (off) | 0   | 3600    |
| supplemental_logging                | int    | 0 (off) | 0   | 2       |
| regexp_engine                       | string | re2     |     |         |
| vacuum_ovfp_checkpoint_threshold    | int    | 1000    | 2   | INT_MAX |

| 파라미터 이름                         | 타입   | 기본값   | 최소값 | 최대값    |
|---------------------------------|------|-------|-----|--------|
| vacuum_ovfp_checkpoint_duration | int  | 45000 | 1   | 600000 |
| deduplicate_key_level           | int  | -1    | -1  | 14     |
| print_index_detail              | bool | no    |     |        |
| enable_memory_monitoring        | bool | no    |     |        |
| flashback_timeout               | int  | 300   | 0   | 3600   |

### access\_ip\_control

**access\_ip\_control** 은 서버 접속을 허용하는 IP를 제한하는 기능 사용 여부를 지정하는 파라미터이다. 기본값은 **no** 이다. 자세한 내용은 [데이터베이스 서버 접속 제한](#) 을 참고한다.

### access\_ip\_control\_file

**access\_ip\_control\_file** 은 서버가 허용하는 IP 목록을 저장한 파일 이름을 지정하는 파라미터이다. **access\_ip\_control** 값이 yes이면 데이터베이스 서버는 이 파라미터로 지정한 파일에 저장된 IP의 접속만 허용한다. 자세한 내용은 [데이터베이스 서버 접속 제한](#) 을 참고한다.

### agg\_hash\_respect\_order

**agg\_hash\_respect\_order** 는 집계 함수에서 그룹이 순서대로 반환되는지 여부를 설정하는 파라미터이다. 기본값은 **yes** 이다. [max\\_agg\\_hash\\_size](#) 를 참고한다.

이 모든 그룹(키와 누적 결과)이 해시 메모리에 상주할 수 있으면, “agg\_hash\_respect\_order=no” 설정은 결과를 출력하기 전에 정렬하는 과정을 생략할 것이므로, 순서가 보장되지 않을 것이라고 예측할 수 있다. 그러나, 오버플로우가 발생하면 정렬 과정이 수행되어야 하며 “agg\_hash\_respect\_order=false”로 설정되었더라도 정렬된 결과를 얻게 된다.

## auto\_restart\_server

**auto\_restart\_server** 는 데이터베이스 서버에 심각한 오류가 발생해서 서버 프로세스(cub\_server)가 비정상 종료된 경우 자동으로 재시작할 것인가를 지정하는 파라미터로, 기본값은 **yes** 이다.

이 파라미터를 **yes** 로 설정하면, 마스터 프로세스(cub\_master)가 비정상 종료된 서버 프로세스를 감지하여 자동으로 재시작 시킨다. 그러나 해당 기능에 의해 자동 재시작된 서버 프로세스가 120초 이내에 다시 비정상 종료될 경우에는 반복적인 서버 프로세스 비정상 종료에 따른 문제를 방지하기 위해 더이상 서버 프로세스를 재시작 시키지 않는다. 또한, 정상적인 종료 절차(**cubrid server stop** 명령)에 의해 종료된 서버 프로세스는 자동으로 재시작 시키지 않는다.

이 파라미터는 리눅스 환경에서만 동작하며, HA로 구성되지 않은 서버 프로세스에만 영향을 준다. 또한, 마스터 프로세스가 구동되지 않은 상태에서는 이 기능이 정상적으로 동작하지 않는다.

## enable\_string\_compression

**enable\_string\_compression** 은 문자열 타입 값을 힙, 인덱스 또는 리스트에 저장할 때 문자열 압축 사용 여부를 설정하는 파라미터이다. **enable\_string\_compression** 값을 **yes** 로 설정하면 문자열 크기가 255바이트 이상이고, 압축된 문자열이 원래 문자열의 크기보다 작을 경우 압축된 문자열로 저장된다.

## index\_scan\_in\_oid\_order

**index\_scan\_in\_oid\_order** 는 인덱스를 스캔한 후 검색 결과 데이터를 가져오는 순서를 OID 순으로 지정하기 위한 파라미터이다. 기본값인 **no** 로 설정하면 데이터 순서대로 결과를 가져오고, **yes**로 설정하면 OID 순서대로 결과를 가져온다.

## index\_unfill\_factor

최초 인덱스 생성 후 **INSERT** 나 **UPDATE** 를 실행할 때 인덱스 페이지가 꽉 차서 여유 공간이 없으면 인덱스 페이지 노드 분할(split)이 발생하는데, 이는 오퍼레이션 시간을 증가시켜 성능에 영향을 미친다. **index\_unfill\_factor** 는 인덱스를 생성할 때 각 인덱스 페이지 노드의 여유 공간을 확보하는 비율을 지정하는 파라미터이다. **index\_unfill\_factor** 설정값은 인덱스를 처음 생성할 때만 적용되며, 페이지에 지정된 빈 공간의 비율을 동적으로 유지하지 않는다. 값의 범위는 0부터 0.5까지이고 기본값은 **0.05** 이다.

인덱스를 생성할 때 인덱스의 페이지 노드에 여유 공간이 없이(**index\_unfill\_factor** 를 0으로 설정) 생성한다면, 추가로 삽입할 때마다 매번 인덱스 페이지 노드의 분할이 발생하여 성능에 영향을 끼친다.

**index\_unfill\_factor** 값이 크면 인덱스 생성 시 노드 여유 공간을 많이 확보한다. 따라서 최초 인덱스 생성 후 노드 여유 공간이 꽉 찰 때까지 상대적으로 긴 시간 동안 인덱스 노드의 분할이 발생하지 않으므로, 상대적으로 성능이 나을 수 있다.

이 값이 작으면 인덱스 생성 시 노드 여유 공간이 작기 때문에, 인덱스 노드의 여유 공간이 금방 꽉 차게 될 가능성이 높으므로, 상대적으로 **INSERT** 나 **UPDATE** 에 의한 인덱스 노드 분할 발생 가능성이 높다.

## stored\_procedure

**stored\_procedure** 는 cub\_pl 프로세스를 실행하여 저장 프로시저(stored procedure)를 사용하게 하기 위한 파라미터이다. 기본값은 **yes**이며, **no**로 설정하는 경우 cub\_pl 프로세스가 실행되지 않고, 저장 프로시저(stored procedure)를 사용할 수 없다.

## stored\_procedure\_uds

**stored\_procedure\_uds** 는 저장 프로시저를 호출하는 cub\_pl 프로세스와 cub\_server 프로세스 간에 유닉스 도메인 소켓을 통한 연결을 사용하기 위한 파라미터이다. 기본값은 **yes**이고, **no**로 설정하는 경우나 Windows의 경우 파라미터의 값과 관련 없이 TCP 소켓 연결로 동작한다.

### 참고

cub\_pl 프로세스의 유닉스 도메인 소켓 파일 경로를 지정하는 **CUBRID\_TMP** 환경 변수에 대한 내용은 [환경 변수 설정](#) 을 참고한다.

## stored\_procedure\_port

**stored\_procedure\_port** 데이터베이스 서버에서 저장 프로시저를 호출하기 위한 TCP 포트 번호를 설정하는 파라미터이다. 이 값은 65,536보다 작아야한다. 기본값은 **0** 이고 이는 임시 포트 범위에서 포트 번호가 자동으로 할당됨을 의미한다. 이 파라미터의 값은 **stored\_procedure** 파라미터가 **yes** 일 때에만 적용된다. cubrid.conf의 [common]에서 이 파라미터를 설정하면 에러가 발생하므로 주의한다.

```
.....
[common]
.....
an error occurs. remove the following line.
stored_procedure_port=4333
.....
[@testdb]
.....
the parameter is configured successfully for testdb
stored_procedure_port=4334
.....
```

## stored\_procedure\_vm\_options

**stored\_procedure\_vm\_options** 는 저장 프로시저가 실행되는 프로시저 언어 서버의 가상 머신의 설정하기 위한 파라미터이다. 각 옵션 문자열은 공백으로 구분해야한다. 기본값은 빈 문자열이다. 만약 파라미터가 [@<database>] 섹션에 설정되면, [common] 섹션에서 설정된 옵션은 해당 데이터베이스에 적용되지 않는다.

#### 참고

현재 cub\_pl 프로세스에서 자바 가상 머신 (Java Virtual Machine)을 내부적으로 사용하므로 JVM 옵션을 입력할 수 있다. JVM 옵션의 경우 표준 옵션, 비표준 옵션 그리고 고급 옵션이 있다. 비표준과 고급 옵션의 경우 모든 JVM 구현에서 지원하는 것을 보장하지 않는다.

```
.....
[common]
.....
stored_procedure_vm_options="-Xms1024m -Xmx1024m -
XX:PermSize=512m -XX:MaxPermSize=512m"
.....
[@testdb]
.....

-XX:PermSize=512m and -XX:MaxPermSize=512m 옵션은 [common]
섹션에 설정되었더라도 testdb에 적용되지 않는다.
stored_procedure_vm_options="-Xms2048m -Xmx2048m"
.....
```

### multi\_range\_optimization\_limit

**multi\_range\_optimization\_limit** 은 다중 범위(col IN (?, ?, ... ,?))의 조건을 가지며 인덱스 사용이 가능한 질의에서, **LIMIT** 절이 지정하는 행의 개수가 이 파라미터가 지정하는 숫자 이내이면 인덱스 정렬 방식에 대한 최적화를 수행한다. 기본값은 **100** 이다.

예를 들어, 이 파라미터의 값이 50일 때 LIMIT 10이면 이 파라미터가 지정한 값 이내이므로 각 조건에 해당하는 범위의 값을 정렬하면서 결과를 생성한다. LIMIT 60이면 파라미터 설정값을 초과하므로 각 조건에 해당하는 범위의 값을 모두 가져온 후 정렬한다.

이 값의 설정에 따라 중간 값의 정렬을 진행하면서(on-the-fly) 결과를 수집하느냐, 결과 값을 먼저 수집한 후 정렬하느냐의 차이가 발생하므로, 이 값이 너무 크면 오히려 성능에 불리할 수 있다.

### optimizer\_enable\_merge\_join



**optimizer\_enable\_merge\_join** 는 정렬 병합 조인(sort merge join) 계획을 질의 실행 계획의 후보에 포함할 것인지 여부를 지정하는 파라미터이다. 기본값은 **no** 이다. 정렬 병합 조인과 관련된 내용은 **SQL 힌트** 를 참고한다.

### use\_stat\_estimation

**use\_stat\_estimation** 는 통계정보 계산 시 예측 정보를 사용할 것인지 여부를 지정하는 파라미터이다. 기본값은 **no** 이다. 힙 매니저에서 DML 처리 중 생성되는 예측 정보는 추가된 객체 수와 관련 있다. 이것은 전체 객체 개수에는 상대적으로 정확하지만 고유 값 개수에는 정확하지 않다.

### server

**server** 는 CUBRID 서비스 시작 시 자동으로 시작하도록 하는 데이터베이스 서버 프로세스들의 이름을 등록하는 파라미터이다. 해당 데이터베이스들의 이름을 쉼표(,)로 구분하여 나열한다.

### service

**service** 는 CUBRID 서비스 시작 시 자동으로 시작하는 프로세스를 등록하는 파라미터로 **server**, **broker**, **manager**, **heartbeat** 의 네 종류 프로세스가 있다. 일반적으로 **service=server,broker,manager** 와 같이 세 종류 프로세스를 등록한다. 각 프로세스에 따른 동작은 다음과 같다.

- **server** : **@server** 파라미터에서 지정한 데이터베이스 프로세스를 시작한다.
- **broker** : 브로커 프로세스를 시작한다.
- **manager** : 매니저 프로세스를 시작한다.
- **heartbeat** : HA 관련 프로세스를 시작한다.

### session\_state\_timeout

**session\_state\_timeout** 은 DB 서버 프로세스 내에서 세션 데이터가 유지되는 시간을 정의하는 시스템 파라미터이다. 세션 데이터는 드라이버가 연결을 종료하거나 세션 기간이 만료될(expired) 때 삭제되며, 응용 클라이언트가 비정상 종료되면 지정한 시간 이후에 세션 기간이 만료된다.

CUBRID 세션 데이터에 해당하는 것은 **SET** 으로 정의된 사용자 변수, **PREPARE** 문, 가장 마지막에 삽입한 ID(**LAST\_INSERT\_ID**), 가장 마지막에 실행한 문장에 영향받은 레코드의 개수(**ROW\_COUNT**)이다. **SET** 으로 정의된 사용자 변수와 **PREPARE** 문은 세션 기간이 만료되기 전에 **DROP** / **DEALLOCATE** 문을 수행하여 삭제할 수 있다.

기본값은 **21,600** (6시간)이고, 단위는 초이다.

### sort\_limit\_max\_count

“ORDER BY ... LIMIT N “ 구문에 의해 top-N개의 행이 정렬될 때 적용될 수 있는 SORT-LIMIT 최적화와 관련하여, 해당 최적화를 적용하는 것을 제한하는 LIMIT 행 개수를 명시한다. N의 값이 **sort\_limit\_max\_count**의 값보다 작을 때 SORT-LIMIT 최적화가 적용된다. 기본값은 **1,000**이며, 최소값은 0(최적화를 항상 안 한다는 뜻), 최대값은 INT\_MAX이다.

좀더 자세한 사항은 **SORT-LIMIT 최적화**를 참고한다.

## sql\_trace\_slow

**sql\_trace\_slow**는 장기 실행 질의(long running query)로 판단될 질의 실행 시간을 설정하는 파라미터이다. ms, s, min, h 단위를 지정할 수 있으며 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위 생략 시 기본 단위는 밀리초(ms)이다. 기본값은 **-1**이고 최대값은 86,400,000 밀리초(24h)이다. -1은 무한대 시간을 의미하며 어떤 질의도 장기 실행 질의로 판단되지 않는다. 자세한 내용은 아래의 **sql\_trace\_execution\_plan**의 설명을 참고한다.

### 참고

**sql\_trace\_slow**는 서버의 질의 실행 시간을 기준으로 실행 시간 초과 여부를 판단하며, **MAX\_QUERY\_TIMEOUT** 브로커 파라미터는 브로커 단에서의 질의 실행 시간을 기준으로 이를 판단한다.

## sql\_trace\_execution\_plan

**sql\_trace\_execution\_plan**은 **sql\_trace\_slow** 파라미터 값의 설정 시간을 초과한 장기 실행 질의(long running query)의 실행 계획을 출력할지 여부를 설정하는 파라미터이다. 기본값은 **no**이다.

이 값이 yes이면 서버 에러 로그 파일(\$CUBRID/log/server 이하의 파일), CAS 로그 파일(\$CUBRID/log/broker/sql\_log 이하의 파일)에 해당 SQL 문, 질의 실행 계획, cubrid statdump 명령의 출력 정보를 기록한다.

이 값이 no면 서버 에러 로그 파일, CAS 로그 파일에 해당 SQL문만 기록한다.

예를 들어 5초를 초과하면 느린 질의(slow query)로 규정하고 해당 질의의 실행 계획을 로그 파일에 출력하고 싶은 경우, **sql\_trace\_slow**의 값을 5,000(ms)로 설정하고 **sql\_trace\_execution\_plan**의 값을 yes로 설정한다.

단, 서버 에러 로그 파일에는 error\_log\_level 파라미터의 값이 NOTIFICATION인 경우에만 해당 정보를 기록한다.

## use\_orderby\_sort\_limit

**use\_orderby\_sort\_limit** 은 **ORDER BY ... LIMIT** *row\_count* 절을 포함하는 구문에서 질의의 정렬 및 합병(sort and merge) 과정의 중간 결과를 *row\_count* 만큼만 유지할 것인지를 지정하는 파라미터이다. **yes** 이면 중간 정렬 결과를 *row\_count* 만큼만 유지하기 때문에 불필요한 비교 및 합병 과정을 줄일 수 있다. 기본값은 **yes** 이다.

### **vacuum\_prefetch\_log\_mode**

**vacuum\_prefetch\_log\_mode** 는 vacuum을 위하여 로그 페이지의 프리패치(prefetch) 모드를 설정하는 파라미터이다.

모드 0에서는 vacuum 마스터 스레드가 공유 버퍼에서 필요한 로그 페이지를 프리패치한다. 모드 1(기본값)에서는 각 vacuum 작업자가 자체 버퍼에서 필요한 로그 페이지를 프리패치한다. 또한 모드 0에서는 **vacuum\_prefetch\_log\_buffer\_size** 시스템 파라미터를 설정해야 하지만, 모드 1에서는 이 파라미터를 무시하고 각 vacuum 작업자가 전체 vacuum 로그 블록(기본 32개의 로그 페이지)을 프리패치한다.

### **vacuum\_prefetch\_log\_buffer\_size**

**vacuum\_prefetch\_log\_buffer\_size** 는 vacuum의 로그 프리패치 버퍼 크기를 설정하는 파라미터이다. 이 파라미터는 **vacuum\_prefetch\_log\_mode** 를 0으로 설정한 경우에만 사용한다.

### **data\_buffer\_neighbor\_flush\_pages**

**data\_buffer\_neighbor\_flush\_pages** 는 백그라운드 플러시(희생자 후보 플러시) 기능을 이용해 플러시될 인접 페이지 수를 관리하는 파라미터이다. 이 값이 1 이하인 경우 인접 플러시 기능이 비활성화된 것으로 간주한다.

### **data\_buffer\_neighbor\_flush\_nondirty**

**data\_buffer\_neighbor\_flush\_nondirty** 인접 플러시 기능이 활성화상태이고 (**data\_buffer\_neighbor\_flush\_pages** 가 1보다 큰 경우) 희생자 후보 페이지가 플러시될 때, 더티 페이지에 인접한 더티가 아닌 단일 페이지들도 함께 플러시된다.

### **tde\_keys\_file\_path**

**tde\_keys\_file\_path** 는 TDE를 위한 키 파일의 경로를 설정하는 파라미터이다. 키 파일의 이름은 [database\_name]\_keys 로 고정되어 있고, 해당 키 파일이 존재하는 디렉토리를 지정한다. 이 시스템 파라미터가 설정되지 않았을 경우에는 데이터베이스 볼륨과 같은 위치에서 키 파일을 찾는다. 키 파일에 대한 자세한 설명은 [파일 기반의 마스터 키 관리](#) 를 참고한다.

### **tde\_default\_algorithm**

**tde\_default\_algorithm** 는 TDE 암호화 테이블 생성 시에 사용하는 기본 알고리즘을 설정하는 파라미터이다. 로그 및 임시 데이터는 항상 이 파라미터로 설정된 알고리즘을 이

용하여 암호화된다. **AES** 와 **ARIA** 가 설정 가능하다. 암호화 알고리즘에 관한 자세한 내용은 **암호화 알고리즘** 을 참고한다.

### recovery\_progress\_logging\_interval

**recovery\_progress\_logging\_interval** 는 복구 (recovery) 의 상세과정을 로그에 출력할지 여부와 출력 주기를 설정한다. 분석(analysis), 리두(redo), 언두(undo) 각 단계별 전체 작업량과 현재 작업량, 앞으로 남은 시간 예상치를 출력한다. 5초보다 작게 설정될 경우 5초로 설정된다.

### supplemental\_log

**supplemental\_log** 는 CDC (Change Data Capture) 또는 **flashback** 지원에 필요한 정보를 로그 볼륨에 기록할지 여부를 결정한다. CDC와 flashback은 물리적 로그를 해석하여 트랜잭션이 데이터베이스를 논리적으로 어떻게 변경했는지 추론한다. 이때 물리적 로그를 해석하기 위해서는 추가 정보가 필요하고 이를 위해 **supplemental\_log** 를 사용한다. 이 시스템 파라미터를 0보다 크게 설정하면 기존 트랜잭션 로그 외에도 더 많은 로그가 생성되고 저장되므로 성능과 로그 저장공간에 영향을 끼친다. 이 시스템 파라미터를 1로 설정하면 사용자가 실행한 DML 및 DDL을 해석하는 데 필요한 정보가 기록된다. 또한 이 시스템 파라미터를 2로 설정하면 DML 해석에 필요한 정보만 기록된다.

### regexp\_engine

**regexp\_engine** 는 정규식 연산자와 함수가 수행할 라이브러리를 선택하는 파라미터이다. **cppstd** 또는 **re2**를 설정할 수 있고 기본값은 **re2**이다. 정규 표현식 기능에 관한 자세한 내용은 **정규 표현식 함수와 연산자** 을 참고한다.

### vacuum\_ovfp\_check\_threshold

인덱스 vacuum 과정에서 읽은 하나의 단말 노드 (leaf node) 에 연결된 오버플로우 페이지의 개수가 어떤 임계값 보다 큰 경우 관리 대상으로 정보를 수집한다. **vacuum\_ovfp\_check\_threshold**로 이 임계 값을 지정할 수 있다. 기본값은 1000 페이지이다.

### vacuum\_ovfp\_check\_duration

**vacuum\_ovfp\_check\_duration**는 vacuum 쓰레드에 의해 수집된 인덱스 오버플로우 수와 그 관련 정보를 유지하는 기간을 지정한다. 설정한 기간 내에 업데이트되지 않은 정보는 제거된다. 값의 단위는 분이다.

### deduplicate\_key\_level

인덱스 생성 구문에서 묵시적으로 지정되는 *deduplicate level*값을 지정한다. 기본값은 -1이다. **테이블 정의문**, **인덱스 정의문**를 참고한다.

## 참고

- **deduplicate\_key\_level**를 **-1**로 지정하는 경우에는 `CREATE INDEX` 구문에서 명시적으로 *deduplicate level*을 지정하더라도 무시된다.

## print\_index\_detail

**SHOW CREATE TABLE** 구문과 같이 인덱스 구문 정보를 보여줄 때 **WITH** 절의 옵션 정보의 출력 여부를 지정한다. 기본값은 NO이다. 단, `unloaddb` 툴에서는 이 설정값의 영향을 받지 않는다.

## enable\_memory\_monitoring

**enable\_memory\_monitoring**은 서버의 힙 메모리 사용량 모니터링 여부를 지정하며, 기본값은 NO이다. 이 값을 수정 후 적용하기 위해서는 서버 재구동이 필요하다. 이 값을 YES로 설정하면, 서버의 메모리 모니터링 기능이 활성화되어 서버의 힙 메모리 사용량을 지속적으로 추적 관리한다. 힙 메모리 사용량은 큐브리드 소스 코드에서 메모리 동적 할당이 발생하는 파일과 라인을 기준으로 추적되며, 동일한 위치에서 여러 번 메모리 할당이 발생하는 경우 메모리 사용량은 누적된다. 추적 중인 메모리가 해제되면 누적된 총 메모리 사용량에서 해제된 메모리 크기 만큼 감산하여 실시간 힙 메모리 사용량을 계속 추적한다. 추적 중인 힙 메모리 사용량은 `memmon` 유틸리티를 통해 확인할 수 있다.

## 참고

- Linux 계열에서만 사용할 수 있다.
- 메모리 사용량 모니터링 기능은 서버에서 메모리를 할당할 때마다 사용량을 추적하는 연산이 추가되어 비용이 발생한다. 특히 HA 환경에서는 슬레이브 노드가 마스터 노드의 로그를 처리하는 과정에서 메모리 할당 비중이 높아져 성능 문제가 발생할 수 있다. 따라서 슬레이브 노드에서는 이 기능의 사용을 권장하지 않는다.
- 큐브리드에서 추적하고 있는 메모리 사용량은 `pmap -d`, `htop`에서 표기되는 메모리 사용량과는 다를 수 있다. 이는 큐브리드의 메모리 사용량 모니터링 기능을 코드 수준으로 자동화하는 과정에서 발생한 `glibc`와의 충돌을 막기 위해 헤더 파일 및 `glibc` 내부에서 발생하는 메모리 할당을 추적하지 않기 때문이며, 다음과 같은 추가적인 차이점이 존재한다.
  - `pmap -d`에서 프로세스의 힙 메모리 사용량은 `writable/private` 항목을 통해 확인할 수 있다. 그러나 이는 운영체제의 메모리 관리 정책에 따른 프로세스의 메모리 점유량을 보여주기 때문에 큐브리드의 메모리 사용량 모니터링 기능이 보여주는 사용량과 차이가 발생할 수 있다.

- htop은 RES(Resident Size) 항목을 통해 프로세스의 메모리 사용량을 제공하나 이는 프로세스의 물리 메모리 사용량을 보여주는 것이기 때문에 프로세스의 힙 메모리 사용량 만을 확인할 수는 없다.

## flashback\_timeout

flashback 유틸리티 실행 시, 사용자가 되돌릴 트랜잭션 식별자를 입력해야 하는 제한 시간을 설정한다. 설정된 제한 시간 이내에 트랜잭션 식별자를 입력하지 않으면 오류 메시지를 출력하고 flashback 유틸리티는 종료된다. 트랜잭션을 되돌리기 위해 보관 로그 볼륨을 계속 참조하기 때문에 제한 시간이 없다면 보관 로그 볼륨들이 삭제되지 못하고 쌓이는 문제가 발생할 수 있다. 따라서 무제한 대기를 위한 설정값은 제공하지 않으며 기본값은 300초이다.

## 브로커 설정

### cubrid\_broker.conf 설정 파일과 기본 제공 파라미터

#### 브로커 시스템 파라미터

다음은 **cubrid\_broker.conf** 설정 파일에 사용할 수 있는 브로커 파라미터이다. 각 파라미터에 대한 설명은 **공통 적용 파라미터** 및 **브로커별 파라미터** 를 참조한다. 동적으로 설정값 변경이 가능한 파라미터는 **broker\_changer** 유틸리티를 이용하여 한시적으로 변경할 수 있다. **cubrid broker restart** 로 전체 브로커를 재시작한 후에도 값이 적용되도록 하려면 **cubrid\_broker.conf** 에 설정된 값을 변경해 두어야 한다.

| 적용 구분      | 용도 | 파라미터 이름             | 타입     | 기본값  | 동적 변경 |
|------------|----|---------------------|--------|------|-------|
| 공통 적용 파라미터 | 접속 | ACCESS_CONTROL      | bool   | no   |       |
|            |    | ACCESS_CONTROL_FILE | string |      |       |
|            |    |                     | bool   | DENY |       |

| 적용 구분     | 용도                  | 파라미터 이름                       | 타입     | 기본값                          | 동적 변경 |
|-----------|---------------------|-------------------------------|--------|------------------------------|-------|
|           |                     | ACCESS_CONTROL_DEFAULT_POLICY |        |                              |       |
|           | 로그                  | ADMIN_LOG_FILE                | string | log/broker/cubrid_broker.log |       |
|           | 브로커 서버 (cub_broker) | MASTER_SHM_ID                 | int    | 30,001                       |       |
| 브로커별 파라미터 | 접속                  | ACCESS_LIST                   | string |                              |       |
|           |                     | ACCESS_MODE                   | string | RW                           | 가능    |
|           |                     | BROKER_PORT                   | int    | 30,000(최대 값: 65,535)         |       |
|           |                     | CONNECT_ORDER                 | string | SEQ                          | 가능    |
|           |                     | ENABLE_MONITOR_HANG           | string | OFF                          |       |
|           |                     | KEEP_CONNECTION               | string | AUTO                         | 가능    |
|           |                     | MAX_NUM_DELAYED_HOSTS_LOOKUP  | int    | -1                           |       |
|           |                     |                               | string |                              | 가능    |

| 적용 구분          | 용도 | 파라미터 이름                         | 타입     | 기본값                                                                                | 동적 변경 |
|----------------|----|---------------------------------|--------|------------------------------------------------------------------------------------|-------|
|                |    | PREFERRED_H<br>OSTS             |        |                                                                                    |       |
|                |    | RECONNECT_TIME                  | sec    | 600                                                                                | 가능    |
|                |    | REPLICA_ONLY                    | string | OFF                                                                                |       |
| 브로커 응용 서버(CAS) |    | APPL_SERVER_MAX_SIZE            | MB     | Windows 32<br>비트: 40,<br>Windows 64<br>비트: 80,<br>Linux: 0 (최<br>대값:<br>2,097,151) | 가능    |
|                |    | APPL_SERVER_MAX_SIZE_HARD_LIMIT | MB     | 1,024(최대값:<br>2,097,151)                                                           | 가능    |
|                |    | APPL_SERVER_PORT                | int    | BROKER_PORT+1                                                                      |       |
|                |    | APPL_SERVER_SHM_ID              | int    | 30,000                                                                             |       |
|                |    | AUTO_ADD_APPL_SERVER            | string | ON                                                                                 |       |
|                |    | MAX_NUM_APPL_SERVER             | int    | 40                                                                                 |       |
|                |    |                                 | int    | 5                                                                                  |       |



| 적용 구분     | 용도 | 파라미터 이름                 | 타입     | 기본값             | 동적 변경 |
|-----------|----|-------------------------|--------|-----------------|-------|
|           |    | MIN_NUM_APPL_SERVER     |        |                 |       |
|           |    | TIME_TO_KILL            | sec    | 120             | 가능    |
| 트랜잭션 및 질의 |    | CCI_DEFAULT_AUTOCOMMIT  | string | ON              |       |
|           |    | LONG_QUERY_TIME         | sec    | 60(최대값: 86,400) | 가능    |
|           |    | LONG_TRANSACTION_TIME   | sec    | 60(최대값: 86,400) | 가능    |
|           |    | MAX_PREPARED_STMT_COUNT | int    | 2,000(최소값: 1)   | 가능    |
|           |    | MAX_QUERY_TIMEOUT       | sec    | 0(최대값: 86,400)  | 가능    |
|           |    | SESSION_TIMEOUT         | sec    | 300             | 가능    |
|           |    | STATEMENT_POOLING       | string | ON              | 가능    |
|           |    | JDBC_CACHE              | string | OFF             | 가능    |

| 적용 구분 | 용도 | 파라미터 이름                 | 타입     | 기본값                      | 동적 변경 |
|-------|----|-------------------------|--------|--------------------------|-------|
| 로그    |    | JDBC_CACHE_HINT_ONLY    | string | OFF                      | 가능    |
|       |    | JDBC_CACHE_LIFE_TIME    | sec    | 1000                     | 가능    |
|       |    | TRIGGER_ACTI<br>ON      | string | ON                       | 가능    |
|       |    | ACCESS_LOG              | string | OFF                      | 가능    |
|       |    | ACCESS_LOG_<br>DIR      | string | log/broker               |       |
|       |    | ACCESS_LOG_<br>MAX_SIZE | KB     | 10M(최대값:<br>2G)          | 가능    |
|       |    | ERROR_LOG_<br>DIR       | string | log/broker/<br>error_log | 가능    |
|       |    | LOG_DIR                 | string | log/broker/<br>sql_log   | 가능    |
|       |    | SLOW_LOG                | string | ON                       | 가능    |
|       |    | SLOW_LOG_<br>DIR        | string | log/broker/<br>sql_log   | 가능    |
|       |    | SQL_LOG                 | string | ON                       | 가능    |
|       |    |                         | KB     | 10000                    | 가능    |

| 적용 구분 | 용도 | 파라미터 이름                 | 타입     | 기본값                  | 동적 변경 |
|-------|----|-------------------------|--------|----------------------|-------|
|       |    | SQL_LOG_MAX_SIZE        |        |                      |       |
|       | 샤드 | SHARD                   | string | OFF                  |       |
|       |    | SHARD_CONNECTION_FILE   | string | shard_connection.txt |       |
|       |    | SHARD_DB_NAME           | string |                      |       |
|       |    | SHARD_DB_PASSWORD       | string |                      |       |
|       |    | SHARD_DB_USER           | string |                      |       |
|       |    | SHARD_IGNORE_HINT       | string | OFF                  |       |
|       |    | SHARD_KEY_FILE          | string | shard_key.txt        |       |
|       |    | SHARD_KEY_FUNCTION_NAME | string |                      |       |
|       |    | SHARD_KEY_LIBRARY_NAME  | string |                      |       |

| 적용 구분 | 용도 | 파라미터 이름                       | 타입     | 기본값                  | 동적 변경 |
|-------|----|-------------------------------|--------|----------------------|-------|
|       |    | SHARD_KEY_MODAL               | int    | 256                  |       |
|       |    | SHARD_MAX_CLIENTS             | int    | 256                  |       |
|       |    | SHARD_MAX_PREPARED_STMT_COUNT | int    | 10,000               |       |
|       |    | SHARD_NUM_PROXY               | int    | 1                    |       |
|       |    | SHARD_PROXY_CONN_WAIT_TIMEOUT | sec    | 8h                   |       |
|       |    | SHARD_PROXY_LOG               | string | ERROR                | 가능    |
|       |    | SHARD_PROXY_LOG_DIR           | string | log/broker/proxy_log |       |
|       |    | SHARD_PROXY_LOG_MAX_SIZE      | KB     | 100,000              | 가능    |
|       |    | SHARD_PROXY_SHM_ID            | int    |                      |       |
|       |    | SHARD_PROXY_TIMEOUT           | sec    | 30(초)                |       |

| 적용 구분 | 용도 | 파라미터 이름           | 타입     | 기본값        | 동적 변경 |
|-------|----|-------------------|--------|------------|-------|
|       | 기타 | MAX_STRING_LENGTH | int    | -1         |       |
|       |    | SERVICE           | string | ON         |       |
|       |    | SSL               | string | OFF        |       |
|       |    | SOURCE_ENV        | string | cubrid.env |       |
|       |    | NET_BUF_SIZE      | KB     | 16K        |       |

## 기본 제공 파라미터

CUBRID 설치 시 생성되는 기본 브로커 설정 파일인 **cubrid\_broker.conf**에는 브로커 파라미터 중에서 반드시 변경해야 할 일부 파라미터가 기본으로 포함된다. 기본으로 포함되지 않는 파라미터의 설정값을 변경하기 원할 경우 직접 추가/편집해서 사용하면 된다.

다음은 설치 시 기본으로 제공되는 **cubrid\_broker.conf** 파일 내용이다.

```
[broker]
MASTER_SHM_ID =30001
ADMIN_LOG_FILE =log/broker/cubrid_broker.log

[%query_editor]
SERVICE =ON
BROKER_PORT =30000
MIN_NUM_APPL_SERVER =5
MAX_NUM_APPL_SERVER =40
APPL_SERVER_SHM_ID =30000
LOG_DIR =log/broker/sql_log
ERROR_LOG_DIR =log/broker/error_log
SQL_LOG =ON
TIME_TO_KILL =120
SESSION_TIMEOUT =300
KEEP_CONNECTION =AUTO

[%BROKER1]
SERVICE =ON
```

```

BROKER_PORT =33000
MIN_NUM_APPL_SERVER =5
MAX_NUM_APPL_SERVER =40
APPL_SERVER_SHM_ID =33000
LOG_DIR =log/broker/sql_log
ERROR_LOG_DIR =log/broker/error_log
SQL_LOG =ON
TIME_TO_KILL =120
SESSION_TIMEOUT =300
KEEP_CONNECTION =AUTO

```

## 브로커 설정 파일 관련 환경 변수

**CUBRID\_BROKER\_CONF\_FILE** 환경 변수를 사용하여 **cubrid-broker.conf** 파일의 위치를 지정할 수 있다. 서로 다른 구성으로 여러 개의 브로커를 실행할 때 사용한다.

## 공통 적용 파라미터

다음은 브로커 전체에 공통으로 적용되는 파라미터로 [broker] 아래에 작성된다.

## 접속

### ACCESS\_CONTROL

**ACCESS\_CONTROL** 은 브로커에 접속하는 응용 클라이언트를 제한하기 위한 파라미터이다. 기본값은 **OFF** 이다. 자세한 내용은 **브로커 서버 접속 제한** 을 참고한다.

### ACCESS\_CONTROL\_FILE

**ACCESS\_CONTROL\_FILE** 은 브로커에 접속을 허용하는 데이터베이스 이름, 데이터베이스 사용자 ID, IP 목록을 저장한 파일 이름을 지정하는 파라미터이다. IP 목록은 하나의 브로커 내에서 <db\_name>:<db\_user> 별로 최대 256 라인까지 작성될 수 있다. 자세한 내용은 **브로커 서버 접속 제한** 을 참고한다.

### ACCESS\_CONTROL\_DEFAULT\_POLICY

**ACCESS\_CONTROL\_FILE** 에 지정한 브로커가 없고 **ACCESS\_CONTROL\_DEFAULT\_POLICY** 의 값이 **ALLOW** 인 경우, 브로커에 접속을 모두 허용한다. 기본값은 **DENY** 이다. 자세한 내용은 **브로커 서버 접속 제한** 을 참고한다.

## 참고

**ACCESS\_CONTROL\_DEFAULT\_POLICY**의 설정값은 **ACCESS\_CONTROL**이 **ON**일 때만 유효하며, **OFF**일 경우에는 설정값이 적용되지 않는다.

## 로그

### ADMIN\_LOG\_FILE

**ADMIN\_LOG\_FILE**은 CUBRID 브로커의 구동에 관한 시간 기록을 저장하는 파일을 지정하기 위한 파라미터이다. 기본값은 **log/broker/cubrid\_broker.log** 파일이다.

## 브로커 서버(cub\_broker)

### MASTER\_SHM\_ID

**MASTER\_SHM\_ID**는 CUBRID 브로커를 관리하기 위해 사용되는 공유 메모리의 ID를 설정하는 파라미터로, 이 값은 시스템 내에서 유일한 값이어야 한다. 기본값은 **30,001**로 설정된다.

## 브로커별 파라미터

다음은 브로커에 개별적으로 적용되는 파라미터로 `[%broker_name]` 아래에 각각 작성된다. `broker_name`의 최대 길이는 영문 63자이다.

## 접속

### ACCESS\_LIST

**ACCESS\_LIST**는 CUBRID 브로커로 접근을 허용하는 응용 클라이언트의 IP 주소 리스트를 저장할 파일 이름을 지정하는 파라미터이다. 192.168.10.\*와 192.168.20.\*인 IP 주소의 접근을 허용하려면 이를 임의의 파일(ip\_lists.txt)에 저장하여 이 파라미터의 값으로 파일명을 설정한다.

### ACCESS\_MODE

**ACCESS\_MODE**는 브로커의 모드를 설정하는 파라미터로 기본값은 **RW**이다. 자세한 내용은 `cubrid_broker.conf`를 참고한다.

### BROKER\_PORT

**BROKER\_PORT**는 해당 브로커의 포트 번호를 지정하기 위한 파라미터로 시스템 내에서 유일한 값이면서 65,535 이하의 값이어야 한다. **query\_editor**의 브로커 포트는 기본값이 **30,000**으로 설정되며, **broker1**의 브로커 포트는 기본값이 **33,000**으로 설정된다.

### CONNECT\_ORDER

CAS가 연결할 호스트 순서를 결정할 때 **\$CUBRID\_DATABASES/databases.txt**에 설정된 호스트에서 순서대로 연결을 시도할지 랜덤한 순서대로 연결을 시도할지를 지정하는 파라미터이다. 기본값은 **SEQ**이며 순서대로 연결을 시도한다. 이 값이 **RANDOM**이면 랜덤한 순서대로 연결을 시도한다. **PREFERRED\_HOSTS** 파라미터 값이 명시되어 있으면 먼저 **PREFERRED\_HOSTS**에 명시된 호스트의 순서대로 연결을 시도한 후 실패할 경우에만 **\$CUBRID\_DATABASES/databases.txt**의 설정 값을 사용한다.

## ENABLE\_MONITOR\_HANG

**ENABLE\_MONITOR\_HANG**은 일정 비율 이상의 CAS가 멈춤(hang) 것으로 판단되면 응용 프로그램이 해당 브로커로의 접속을 차단할 것인지 지정하는 파라미터이다. 이 값이 ON이면 해당 기능을 수행한다. 기본값은 OFF로, 해당 기능을 수행하지 않는다.

브로커 프로세스는 CAS의 멈춤(hang)이 1분 이상 지속되는 경우 CAS를 멈춤(hang) 상태로 판단하고, 해당 CAS의 개수에 따라 해당 브로커 프로세스가 비정상으로 판단되면 정상화되기 전까지 해당 브로커로 접속을 시도하는 응용 프로그램을 차단하여, 접속 URL에 설정한 대체 호스트(altHosts)로의 접속을 유도한다.

## KEEP\_CONNECTION

**KEEP\_CONNECTION**은 CAS와 응용 클라이언트 사이의 연결 방식을 지정하는 파라미터로 **ON / AUTO** 중 하나로 설정된다. 이 파라미터가 **ON**으로 설정되면 커넥션 단위로 CAS와 연결한다. 또한 **AUTO**로 설정되면 CAS의 개수가 클라이언트 개수보다 많은 경우 커넥션 단위로 연결하고, CAS의 개수가 클라이언트의 개수보다 적은 경우 트랜잭션 단위로 연결한다. 기본값은 **AUTO**이다.

## MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP

databases.txt의 db-host에 여러 대의 DB 서버를 명시한 HA 환경에서 거의 모든 DB 서버에서 복제 지연이 발생하는 경우, **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP** 파라미터에서 명시한 대수의 복제 지연 서버까지만 연결 여부를 검토한 후 연결을 결정한다(어떤 DB 서버의 복제 지연 여부는 standby 상태의 호스트만을 대상으로 판단하며, **ha\_delay\_limit** 파라미터의 설정에 따라 결정됨). 보다 자세한 사항은 **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP**을 참고한다.

## PREFERRED\_HOSTS

**PREFERRED\_HOSTS**는 CAS가 우선적으로 연결할 호스트 순서를 지정하는 파라미터이다. **PREFERRED\_HOSTS**에 설정된 호스트 순서대로 연결을 시도했으나 실패하는 경우 **\$CUBRID\_DATABASES/databases.txt**에 설정된 호스트 순서대로 연결을 시도한다. 기본값은 **NULL**이다. 자세한 내용은 **cubrid\_broker.conf**를 참고한다.

## RECONNECT\_TIME

특정 상황에서 **RECONNECT\_TIME**으로 명시한 시간이 경과하면 CAS가 다른 DB 서버에 재연결을 시도한다. 기본값은 600s(10min)이다. 값 뒤에 ms, s, min, h의 단위 지정이 가



능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다.

CAS가 서버에 재연결을 시도하는 특정 상황은 다음과 같다.

- CAS가 **PREFERRED\_HOSTS**에 명시한 DB 서버가 아닌 databases.txt의 db-host에 명시한 DB 서버에 연결한 경우
- “ACCESS\_MODE=RO”(Read Only)인 CAS가 standby DB 서버가 아닌 active DB 서버에 연결한 경우
- CAS가 복제 지연 상태인 DB 서버와 연결한 경우

**RECONNECT\_TIME** 값이 0이면 재연결을 시도하지 않는다.

## REPLICA\_ONLY

**REPLICA\_ONLY**의 값이 **ON**이면 CAS가 레플리카에만 접속된다. 기본값은 **OFF**이다. **REPLICA\_ONLY**의 값이 **ON** 이더라도 **ACCESS\_MODE**의 값이 **RW**이면 레플리카 DB에 직접 접속하여 쓰기 작업을 수행할 수 있다. 단, 레플리카에 직접 쓰는 데이터는 복제되지 않는다.

### 참고

레플리카에 직접 데이터를 쓰는 경우 복제 불일치가 발생함에 주의해야 한다.

## 브로커 응용 서버(CAS)

### APPL\_SERVER\_MAX\_SIZE

**APPL\_SERVER\_MAX\_SIZE**는 CAS가 처리하는 프로세스 메모리 사용량의 최대 크기를 지정하는 파라미터이다. 값 뒤에 B, K, M, G로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes를 의미한다. 단위 생략 시 M으로 지정된다.

이 파라미터는 진행 중인 트랜잭션이 있을 경우 사용자에게 의해 정상 종료(커밋 혹은 롤백)되기를 기다렸다가 CAS를 재구동하는 동작에 영향을 준다.

**APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**는 **APPL\_SERVER\_MAX\_SIZE**와 비슷하지만, 진행 중인 트랜잭션이 있을 경우 이를 강제 종료(롤백)하고 CAS를 재구동하는 동작에 영향을 준다는 점이 다르다.

**APPL\_SERVER\_MAX\_SIZE** 파라미터는 Windows 버전과 Linux 버전의 기본값이 다르므로 주의한다.

Windows 버전의 CUBRID는 32비트 버전에서는 **APPL\_SERVER\_MAX\_SIZE**의 기본값이 **40** (MB)이고, 64비트 버전에서는 **80** (MB)이다. 최대값은 Windows 버전과 Linux 버전 모

두 동일하며 2,097,151 (MB)이다. 현재 프로세스의 크기가 **APPL\_SERVER\_MAX\_SIZE**의 값을 초과하면, 브로커가 해당 CAS를 재구동한다.

Linux 버전의 CUBRID는 **APPL\_SERVER\_MAX\_SIZE**의 기본값이 0이고, 다음의 경우에 해당 CAS를 재구동한다.

- **APPL\_SERVER\_MAX\_SIZE**의 값이 0 또는 음수인 경우: 현재 프로세스의 크기가 CAS의 초기 메모리의 2배가 될 때
- **APPL\_SERVER\_MAX\_SIZE**의 값이 양수인 경우: **APPL\_SERVER\_MAX\_SIZE**의 설정 값을 초과할 때

#### 참고

이 값을 너무 작게 설정하면 CAS가 빈번하게 재구동될 수 있으므로 주의한다. 일반적으로 **APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**의 값을 **APPL\_SERVER\_MAX\_SIZE**의 값보다 크게 설정한다. 자세한 내용은 **APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**의 설명을 참고한다.

### APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT

**APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**는 CAS가 처리하는 프로세스 메모리 사용량의 최대 크기를 지정하는 파라미터이다. 값 뒤에 B, K, M, G로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes를 의미한다. 단위 생략 시 M으로 지정된다. 기본 값은 1,024 (MB)이며, 최대값은 2,097,151 (MB)이다.

이 파라미터는 진행 중인 트랜잭션이 있어도 이를 강제 종료(롤백)하고 CAS를 재구동하는 동작에 영향을 준다. **APPL\_SERVER\_MAX\_SIZE**는 **APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**와 비슷하지만, 진행 중인 트랜잭션이 있을 경우 사용자에게 의해 정상 종료(커밋 혹은 롤백)되기를 기다렸다가 CAS를 재구동하는 동작에 영향을 준다는 점이 다르다.

#### 참고

이 값을 너무 작게 설정하면 CAS가 빈번하게 재구동될 수 있으므로 주의한다. CAS를 재구동할 때 메모리 사용량이 증가해도 트랜잭션이 정상 종료되기까지 기다리기 위해 **APPL\_SERVER\_MAX\_SIZE**를 설정하고, 메모리 사용량이 허용하는 기준을 넘으면 트랜잭션을 강제 종료하기 위해 **APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**를 설정한다. 따라서, 일반적으로 **APPL\_SERVER\_MAX\_SIZE\_HARD\_LIMIT**의 값을 **APPL\_SERVER\_MAX\_SIZE**의 값보다 크게 설정한다.

## APPL\_SERVER\_PORT

**APPL\_SERVER\_PORT** 는 Windows 운영체제에만 적용가능한 파라미터로, 클라이언트와 CAS의 통신 포트를 별도로 지정하기 위한 파라미터이다.

Linux 운영체제에서는 클라이언트가 브로커와 연결을 하면, 브로커는 CAS를 할당한 후 클라이언트와의 네트워크 연결을 그대로 CAS에게 전달하기 때문에 클라이언트는 재 접속 과정 없이 CAS와 통신이 가능하다. 이 경우, 별도의 **APPL\_SERVER\_PORT** 는 사용되지 않는다.

반면 Windows에서는 모든 CAS가 독립적인 네트워크 포트를 할당받아 네트워크 연결을 기다리고 있으며, 클라이언트가 브로커에 연결하면 브로커는 가용한 CAS로 접속하기 위한 포트 번호를 클라이언트에게 전달한다. 순차적으로 클라이언트는 현재의 브로커와 네트워크 연결을 종료하고 브로커에서 전달받은 포트 번호로 CAS와 새롭게 접속을 하게 된다.

**APPL\_SERVER\_PORT** 를 별도로 설정하지 않으면 이 값은 **BROKER\_PORT** 파라미터 값에 1을 더한 값이 된다. 예를 들어, **BROKER\_PORT** 의 값이 30,000이고 **APPL\_SERVER\_PORT** 의 값은 미설정 상태에서 **MIN\_NUM\_APPL\_SERVER** 의 값이 5인 경우 브로커 초기 구동 시 5개의 CAS가 각각 30,001~30,005의 포트를 사용한다. 동일 조건 하에 **APPL\_SERVER\_PORT** 의 값만 35,000라면 브로커 초기 구동 시 5개의 CAS가 각각 35,000~35,004의 포트를 사용한다. CAS의 최대 개수가 **cubrid\_broker\_conf** 의 **MAX\_NUM\_APPL\_SERVER** 파라미터에 의해 제한되므로 설정할 수 있는 CAS의 통신 포트의 개수 역시 최대 **MAX\_NUM\_APPL\_SERVER** 파라미터의 설정값으로 제한된다.

Windows 운영체제에서 응용 클라이언트와 CUBRID 브로커 사이에 방화벽이 존재한다면 반드시 **BROKER\_PORT** 및 **APPL\_SERVER\_PORT** 에서 설정된 통신 포트를 열어야 한다.

### 참고

cub\_master, cub\_broker 프로세스의 유닉스 도메인 소켓 파일 경로를 지정하는 **CUBRID\_TMP** 환경 변수에 대한 내용은 [환경 변수 설정](#) 을 참고한다.

## APPL\_SERVER\_SHM\_ID

**APPL\_SERVER\_SHM\_ID** 는 CAS가 이용하는 공유 메모리 ID를 지정하기 위한 파라미터로 시스템 내에서 유일한 값이어야 한다. 기본값은 해당 브로커의 포트와 동일한 값이다.

## AUTO\_ADD\_APPL\_SERVER

**AUTO\_ADD\_APPL\_SERVER** 는 필요한 경우 CAS를 **MAX\_NUM\_APPL\_SERVER** 값까지 자동으로 증가시킬지 설정하는 파라미터로 ON과 OFF를 가지며, 기본값은 **ON** 이다.

## MAX\_NUM\_APPL\_SERVER

**MAX\_NUM\_APPL\_SERVER** 는 해당 브로커에 동시 접속할 수 있는 CAS의 최대 개수를 설정하는 파라미터로, 기본값은 **40** 이다.

## MIN\_NUM\_APPL\_SERVER

**MIN\_NUM\_APPL\_SERVER** 는 해당 브로커에 대한 연결 요청이 없더라도 기본적으로 대기하고 있는 CAS 프로세스의 최소 개수를 설정하는 파라미터로, 기본값은 **5** 이다.

## TIME\_TO\_KILL

**TIME\_TO\_KILL** 은 자동 추가된 CAS 중 유휴 상태의 CAS를 제거하기 위한 기준 시간을 설정하는 파라미터이다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정되며, 기본값은 **120** s이다.

유휴 상태(idle)란 작업이 없이 쉬고 있는 상태로, 이 상태가 **TIME\_TO\_KILL** 시간 이상 유지되면 해당 CAS를 제거한다.

이 파라미터에 설정된 값은 자동 추가된 CAS에만 적용되므로 **AUTO\_ADD\_APPL\_SERVER** 파라미터가 **ON** 인 경우에만 적용된다. **TIME\_TO\_KILL** 파라미터의 값을 너무 작게 설정하면 CAS의 제거/추가가 너무 빈번하게 발생할 수 있으므로 주의한다.

## 트랜잭션 및 질의

### CCI\_DEFAULT\_AUTOCOMMIT

**CCI\_DEFAULT\_AUTOCOMMIT** 은 CCI 인터페이스 또는 CCI기반 인터페이스(PHP, OLE DB, Perl, Python, Ruby 등)로 개발된 응용 프로그램의 자동 커밋 여부를 설정하는 파라미터로 기본값은 **ON** 이다. 이 파라미터는 JDBC로 개발된 응용 프로그램에는 영향을 끼치지 않는다.

**CCI\_DEFAULT\_AUTOCOMMIT** 파라미터의 값이 OFF인 경우 트랜잭션이 종료될 때까지 브로커 응용 서버(CAS) 프로세스를 점유한 상태가 되므로, **SELECT** 문 수행 시에도 fetch 완료 후 반드시 커밋을 수행할 것을 권장한다.

#### 참고

**CCI\_DEFAULT\_AUTOCOMMIT** 파라미터는 2008 R4.0부터 지원하기 시작했고, 이 때 기본값은 OFF였다. **CCI\_DEFAULT\_AUTOCOMMIT** 을 설정하지 않은 2008 R4.0 혹은 그 전 버전 사용자는 자동 커밋 모드가 OFF이므로, 2008 R4.1 이상 버전으로 업그레이드한 사용자가 기존 응용 프로그램을 그대로 사용하고자 하는 경우, 이 값을 OFF로 설정해야 의도하지 않은 트랜잭션의 자동 커밋을 방지할 수 있다.

### ⚠ 경고

ODBC 드라이버는 **CCI\_DEFAULT\_AUTOCOMMIT**의 설정이 무시되어 항상 OFF인 상태로 동작하므로, 프로그램에서 자동 커밋 여부를 직접 설정해야 한다.

## LONG\_QUERY\_TIME

**LONG\_QUERY\_TIME**은 장기 실행 질의(long-duration query)로 판단될 질의 실행 시간을 설정하는 파라미터이다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다. 기본값은 **60** (초)이고, 최대값은 86,400(하루)이다. 어떤 질의를 수행할 때 이 값을 초과한 시간이 소요되는 경우, “cubrid broker status” 명령에서 출력하는 LONG-Q의 값이 하나 증가하고, 해당 SQL은 CAS의 SQL SLOW 로그 파일(\$CUBRID/log/broker/sql\_log/\*.slow.log)에 기록된다. **SLOW\_LOG** 파라미터를 참고한다.

소수점을 사용하여 밀리초(msec) 단위의 값을 설정할 수 있다. 예를 들어 500밀리초로 설정하려면 값을 0.5로 설정한다.

파라미터 값을 **0**으로 설정하면 장기 실행 질의를 판단하지 않는다.

## LONG\_TRANSACTION\_TIME

**LONG\_TRANSACTION\_TIME**은 장기 실행 트랜잭션(long-duration transaction)으로 판단될 트랜잭션의 실행 시간을 설정하는 파라미터이다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다. 기본값은 **60** (초)이고, 최대값은 86,400(하루)이다.

소수점을 사용하여 밀리초(msec) 단위의 값을 설정할 수 있다. 예를 들어 값을 0.5로 설정하여 500밀리초로 설정할 수 있다.

파라미터 값을 **0**으로 설정하면 장기 실행 트랜잭션을 판단하지 않는다.

## MAX\_PREPARED\_STMT\_COUNT

**MAX\_PREPARED\_STMT\_COUNT**은 사용자(응용 프로그램) 접속 당 허용하는 prepared statement의 개수를 제한하는 파라미터이다. 기본값은 **2,000**이며 최소값은 1이다. 이 파라미터 값을 사용자가 적절히 지정함으로써, 응용 프로그램의 작성 실수로 인해 시스템이 허용하는 메모리를 초과하여 prepared statement 문을 생성하는 것을 사전에 방지할 수 있다.

### i 참고

**broker\_changer** 명령을 이용하여 **MAX\_PREPARED\_STMT\_COUNT**의 값을 동적으로 변경하고자 하는 경우 이전의 설정 값보다 큰 값으로 설정하는 경우만 허용하며, 이전의 설정 값보다 작은 값으로 설정하는 것은 허용하지 않는다.

## MAX\_QUERY\_TIMEOUT

**MAX\_QUERY\_TIMEOUT**은 질의 수행의 타임아웃을 설정하는 브로커 파라미터로, 질의 수행을 시작한 후 지정 시간을 초과하면 수행하던 질의를 멈추고 롤백한다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다. 기본값은 0이며, 무한 대기를 의미한다. 값의 범위는 0부터 86400초(1일)까지이다.

응용 프로그램에서 질의 타임아웃을 설정한 경우, 0을 제외하고 **MAX\_QUERY\_TIMEOUT** 값과 응용 프로그램의 질의 타임아웃 값 중 작은 값을 적용한다.

### 참고

CCI 응용 프로그램의 질의 타임아웃 설정은 `cci_connect_with_url()` 함수, `cci_set_query_timeout()` 함수를 참고하며, JDBC 응용 프로그램의 질의 타임아웃 설정은 `setQueryTimeout` 메서드를 참고한다.

## SESSION\_TIMEOUT

**SESSION\_TIMEOUT**은 응용 클라이언트에 대한 CAS의 세션 타임아웃 값을 설정하는 파라미터이다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다. 기본값은 300 (초)이다.

트랜잭션 시작 이후 커밋 혹은 롤백하지 않은 채로 아무런 요청이 없는 상태에서 이 파라미터가 설정한 시간을 초과하면 해당 응용 클라이언트와 CAS 간의 접속이 종료된다.

## STATEMENT\_POOLING

**STATEMENT\_POOLING**은 statement 풀링 기능의 사용 여부를 설정하는 파라미터로 기본값은 **ON**이다.

CUBRID는 트랜잭션이 커밋 또는 롤백되는 경우, 해당 클라이언트 세션에 존재하는 prepared statement의 핸들을 모두 close하는데, **STATEMENT\_POOLING**의 값이 **ON**인 경우에는 prepared statement의 핸들을 지속적으로 풀에 유지하므로 이를 재사용할 수 있다. 따라서, prepared statement를 재사용하는 일반 응용 프로그램 또는 statement pooling이 구현된 DBCP와 같은 라이브러리가 적용된 환경에서는 반드시 기본 설정(**ON**)을 유지해야 한다.

**STATEMENT\_POOLING** 의 값이 **OFF** 인 상태에서 트랜잭션 커밋 또는 종료 이후 해당 prepared statement를 실행하면 다음과 같은 메시지가 출력된다.

```
Caused by: cubrid.jdbc.driver.CUBRIDException: Attempt to
access a closed Statement.
```

## JDBC\_CACHE

**JDBC\_CACHE** 는 결과 캐시 기능의 사용 여부를 설정하는 파라미터로 기본 값은 **OFF** 이다.

이 파라미터 값을 **ON** 으로 설정하면 JDBC의 모든 SELECT 질의는 **JDBC\_CACHE\_LIFE\_TIME** 으로 설정된 시간 동안 캐시된다.

## JDBC\_CACHE\_HINT\_ONLY

**JDBC\_CACHE\_HINT\_ONLY** 는 결과 캐시 기능을 질의 힌트 `/*+ JDBC_CACHE */`에 의해서만 사용할지의 여부를 설정하는 파라미터로 기본 값은 **OFF** 이다.

질의 힌트가 주어졌을 때 동작은 **JDBC\_CACHE** 설정과 동일하다.

## JDBC\_CACHE\_LIFE\_TIME

**JDBC\_CACHE\_HINT\_ONLY** 는 JDBC 클라이언트의 결과 캐시 유지 시간을 설정하는 파라미터로 기본 값은 1000(초)이다.

캐시 유지 시간 동안에만 결과 캐시는 유효하다. 캐시 유지 시간이 지난 후에는 캐시되었던 결과들은 더 이상 사용할 수 없으며 질의 결과는 다시 캐시된다.

캐시 유지 시간은 **JDBC\_CACHE** 나 **JDBC\_CACHE\_HINT\_ONLY** 파라미터가 **ON** 으로 설정되어 있어야만 동작한다.

## 주의

**JDBC\_CACHE** 와 **JDBC\_CACHE\_HINT\_ONLY**, **JDBC\_CACHE\_LIFE\_TIME** 파라미터 설정은

질의 캐시 관련 파라미터 **max\_query\_cache\_entries** 와 **query\_cache\_size\_in\_pages** 가 0보다 큰 값으로 설정되었을 때만 의미를 가진다.

JDBC 클라이언트 결과 캐시가 동작하기 위해서는 앞서 언급된 3개의 JDBC관련 파라미터 설정과 함께 SELECT 질의가 반드시 질의 힌트 `/*+ QUERY CACHE */`를 포함하고 있어야 한다.

## TRIGGER\_ACTION

이 파라미터를 지정한 브로커에 대해 트리거의 동작을 켜거나 끈다. **ON** 또는 **OFF** 로 값을 지정하며, 기본값은 **ON** 이다.

## 로그

### ACCESS\_LOG

**ACCESS\_LOG** 는 해당 브로커의 접속 로그를 저장할 것인지 지정하는 파라미터로 기본값은 **OFF** 이다. 브로커 접속 로그 파일명은 *broker\_name.access*이고, **\$CUBRID/log/broker** 디렉터리에 저장된다.

### ACCESS\_LOG\_DIR

**ACCESS\_LOG\_DIR** 은 브로커 접속 로그(**ACCESS\_LOG**) 파일이 생성되는 디렉토리를 지정한다. 기본값은 **log/broker** 이다.

### ACCESS\_LOG\_MAX\_SIZE

**ACCESS\_LOG\_MAX\_SIZE** 는 브로커 접속 로그(**ACCESS\_LOG**) 파일의 최대 크기를 지정하며, 브로커 접속 로그 파일이 지정한 크기보다 커지면 *broker\_name.access.YYYYMMDDHHMISS* 형식의 이름으로 백업된 후 새 파일 (*broker\_name.access*)에 로그가 기록된다. 기본값은 10M 이고, 최대 2G로 설정할 수 있으며, 브로커 구동중 동적으로 변경이 가능하다.

### ERROR\_LOG\_DIR

**ERROR\_LOG\_DIR** 은 브로커에 대한 에러 로그가 저장되는 디렉토리를 지정하는 파라미터로, 기본값은 **log/broker/error\_log** 이다. 브로커 에러 로그 파일명은 *broker\_name\_id.err*이다.

### LOG\_DIR

**LOG\_DIR** 은 SQL 로그가 저장되는 디렉토리를 지정하는 파라미터로, 기본값은 **log/broker/sql\_log** 이다. SQL 로그가 기록되는 파일명은 *broker\_name\_id.sql.log* 이다.

### SLOW\_LOG

**SLOW** SQL 로깅 여부를 지정하는 파라미터이다. 기본값은 **ON** 이다. 이 값이 **ON** 이면 **LONG\_QUERY\_TIME** 시간을 초과한 장기 실행(long-duration query) 질의문 또는 에러가 발생한 질의문이 **SLOW** SQL 로그 파일에 저장된다. 생성되는 파일의 이름은 *broker\_name\_id.slow.log* 이며, **SLOW\_LOG\_DIR** 이하에 생성된다.

### SLOW\_LOG\_DIR

**SLOW** SQL 로그 파일이 생성되는 디렉토리를 지정한다. 기본값은 **log/broker/sql\_log** 이다.

### SQL\_LOG

**SQL\_LOG** 는 응용 클라이언트의 요청에 따라 CAS가 처리한 SQL 문에 대해 어떤 로그를 기록할 것인지 결정하는 파라미터로 기본값은 **ON** 이다. 이 파라미터가 **ON** 으로 설정되면, 모든 로그를 기록한다. SQL 로그가 기록되는 파일명은 *broker\_name\_id.sql.log* 이



며, 설치 디렉터리의 **log/broker/sql\_log** 디렉터리에 생성된다. 파라미터 값은 다음과 같다.

- **OFF** : 모든 로그를 기록하지 않음
- **ERROR** : 에러가 발생한 질의에 대한 로그만 기록
- **NOTICE** : 설정된 시간을 초과한 장기 실행 질의/트랜잭션의 로그, 에러가 발생한 질의에 대한 로그 기록
- **TIMEOUT** : 설정된 시간을 초과한 장기 실행 질의/트랜잭션의 로그 기록
- **ON / ALL** : 모든 로그 기록

## SQL\_LOG\_MAX\_SIZE

**SQL\_LOG\_MAX\_SIZE** 는 SQL 로그 파일과 SLOW SQL 로그 파일의 최대 크기를 지정하는 파라미터이다. 값 뒤에 B, K, M, G로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes를 의미한다. 단위 생략 시 K로 지정된다. 기본값은 **10,000** (KB)이다.

- **SQL\_LOG** 파라미터가 **ON** 으로 설정된 경우에 생성되는 SQL 로그 파일의 크기가 파라미터의 설정값에 도달하면 *broker\_name\_id.sql.log.bak* 이 생성된다.
- **SLOW\_LOG** 파라미터가 **ON** 으로 설정된 경우에 생성되는 SLOW SQL 로그 파일의 크기가 이 파라미터의 설정값에 도달하면 *broker\_name\_id.slow.log.bak* 이 생성된다.

## 샤드

SHARD 기능을 사용하려면 **cubrid\_broker.conf.shard**를 참고하여 **cubrid\_broker.conf**에 다음 파라미터들을 설정해야 한다.

### SHARD

SHARD 기능의 활성화 여부를 지정하는 파라미터이다. **ON** 또는 **OFF**로 설정하며, 기본값은 **OFF**이다.

### SHARD\_CONNECTION\_FILE

shard 연결 파일의 경로이다. shard 연결 파일은 **\$CUBRID/conf** 내에 위치해야 한다. 자세한 설명은 **shard 연결 파일** 을 참고한다.

### SHARD\_DB\_NAME

shard DB의 이름으로서 응용의 연결 요청이 유효한지 검사하는 데 사용된다.

### SHARD\_DB\_PASSWORD

backend shard DB의 사용자 비밀번호로서, CAS 프로세스에서 backend DBMS와 연결을 수행하는데 사용되며, 응용의 연결 요청이 유효한지 검사하는 데에도 사용된다. 모든 shard DB의 사용자 비밀번호는 동일해야 한다.

**cubrid\_broker.conf**에 **SHARD\_DB\_PASSWORD**의 값을 노출시키고 싶지 않은 경우 환경 변수를 사용할 수 있다. 환경 변수 이름의 포맷은

**<broker\_name>\_SHARD\_DB\_PASSWORD**

이고, <broker\_name>은 항상 대문자로 변환되어야 한다. 예를 들어, 브로커의 이름이 **shard10**이라면 shard DB 암호를 설정하는 환경 변수 이름은 **SHARD1\_SHARD\_DB\_PASSWORD**가 된다. 단, “cubrid broker restart” 명령으로 SHARD 기능이 재구동되는 경우 **SHARD\_DB\_PASSWORD**의 환경 변수 또는 cubrid\_broker.conf의 **SHARD\_DB\_PASSWORD** 파라미터 값이 반드시 설정되어 있어야 한다.

```
export SHARD1_SHARD_DB_PASSWORD=shard123
```

#### 참고

SHARD\_DB\_USER/SHARD\_DB\_PASSWORD 파라미터는 제거될 예정(deprecated)이므로, 응용 프로그램을 통해 접속 정보를 입력할 것을 권장한다.

## SHARD\_DB\_USER

backend shard DB의 사용자 이름으로서, CAS 프로세스에서 backend DBMS와 연결을 수행하는데 사용되며, 응용의 연결 요청이 유효한지 검사하는 데에도 사용된다. 모든 shard DB의 사용자 이름은 동일해야 한다.

#### 참고

SHARD\_DB\_USER/SHARD\_DB\_PASSWORD 파라미터는 제거될 예정(deprecated)이므로, 응용 프로그램을 통해 접속 정보를 입력할 것을 권장한다.

## SHARD\_IGNORE\_HINT

이 값이 ON이면 특정 shard로 연결하기 위해 제공되는 힌트가 무시되고, 정해진 규칙에 따라 접속할 데이터베이스를 선택한다. 기본값은 **OFF** 이다. 모든 데이터베이스가 같은 데이터로 복제되어 있는 상태에서 읽기 부하를 자동으로 로드 밸런싱하여 처리하고자 할 때 사용할 수 있는 방식이다. 예를 들어 응용 프로그램의 부하를 여러 개의 복제 노드 중 하나에 접속하고자 할 때 특정 shard 하나의 연결만 제공하면 어느 노드(데이터베이스)에 연결할지는 proxy가 자동으로 결정한다.

## SHARD\_KEY\_FILE

shard key 설정 정보 파일의 경로이다. shard key 설정 파일은 **\$CUBRID/conf** 내에 위치해야 한다. 자세한 설명은 **shard key 설정 파일** 을 참고한다.

## SHARD\_KEY\_FUNCTION\_NAME

shard key에 대한 사용자 해시 함수의 이름을 지정하기 위한 파라미터이다. 자세한 내용은 **사용자 정의 해시 함수** 를 참고한다.

## SHARD\_KEY\_LIBRARY\_NAME

shard key에 대한 사용자 해시 함수를 지정하기 위해 실행 시간에 로딩 가능한 라이브러리 경로를 지정한다. **SHARD\_KEY\_LIBRARY\_NAME** 파라미터가 설정된 경우 반드시 **SHARD\_KEY\_FUNCTION\_NAME** 파라미터도 설정되어야 한다. 자세한 내용은 **사용자 정의 해시 함수** 을 참고한다.

## SHARD\_KEY\_MODULAR

내장된 shard key 해시 함수 결과의 범위를 지정하기 위한 파라미터로서, 기본 shard key 해시 함수의 결과는  $\text{shard\_key(정수형)} \% \text{SHARD\_KEY\_MODULAR}$ 이다. 최소값은 1, 최대값은 256이다. 관련된 내용은 **shard key 설정 파일** 과 **사용자 정의 해시 함수** 을 참고한다.

## SHARD\_MAX\_CLIENTS

proxy로 동시에 연결 가능한 응용의 개수이다. 기본값은 256이며, 최대값은 proxy 당 10,000이다.

## SHARD\_MAX\_PREPARED\_STMT\_COUNT

proxy가 관리하는 statement pool의 최대 크기이다. 기본값은 10,000이다.

## SHARD\_NUM\_PROXY

proxy 프로세스의 개수이다. 기본값은 1 이며, 최대값은 8이다.

## SHARD\_PROXY\_CONN\_WAIT\_TIMEOUT

이 파라미터로 명시한 시간 동안 아무런 요청이 없으면 CAS가 DB와의 접속을 끊는다. 기본값은 **8h**이다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다. 예전 비밀번호 정보를 지닌 CAS는 더 이상 사용할 수 없으므로 종료되어야 되는데, 이 기능은 이러한 CAS가 불필요하게 계속 유지되는 것을 방지한다.

## SHARD\_PROXY\_LOG

proxy 로그 레벨로서 다음의 값 중 하나로 설정 가능하다. 프록시 로그 레벨 정책: 상위 level을 설정하면 하위의 모든 로그가 남는다.

- 예) SCHEDULE을 설정하면, ERROR | TIMEOUT | NOTICE | SHARD | SCHEDULE 로그를 모두 남긴다.
- **ALL**: 모든 로그 기록
- **ON**: 모든 로그 기록
- **SHARD**: shard DB 선택과 처리에 대한 로그 기록
- **SCHEDULE**: 작업 할당에 대한 로그 기록
- **NOTICE**: 주요한 알림에 대한 로그 기록
- **TIMEOUT**: 임계 시간 초과에 대한 로그 기록
- **ERROR**: 에러 로그 기록
- **NONE**: 로그 기록하지 않음
- **OFF**: 로그 기록하지 않음

## SHARD\_PROXY\_LOG\_DIR

proxy 로그를 저장할 디렉터리 경로이다.

## SHARD\_PROXY\_LOG\_MAX\_SIZE

proxy 로그 파일의 최대크기이다. 값 뒤에 B, K, M, G의 단위 지정이 가능하며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes를 의미한다. 단위 생략 시 K로 지정된다. 최대 1,000,000(KB)까지 설정할 수 있다.

## SHARD\_PROXY\_SHM\_ID

proxy가 이용하는 공유 메모리 ID 를 지정하기 위한 파라미터로 시스템 내에서 유일한 값이어야 한다.

## SHARD\_PROXY\_TIMEOUT

proxy에서 CAS가 사용 가능해지기를 기다리거나 statement가 준비(prepare)되기를 기다리는 최대 시간. 대기 시간이 만료되면 드라이버의 요청을 에러 처리함. 기본값: 30(초). 이 값이 0이면 시스템 파라미터 query\_timeout의 값에 의해 대기 시간이 결정되며, query\_timeout의 값도 0이면 무한 대기한다. SHARD\_PROXY\_TIMEOUT의 값이 0보다 크면 query\_timeout 값과 SHARD\_PROXY\_TIMEOUT 값 중 큰 값에 의해 대기 시간이 결정된다. 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위가 생략되면 s로 지정된다.

## 참고

### proxy 설정을 위해 필요한 파라미터

CUBRID proxy 설정을 위해서는 SHARD\_MAX\_CLIENTS, MAX\_NUM\_APPL\_SERVER, SHARD\_NUM\_PROXY의 설정이 필요하다.

- Linux에서 proxy 프로세스 한 개당 사용되는 파일 디스크립터(fd)는 다음과 같이 제한된다.
  - $((\text{SHARD\_MAX\_CLIENTS} + \text{MAX\_NUM\_APPL\_SERVER}) / \text{SHARD\_NUM\_PROXY}) + 256 \leq 10,000$

위의 공식을 자세히 설명하면 다음과 같다.

- SHARD\_MAX\_CLIENTS는 SHARD 구성 시스템에 접속하는 응용 프로그램들의 최대 접속 개수를 나타낸다.
- MAX\_NUM\_APPL\_SERVER는 proxy 시스템에 접속하는 모든 CAS들의 최대 접속 개수를 나타낸다.
- SHARD\_NUM\_PROXY는 SHARD 구성 시스템에서 사용하는 proxy 프로세스의 개수를 나타낸다.
- "SHARD\_MAX\_CLIENTS / SHARD\_NUM\_PROXY"는 proxy 프로세스 한 개당 접속하는 응용 프로그램들의 최대 접속 개수이다.
- "MAX\_NUM\_APPL\_SERVER / SHARD\_NUM\_PROXY"는 proxy 프로세스 한 개당 접속하는 CAS들의 최대 접속 개수이다.
- 256은 Linux에서 proxy 프로세스 한 개당 내부적으로 사용되는 fd의 개수이다.

Linux 시스템에서 SHARD 파라미터를 설정하는 예로, 응용 프로그램의 최대 동시 접속 개수(SHARD\_MAX\_CLIENTS)를 5,000으로 하고, CAS의 최대 개수(MAX\_NUM\_APPL\_SERVER)를 200으로 하며, proxy 프로세스의 최대 개수(SHARD\_NUM\_PROXY)를 1로 하면,  $(5,000 + 200) / 1 + 256 = 5,456$  이 되어 10,000보다 작으므로 이러한 설정은 가능하다.

이와 관련하여 각 프로세스 간 연결 관계를 살펴보면 다음과 같다. "proxy"는 "응용 client"와 "CAS" 사이에서 연결을 중재하는 역할을 수행한다.

아래에서 []는 프로세스를 나타내며, ->는 요청하는 방향을 나타낸다.

```
[응용 client] --(최초 접속 요청)-----> [broker] (proxy 선택)
 <--(연결할 proxy 알려줌)----
 -----> [proxy] --(CAS
선택)--> [CAS] ----> [DB server]
```

```

<-----
<-----
<---(이후 같은 proxy 사용)----->
<----->

```

broker는 응용 client가 최초 접속을 요청할 때 proxy를 선택하기 위해 한 번만 사용되고, 이후 CUBRID는 [응용 client] - [proxy] - [CAS] - [DB Server]의 연결을 유지한다.

또한, CAS는 DB server와 연결된 이후 계속 연결을 유지한다.

## 기타

### MAX\_STRING\_LENGTH

**MAX\_STRING\_LENGTH** 는 BIT, VARBIT, CHAR, VARCHAR인 데이터 타입에 대해서 최대 문자열 길이를 지정하는 파라미터이다. 기본값인 **-1** 로 설정되면 데이터베이스에서 정의된 문자열 길이가 그대로 사용되고, 파라미터의 값이 **100** 으로 설정되면 임의의 속성이 VARCHAR(1000)으로 정의되었어도 100으로 정의된 것처럼 동작한다.

### SERVICE

**SERVICE** 는 해당 브로커의 구동 여부를 결정하기 위한 파라미터로, **ON** 또는 **OFF** 의 값으로 설정된다. 기본값은 **ON** 이며, 이 파라미터가 **ON** 으로 설정된 경우에만 해당 브로커를 구동할 수 있다.

### SSL

**SSL** 은 해당 브로커에 패킷 암호화 (**SSL**) 적용 여부를 결정하기 위한 파라미터로, **ON** 또는 **OFF** 의 값으로 설정된다. 기본값은 **OFF** 이며, 이 파라미터가 **ON** 으로 설정된 경우 브로커/CAS는 **TLS** 를 이용한 패킷 암호화 모드로 동작한다.

#### ⚠ 경고

브로커가 패킷 암호화 모드로 설정된 경우 (**SSL=ON**), **jdbc** 등의 클라이언트 등은 보안모드로 접속하여야 한다. 그렇지 않은 경우 연결 요청은 브로커에 의해서 거부된다. 반대로 이 파라미터가 **OFF** 로 설정된 경우, 보안 모드가 설정된 client로부터의 접속 요청은 거부된다.

### SOURCE\_ENV

**SOURCE\_ENV** 는 브로커 각각에 대해 개별적으로 운영체제 환경 변수를 설정할 수 있는 파일을 정하는 파라미터로, 파일 확장자는 반드시 **env** 여야 한다. **cubrid.conf** 에서 지정

하는 모든 파라미터는 환경 변수를 통해서도 설정할 수 있다. 예를 들어, **cubrid.conf** 에서 **lock\_timeout** 은 환경 변수 **CUBRID\_LOCK\_TIMEOUT** 으로 지정할 수 있다. 또 다른 예로, broker1에서만 데이터 정의문 수행을 차단하려면 **SOURCE\_ENV** 에서 지정한 파일에 **CUBRID\_BLOCK\_DDL\_STATEMENT** 를 1로 설정하면 된다.

환경변수가 있으면 **cubrid.conf** 보다 우선한다. 기본값은 **cubrid.env** 이다.

## NET\_BUF\_SIZE

**NET\_BUF\_SIZE** 는 **CAS** 가 클라이언트에게 데이터를 전송하기 위해 사용하는 네트워크 버퍼의 크기이다. **CAS** 가 클라이언트에 보낼 결과셋 (ResultSet)들은 네트워크 버퍼에 저장되며, 네트워크 버퍼가 가득찬 경우 클라이언트에 전송된다. 기본값은 16K이며, 32K/48K/64K 중에 선택 가능하다. **NET\_BUF\_SIZE** 는 CAS가 클라이언트에게 한번에 전송하는 패킷 크기이며 이 파라미터는 전송 효율에 영향을 줄 수 있다.

## HA 설정

HA 설정은 **환경 설정**을 참고한다.

## SHARD 설정

SHARD 설정은 **설정**을 참고한다.

# SystemTap

## 개요

SystemTap은 실행 중인 프로세스를 동적으로 모니터링하고 추적할 수 있는 도구이다. CUBRID는 SystemTap을 지원하며, 이를 통해 성능 병목 현상의 원인을 찾아낼 수 있다.

SystemTap 스크립트의 기본 아이디어는 이벤트를 명명하고, 거기에 핸들러를 부여하는 것이다. 핸들러는 이벤트가 발생할 때마다 행해져야 할 작업을 명시하는 스크립트 문장이다.

SystemTap을 사용하여 CUBRID의 성능을 모니터링하려면 먼저 SystemTap을 설치해야 한다. 설치 이후 C 언어와 비슷한 SystemTap 스크립트를 작성, 수행하여 시스템의 성능을 모니터링할 수 있다.

SystemTap은 Linux OS에서만 지원한다.

SystemTap에 대한 자세한 내용 및 설치 방법에 대해서는 <https://sourceware.org/systemtap/index.html>을 참고한다.

## SystemTap 설치하기

### 설치 확인

1. /etc/group 파일에 stapusr, stapdev 그룹 계정이 있는지 확인한다. 이 계정이 없으면 설치가 되지 않은 상태일 것이다.
2. stapusr, stapdev 그룹 계정에 CUBRID를 설치할 때 사용한 사용자 계정을 추가한다. 여기서는 cubrid라고 가정한다.

```
$ vi /etc/group

stapusr:x:156:cubrid
stapdev:x:158:cubrid
```

3. SystemTap의 수행 가능 여부는 간단히 다음 명령을 실행하여 확인할 수 있다.

```
$ stap -ve 'probe begin { log("hello world") exit() }'
```

### 버전

CUBRID에서 SystemTap 스크립트를 실행하려면 SystemTap 2.2 이상 버전을 사용해야 한다.

다음은 CentOS 6.3에서 SystemTap을 설치한 예이다. 버전 확인 및 설치 방법은 각 Linux 배포판마다 다를 수 있다.

1. 현재 설치된 SystemTap 버전을 확인한다.

```
$ sudo yum list|grep systemtap
systemtap.x86_64
1.7-5.el6_3.1 @update
systemtap-client.x86_64
1.7-5.el6_3.1 @update
systemtap-devel.x86_64
1.7-5.el6_3.1 @update
systemtap-runtime.x86_64
1.7-5.el6_3.1 @update
systemtap-grapher.x86_64
1.7-5.el6_3.1 update
systemtap-initscript.x86_64
1.7-5.el6_3.1 update
systemtap-sdt-devel.i686
1.7-5.el6_3.1 update
systemtap-sdt-devel.x86_64
1.7-5.el6_3.1 update
```



```
systemtap-server.x86_64
1.7-5.el6_3.1 update
systemtap-testsuite.x86_64
1.7-5.el6_3.1 update
```

2. SystemTap 2.2 미만 버전이 설치되어 있다면 삭제한다.

```
$ sudo yum remove systemtap-runtime
$ sudo yum remove systemtap-devel
$ sudo yum remove systemtap
```

3. SystemTap 2.2 이상 버전의 RPM 배포 패키지를 설치한다. RPM 배포 패키지는 ( <http://rpmfind.net/linux/rpm2html/>)에서 찾을 수 있다.

```
$ sudo rpm -ivh systemtap-devel-2.3-3.el6.x86_64.rpm
$ sudo rpm -ivh systemtap-runtime-2.3-3.el6.x86_64.rpm
$ sudo rpm -ivh systemtap-client-2.3-3.el6.x86_64.rpm
$ sudo rpm -ivh systemtap-2.3-3.el6.x86_64.rpm
```

## 관련 용어

### 마커(Marker)

코드 안에 위치하는 마커는 실행 중에 제공할 수 있는 함수(프로브)를 호출하기 위한 훅(hook)을 제공한다. 마커가 발동될 때마다 사용자가 제공한 함수가 호출되고, 해당 함수가 종료되면 호출자에게 돌아온다.

마커 발동 시 사용자가 정의하는 함수, 즉 프로브는 추적 및 성능 측정을 위해 사용될 수 있다.

### 프로브(Probe)

프로브(probe)는 어떤 이벤트가 발생했을 때의 동작을 정의하는 일종의 함수로서, 이벤트와 핸들러로 나뉜다.

SystemTap 스크립트에는 특정 이벤트, 즉 마커가 발생할 때의 동작을 정의한다.

SystemTap 스크립트는 여러 개의 프로브를 가질 수 있으며, 프로브의 핸들러는 프로브 바디(probe body)라고 불린다.

SystemTap 스크립트는 코드의 재컴파일 없이 계측 코드의 삽입을 허용하며 핸들러에 관해 더 많은 유연성을 제공한다. 이벤트는 핸들러가 실행하도록 트리거로 동작한다. 핸들러는 데이터를 기록하고 그것을 출력하도록 명시될 수 있다.

CUBRID가 제공하는 마커는 **CUBRID 마커**를 참고한다.

## 비동기 이벤트

비동기 이벤트는 내부에서 정의된 것으로, 코드에서 특정 작업이나 위치에 의존적이지 않다. 이러한 계열의 프로브 이벤트는 주로 카운터, 타이머 등이다.

비동기 이벤트의 예는 다음과 같다.

- begin

SystemTap 세션의 시작.

예) SystemTap이 시작하는 순간.

- end

SystemTap 세션의 끝.

- timer events

핸들러가 주기적으로 실행되는 것을 명시하는 이벤트.

예) 5초마다 “hello world”를 출력한다.

```
probe timer.s(5)
{
 printf("hello world\n")
}
```

## CUBRID에서 SystemTap 사용하기

### CUBRID 소스 빌드

SystemTap 은 리눅스에서만 사용할 수 있다.

CUBRID 소스를 빌드해 SystemTap을 사용하려면 **ENABLE\_SYSTEMTAP** 을 **ON** (기본값)으로 설정한다.

이 옵션은 릴리즈 빌드에 포함되어 있으며, CUBRID 소스 파일을 빌드하지 않고 인스톨 패키지를 이용하여 설치한 사용자라도 SystemTap 스크립트를 이용할 수 있다.

다음은 CUBRID의 소스를 빌드하는 예이다.

```
build.sh -m release
```

## SystemTap 스크립트 실행

CUBRID에서 SystemTap 스크립트 예제는 \$CUBRID/share/systemtap 이하 디렉터리에 제공하고 있다.

다음은 buffer\_access.stp 파일을 수행하는 명령의 예이다.

```
cd $CUBRID/share/systemtap/tapset/scripts
stap -k buffer_access.stp -o result.txt
```

## 결과 출력

특정 스크립트를 수행하면, 스크립트에 기록한 코드에 의해 필요한 정보를 콘솔에 출력한다. -o filename 옵션을 명시하는 경우 해당 옵션에 명시한 filename에 결과를 기록한다.

다음은 앞서 보인 예제의 출력 결과이다.

```
Page buffer hit count: 172
Page buffer miss count: 172
Miss ratio: 50
```

## CUBRID 마커

SystemTap의 가장 유용한 기능은 마커를 사용자 소스 코드(CUBRID 코드) 안에 삽입할 수 있다는 점과 이 마커에 다다를 때 발동하는 프로브를 스크립트에서 작성할 수 있다는 점이다.

## 연결 마커

일정 기간 동안 연결 활동(연결 개수, 연결 지속 시간, 평균 연결 지속 시간, 최대 연결 획득 개수 등)과 관련된 분석에 도움이 되는 정보를 수집하는 것은 관심이 가는 일이다. 이러한 모니터링 스크립트를 작성하기 위해서는 연결 시작 마커와 연결 끝 마커가 필요하다.

conn\_start(connection\_id, user)

서버에서 질의 실행이 시작되면 이 마커가 발동된다.

매개변수:

- **connection\_id** - 연결 ID를 포함한 정수값
- **user** - 연결의 사용자 이름.

conn\_end(connection\_id, user)

서버에서 질의 실행이 끝나면 이 마커가 발동된다.

매개변수:

- **connection\_id** – 연결 ID를 포함한 정수값
- **user** – 연결의 사용자 이름

## 질의 마커

이벤트 관련 질의 실행을 위한 마커로서, 비록 전체 시스템에 관련된 전체(global) 정보를 포함하지는 않지만 모니터링 작업에서 매우 유용하다. 적어도 아래 두 개의 마커는 가장 기본이 되는 것이다. 질의 실행의 시작과 종료 시에 각각 해당 마커가 발동된다.

`query_exec_start(query_string, query_id, connection_id, user)`

서버에서 질의 실행이 시작되면 이 마커가 발동된다.

매개변수:

- **query\_string** – 실행할 질의를 나타내는 문자열
- **query\_id** – 질의 식별자
- **connection\_id** – 연결 식별자
- **user** – 연결할 때 사용하는 사용자 이름

`query_exec_end(query_string, query_id, connection_id, user, status)`

서버에서 질의 실행이 끝나면 이 마커가 발동된다.

매개변수:

- **query\_string** – 실행할 질의를 나타내는 문자열
- **query\_id** – 질의 식별자
- **connection\_id** – 연결 식별자
- **user** – 연결할 때 사용하는 사용자 이름
- **status** – 질의 실행 시 반환 상태(Success, Error)

## 객체 연산 마커

저장 엔진을 포함하는 연산들은 치명적이며 테이블이나 객체 수준에서 업데이트를 조사하는 것(probing)은 데이터베이스의 동작을 모니터링하는 데 큰 도움이 된다. 객체가 매번 INSERT/UPDATE/

DELETE될 때마다 마커가 발동되는데, 이 점은 모니터링 스크립트와 서버 둘 다에 성능 상 약점이 될 수 있다.

`obj_insert_start(table)`

이 마커는 객체가 삽입되기 전에 발동된다.

매개변수:

**table** – 이 연산의 대상 테이블

`obj_insert_end(table, status)`

이 마커는 객체가 삽입된 이후에 발동된다.

매개변수:

· **table** – 이 연산의 대상 테이블

· **status** – 이 연산의 성공 여부를 나타내는 값

`obj_update_start(table)`

이 마커는 객체가 갱신되기 전에 발동된다.

매개변수:

**table** – 이 연산의 대상 테이블

`obj_update_end(table, status)`

이 마커는 객체가 갱신된 후에 발동된다.

매개변수:

· **table** – 이 연산의 대상 테이블

· **status** – 이 연산의 성공 여부를 나타내는 값

`obj_deleted_start(table)`

이 마커는 객체가 삭제되기 전에 발동된다.

매개변수:

**table** – 이 연산의 대상 테이블

`obj_delete_end(table, status)`

이 마커는 객체가 삭제된 후에 발동된다.

매개변수:

- **table** – 이 연산의 타겟 테이블
- **status** – 이 연산의 성공 여부를 나타내는 값

## 인덱스 연산 마커

위의 마커는 테이블 기반 마커이고, 다음은 인덱스 기반 마커이다.

잘못된 인덱스의 사용은 시스템에서 많은 문제를 유발하는 원인이 될 수 있으며, 인덱스를 모니터링 할 수 있다는 점은 매우 유용하다. 아래 마커들은 테이블에서 사용된 마커와 매우 유사한데, 인덱스가 테이블과 같은 연산을 지원하기 때문이다.

`idx_insert_start(classname, index_name)`

이 마커는 B-Tree에 인덱스 노드를 삽입하기 전에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름
- **index\_name** – 대상 인덱스 이름

`idx_insert_end(classname, index_name, status)`

이 마커는 B-Tree에 인덱스 노드를 삽입한 이후에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름
- **index\_name** – 대상 인덱스 이름
- **status** – 연산의 성공 여부를 나타내는 값

`idx_update_start(classname, index_name)`

이 마커는 B-Tree에서 인덱스 노드를 갱신하기 전에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름

- **index\_name** – 대상 인덱스 이름

`idx_update_end(classname, index_name, status)`

이 마커는 B-Tree에서 인덱스 노드를 갱신한 이후에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름
- **index\_name** – 대상 인덱스 이름
- **status** – 연산의 성공 여부를 나타내는 값

`idx_delete_start(classname, index_name)`

이 마커는 B-Tree에서 인덱스 노드를 삭제하기 전에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름
- **index\_name** – 대상 인덱스 이름

`idx_delete_end(classname, index_name, status)`

이 마커는 B-Tree에서 인덱스 노드를 삭제한 후에 발동된다.

매개변수:

- **classname** – 대상 인덱스의 테이블 이름
- **index\_name** – 대상 인덱스 이름
- **status** – 연산의 성공 여부를 나타내는 값

## 잠금(locking) 마커

잠금 이벤트를 포함하는 마커는 아마도 전체를 모니터링하는 작업에서 가장 중요할 것이다. 잠금 시스템은 서버 성능과에 큰 영향을 끼치며, 잠금 대기 시간 및 카운트(교착 상태 및 회피된 트랜잭션 개수)는 문제를 찾는 데 매우 유용하다.

`lock_acquire_start(OID, table, type)`

이 마커는 잠금이 요청되기 전에 발동된다.

매개변수:

- **OID** – 잠금 요청 대상 객체 ID
- **table** – 객체를 유지하고 있는 테이블
- **type** – 잠금 타입(X\_LOCK, S\_LOCK 등)

lock\_acquire\_end(*OID, table, type*)

이 마커는 잠금 요청이 완료된 이후에 발동된다.

매개변수:

- **OID** – 잠금 요청 대상 객체 ID
- **table** – 객체를 유지하고 있는 테이블
- **type** – 잠금 타입(X\_LOCK, S\_LOCK etc.)
- **status** – 잠금 요청의 성공 여부를 나타내는 값

lock\_release\_start(*OID, table, type*)

이 마커는 잠금이 해제된 이후에 발동된다.

매개변수:

- **OID** – 잠금 요청 대상 객체 ID
- **table** – 객체를 유지하고 있는 테이블
- **type** – 잠금 타입(X\_LOCK, S\_LOCK etc.)

lock\_release\_end(*OID, table, type*)

이 마커는 잠금 해제가 완료된 후에 발동된다.

매개변수:

- **OID** – 잠금 요청 대상 객체 ID
- **table** – 객체를 유지하고 있는 테이블
- **type** – 잠금 타입(X\_LOCK, S\_LOCK etc.)



## 트랜잭션 마커

서버 모니터링에서 관심있게 봐야 할 측정 대상은 트랜잭션의 활동이다. 간단한 예로, 트랜잭션이 취소된 개수는 교착 상태가 발생한 개수와 밀접하게 관련되어 있으며, 매우 중요한 성능 식별자라 할 수 있다. 또 다른 직관적인 사용 예는 간단한 SystemTap 스크립트를 사용하여 TPS와 같은 시스템 성능 통계를 수집하는 방법을 단순화하는 것이다.

### tran\_commit(*tran\_id*)

이 마커는 트랜잭션이 성공적으로 완료된 이후에 발동된다.

매개변수:

**tran\_id** – 트랜잭션 식별자

### tran\_abort(*tran\_id, status*)

이 마커는 트랜잭션이 중단(abort)된 이후에 발동된다.

매개변수:

- **tran\_id** – 트랜잭션 식별자
- **status** – 종료 상태

### tran\_start(*tran\_id*)

이 마커는 트랜잭션이 시작된 이후에 발동된다.

매개변수:

**tran\_id** – 트랜잭션 식별자

### tran\_deadlock()

이 마커는 교착상태가 감지된 이후에 발동된다.

## I/O 마커

I/O 액세스는 RDBMS의 주요 병목(bottleneck)이며, I/O 성능을 모니터링하는 마커가 제공되어야 한다. 이 마커를 통해 사용자는 I/O 페이지 액세스 시간을 측정하고, 이 측정에 기반하여 다양하고 복잡한 통계를 집계할 수 있다.

### pgbuf\_hit()

이 마커는 요청한 페이지가 페이지 버퍼에 존재하여 디스크에서 따로 가져올 필요가 없을 때 발동된다.

### pgbuf\_miss()

이 마커는 요청한 페이지가 페이지 버퍼에 존재하지 않아 반드시 디스크에서 가져와야 할 때 발동된다.

`io_write_start(query_id)`

이 마커는 디스크에 페이지를 기록하는 프로세스가 시작할 때 발동된다.

매개변수:

**query\_id** – 질의 식별자

`io_write_end(query_id, size, status)`

이 마커는 디스크에 페이지를 기록하는 프로세스가 종료될 때 발동된다.

매개변수:

- **query\_id** – 질의 식별자
- **size** – 기록되는 바이트 수
- **status** – 연산이 성공적으로 종료되었는지 여부를 나타내는 값

`io_read_start(query_id)`

이 마커는 디스크에서 페이지를 읽는 작업이 시작될 때 발동된다.

매개변수:

**query\_id** – 질의 식별자

`io_read_end(query_id, size, status)`

이 마커는 디스크에서 페이지를 읽는 작업이 종료될 때 발동된다.

매개변수:

- **query\_id** – 질의 식별자
- **size** – 읽은 바이트 수
- **status** – 연산이 성공적으로 종료되었는지 여부를 나타내는 값

## 기타 마커

`sort_start()`

이 마커는 정렬 연산이 시작될 때 발동된다.

`sort_end(nr_rows, status)`

이 마커는 정렬 연산이 완료될 때 발동된다.

매개변수:

- **nr\_rows** – 정렬되는 행의 개수
- **status** – 연산이 성공적으로 종료되었는지 여부

## cubrid 스크립트

### unloaddb.sh 스크립트

**unloaddb**의 실행 시간을 단축하기 위해서 서로 다른 테이블을 언로드 하기 위한 **cubrid unloaddb** 명령어를 동시에 여러 개 실행할 수 있다. **unloaddb.sh**은 이러한 과정을 자동화해주는 shell script로, `$CUBRID/share/scripts`에서 찾을 수 있다.

다수의 자식 process에게 균등하게 unloaddb의 부하를 배분하여 실행시키기 위해서 unloaddb.sh은 catalog에서 언로드 대상 테이블들의 크기 정보를 가져온다 (DBA 권한 필요).

테이블 크기를 기준으로 각 자식 프로세스들에게 언로드할 테이블들이 할당되며, 각 자식 프로세스들은 할당된 테이블들의 unloaddb를 동시에 수행한다.

#### ⚠ 경고

##### 제약사항:

- Linux 환경
- DBA 권한이 있는 사용자만 실행 가능

다음은 **unloaddb.sh**에 대한 [options]이다.

#### **-u user**

언로딩할 데이터베이스의 사용자 계정을 지정한다. 옵션을 지정하지 않으면 기본값은 DBA가 된다 (**user**는 **DBA 권한**이 있어야 한다).

```
sh unloaddb.sh -u dba demodb
```

**-t no-process**

동시에 실행될 자식 프로세스의 개수 (no-process, 기본: 8개, 최대 16개 )

```
sh unloaddb.sh -t 4 demodb
```

**-i input-class-file**

입력 파일에서 지정된 테이블들만 언로드 한다. (기본: 지정된 데이터베이스 전체)

```
sh unloaddb.sh -i table_list.txt demodb
```

다음은 입력 파일 table\_list.txt의 예이다.

```
public.table_1
public.table_2
..
public.table_n
```

**-D output-directory**

스키마와 객체 파일이 생성될 디렉터리를 지정한다. 옵션이 지정되지 않으면 현재 디렉터리에 생성된다.

```
sh unloaddb.sh -D /tmp demodb
```

**-s**

언로드 작업을 통해 생성되는 출력 파일 중 스키마 파일만 생성되도록 지정하는 옵션이다 (**schema only**).

```
sh unloaddb.sh -s demodb
```

**-d**

언로드 작업을 통해 생성되는 출력 파일 중 데이터 파일만 생성되도록 지정하는 옵션이다 (**data only**).

```
sh unloaddb.sh -d demodb
```

**-v**

언로드 작업이 진행되는 동안 데이터베이스의 테이블 크기 등의 부가 정보를 화면에 출력하는 옵션이다 (**verbose mode**).

```
sh unloaddb.sh -v demodb
```

### unloadbdb.sh에 의해 생성되는 파일

- 언로드 대상 테이블 전체에 대한 schema 파일
- 자식 프로세스 숫자 만큼의 object 파일
- 로그 파일: 디렉터리 {데이터베이스 이름}\_unloaddb.log 가 생성되며, 여기에 로그 파일들이 생성된다.

unloaddb 실행 예

```
sh unloaddb.sh -t 4 demodb
```

파일 생성 예

```
* demodb_schema: 언로드된 테이블의 스키마 파일
* demodb_0_objects: 1번째 자식 프로세스가 언로드한 오브젝트 파일
* demodb_1_objects: 2번째 자식 프로세스가 언로드한 오브젝트 파일
* demodb_2_objects: 3번째 자식 프로세스가 언로드한 오브젝트 파일
* demodb_3_objects: 4번째 자식 프로세스가 언로드한 오브젝트 파일
* demodb_unloaddb.log/demodb_0.files: 첫번째 자식 프로세스에 의해서 언로드된 테이블 이름
* demodb_unloaddb.log/demodb_0.pid: 첫번째 자식 프로세스의 프로세스 번호
* demodb_unloaddb.log/demodb_0_unloaddb.log: 첫번째 자식 프로세스가 실행한 'cubrid unloaddb'의 log
* demodb_unloaddb.log/demodb_0.status: 첫번째 자식 프로세스의 실행 결과 성공 여부
```

### 참고

- unloaddb.sh의 실행이 완전히 종료되기 이전에 터미널에서 인터럽트 키 ( **CTRL-C**)를 입력하면, 언로드가 진행중인 unloaddb object 파일들은 삭제된다 (언로드가 완료된 object 파일들은 삭제되지 않는다).

### restore\_to\_newdb.sh 스크립트

사용자 실수나 시스템 오류로 데이터 복구가 필요한 경우, “백업본”을 이용해 운영 중인 서버에 복구를 진행해야 한다. 이때 추가 계정 생성이나 엔진 설치 등 번거로운 작업이 선행되어야 하는 불편함이 있다. restore\_to\_newdb.sh 스크립트는 별도의 추가 작업 없이, CUBRID가 설치된 기존 운영 환경

에서 백업본을 사용해 신규 DB 이름으로 복구할 수 있도록 지원한다. 스크립트 위치는 \$CUBRID/share/scripts 에서 찾을 수 있다.

### ⚠ 경고

#### 제약사항:

- Linux 환경에서만 지원된다.
- 복구 대상 DB의 백업 파일이 존재해야 한다.
- 기존 DB명과 복구할 DB명이 서로 달라야 한다.
- 복구할 DB명은 \$CUBRID/databases.txt에 등록되어 있으면 안된다.

기본 명령어 수행 방법이다.

```
sh restore_to_newdb.sh [options] backedup-database-name new-database-name
```

- **restore\_to\_newdb.sh** : 백업본을 이용해 새로운 DBMS를 생성하는 스크립트이다.
- **backup-database-name** : 복구할 백업 DB의 이름이다.
- **new-database-name** : 새로 생성할 복구 DB의 이름이다.

다음은 **restore\_to\_newdb.sh** 에 대한 [options] 설명이다.

#### -F path

복구할 DB의 데이터베이스 경로를 절대경로로 지정한다. 지정하지 않으면 현재 경로를 기본값으로 사용한다.

```
sh restore_to_newdb.sh -F /home/cubrid/newdbpath backedup-database-name new-database-name
```

#### -B path

백업본이 위치한 경로를 절대경로로 지정한다. 지정하지 않으면 현재 경로를 기본값으로 사용한다.

```
sh restore_to_newdb.sh -B /home/cubrid/backupdbpath backedup-database-name new-database-name
```

#### -d date

지정한 날짜 및 시간까지 복구를 수행한다. 형식은 dd-mm-yyyy:hh:mi:ss이다. (예: 14-10-2008:14:10:00)

이 옵션은 cubrid restoredb의 -d 옵션과 동일하다.

```
sh restore_to_newdb.sh backed-up-database-name new-database-name -
F /home/cubrid -B /home/cubrid/DB/backupdbpath -d 30-10-2025:12:
20:00
```

### -l level

백업 수준(0, 1, 2)을 지정하여 복구를 진행한다. cubrid restoredb의 -l 옵션과 동일하다.

```
sh restore_to_newdb.sh -B /home/cubrid/DB/backupdbpath -l 1
backed-up-database-name new-database-name
```

### -p

로그 아카이브가 없는 경우, 부분 복구를 수행한다. 상세 내용은 cubrid restoredb의 -p 옵션을 참고한다.

```
sh restore_to_newdb.sh -B /home/cubrid/DB/backupdbpath -p
backed-up-database-name new-database-name
```

### -k path

복구 시 필요한 키 파일의 경로를 지정한다. 상세 내용은 cubrid restoredb의 -k 옵션을 참고한다.

```
sh restore_to_newdb.sh -B /home/cubrid/DB/backupdbpath -k /home/
cubdev/DB/backupdbpath/backupdb_bk1_keys backed-up-database-name
new-database-name
```

### restoredb\_newdb.sh에 수행화면

```
sh restore_to_newdb.sh -B /home/cubrid/DB/backupdbpath backed-up-
database-name new-database-name
```

```
Validation passed: newdb_path and vol-path are different.
Validation passed: 'new-database-name' does not exist in /home/
cubrid/CUBRID-11.4-Linux.x86_64/databases/databases.txt.
```

```
Confirmed: Files found for pattern (/home/cubrid/DB/backupdbpath/
backed-up-database-name_bk0v*).
file: /home/cubrid/DB/backupdbpath/backed-up-database-name_bk0v*
Warning: /home/cubrid/CUBRID-11.4-Linux.x86_64/databases/
```

```
databases.txt already exists.
Existing databases.txt backed up as databases.txt.bak.
Restoring... /

CUBRID 11.4

Restoring... |Updated CUBRID-11.4-Linux.x86_64/databases/
databases.txt
successfully.
database restoration completed successfully.
```

복구가 정상 완료되면, 복구된 DBMS를 구동한다.

```
[cubrid ~]$ cubrid server start new-database-name
@cubrid server start: new-database-name

This may take a long time depending on the amount of recovery works
to do.

CUBRID 11.4

++ cubrid server start: success
Calling java stored procedure is allowed
```

#### 참고

- 해당 스크립트를 이용한 복구 작업은 \$CUBRID\_DATABASES를 수정하기 때문에, 스크립트 종료 후 생성 대상 DB가 정상 등록되었는지 확인하는 것이 좋다.
- 더 이상 사용하지 않는 복구된 DBMS는 deletedb를 통해 삭제하기를 권고한다

#### restoredb\_db\_new.sh를 수행한 경로에 log파일 확인

복구 작업을 수행한 경로에는 restoredb\_YYYYMMDDhhmiss.log 형식의 로그 파일이 생성된다. 문제가 발생했을 경우, 이 로그 파일을 통해 복구 진행 내역을 확인할 수 있다.

```
[cubrid ~]$ cat restoredb_20250411185128.log

[Database(db1) Restore (level = 0) start]

- restore start time: Fri Api 11 18:51:28 2025

- restore steps: 1
 step 1) restore using (level = 0) backup data
```



```

- restore progress status (using level = 0 backup data)

volume name | # of pages | restore progress
status | done

dbms_keys | 1 |
| done
dbms_vinf | 1 |
| done
dbms | 32768 |
| done
dbms_lgar001 | 32768 |
| done
dbms_lgar002 | 32768 |
| done
dbms_lginf | 1 |
| done
dbms_lgat | 32768 |
| done

- restore end time: Fri Api 11 18:51:30 2025

[Database(db1) Restore (level = 0) end]

```

## 트러블슈팅

### SQL 로그 확인

#### CAS의 SQL 로그

특정 오류가 발생했을 때 보통 브로커 응용 서버(CAS)의 SQL 로그를 확인한다.

각 CAS 당 하나의 SQL 로그 파일이 생성되는데, CAS 프로세스가 많은 환경에서는 SQL 로그도 많아지므로 오류가 발생한 SQL 로그 파일을 찾기가 어렵다. 하지만, SQL 로그 파일 이름은 뒷부분에 CAS ID를 포함하고 있기 때문에 오류 발생 질의를 수행한 CAS의 ID를 알면 SQL 로그 파일을 찾기 쉽다.

## 참고

CAS의 SQL 로그 파일은 <broker\_name>\_<app\_server\_num>.sql.log으로 저장되는데([브로커 로그](#) 참고), <app\_server\_num>이 CAS ID이다.

## CAS 정보 출력 함수

`cci_get_cas_info()` 함수 또는 JDBC의 `cubrid.jdbc.driver.CUBRIDConnection` 클래스의 `toString()` 메서드는 질의를 수행할 때 해당 질의가 수행된 브로커의 호스트와 CAS ID를 포함한 정보를 출력하여, 이를 통해 해당 CAS의 SQL 로그 파일을 쉽게 찾을 수 있다.

```
<host>:<port>,<cas id>,<cas process id>
예) 127.0.0.1:33000,1,12916
```

## 응용 프로그램 로그

응용 프로그램에서 로그를 출력하도록 설정하면 특정 질의에서 오류가 발생했을 때 해당 오류가 발생한 CAS ID를 확인할 수 있다. JDBC는 `java.util.logging`에서 `cubrid.jdbc`의 레벨을 **FINE**이하로 설정해야 하고, 로그 형식도 응용 프로그램의 `java.util.logging` 설정이 결정한다. 에러가 발생했을 때 작성되는 응용 프로그램 로그의 예는 다음과 같으며, JDBC 예는 [드라이버 로그 설정](#)의 설정(**SimpleFormatter**)을 사용했을 때의 출력이다.

### JDBC 응용 프로그램 로그

```
May 9, 2013 4:25:08 PM cubrid.jdbc.jci.UConnection logException
FINE: DUMP EXCEPTION
[cubrid.jdbc.driver.CUBRIDException]
cubrid.jdbc.driver.CUBRIDException: Syntax: syntax error, unexpected
IdName [CAS INFO - localhost:33000,1,30560],[SESSION-16],[URL-
jdbc:cubrid:localhost:33000:demodb:*****:?
logSlowQueries=true&slowQueryThresholdMillis=5].
```

### CCI 응용 프로그램 로그

```
Syntax: syntax error, unexpected IdName [CAS INFO - 127.0.0.1:33000,
1, 30560].
```

## 슬로우 쿼리

슬로우 쿼리 발생 시, 응용 프로그램 로그와 CAS의 SQL 로그를 통해 슬로우 쿼리의 원인을 파악해야 한다.

CUBRID는 응용 프로그램-브로커-DB 서버의 3 계층 구조로 되어 있기 때문에, 슬로우 쿼리(slow query) 발생 시 원인이 응용 프로그램-브로커 구간에 있는지 브로커-DB 서버 구간에 있는지 파악하기 위해 응용 프로그램 로그 또는 브로커 응용 서버(CAS)의 SQL 로그를 확인해야 한다.

응용 프로그램 로그에는 슬로우 쿼리가 출력되었는데 CAS의 SQL 로그에는 해당 질의가 슬로우 쿼리로 출력되지 않았다면, 응용 프로그램-브로커 사이에서 속도가 저하된 원인이 존재할 것이다.

몇가지 예는 다음과 같다.

- 응용 프로그램-브로커 사이에서 네트워크 통신 속도가 저하되었는지 확인해 본다.
- 브로커 로그( **\$CUBRID/log/broker** 디렉터리 이하에 존재)에 기록되는 정보를 통해 CAS가 재시작된 경우가 있는지 확인한다. CAS 개수가 부족한 것으로 파악되면(단, 슬로우 쿼리가 있으면 슬로우 쿼리 튜닝 후) CAS 개수를 늘리는데, 이를 위해 cubrid\_broker.conf의 **MAX\_NUM\_APPL\_SERVER** 값을 적절히 늘려야 한다. 이와 함께 cubrid.conf의 **max\_clients** 값도 늘리는 것을 고려해야 한다.

응용 프로그램 로그와 CAS의 SQL 로그에 둘 다 슬로우 쿼리로 출력되고 둘 사이에 해당 질의의 수행 시간 차이가 거의 없다면, 브로커-DB 서버 사이에서 속도가 저하된 원인이 존재할 것이다. 한 예로, DB 서버에서 질의를 처리하는데 시간이 걸렸을 것이다.

슬로우 쿼리 발생 시 각 응용 프로그램 로그의 예는 다음과 같다. JDBC 예는 **드라이버 로그 설정**의 설정(**SimpleFormatter**)을 사용했을 때의 출력이다.

#### JDBC 응용 프로그램 로그

```
May 9, 2013 4:25:08 PM cubrid.jdbc.jci.UConnection logSlowQuery
FINEST: SLOW QUERY
[CAS INFO]
localhost:33000, 1, 12916
[TIME]
START: 2013-05-09 16:25:08.775, ELAPSED: 52
[SQL]
SELECT * from db_class a, db_class b
```

#### CCI 응용 프로그램 로그

```
2013-05-10 18:11:23.023 [TID:14346] [DEBUG][CONHANDLE - 0002][CAS
INFO - 127.0.0.1:33000, 1, 12916] [SLOW QUERY - ELAPSED : 45] [SQL -
select * from db_class a, db_class b]
```

응용 프로그램과 브로커의 슬로우 쿼리 정보는 각각 다른 파일로 다음과 같은 경우에 저장된다.

- CCI 응용 프로그램의 슬로우 쿼리 정보는 연결 URL의 **logSlowQueries** 속성을 **yes**로 설정하고 **slowQueryThresholdMillis** 값을 설정하면 **logFile** 속성으로 지정한 응용 프로그램 로그 파일에 저장된다(**cci\_connect\_with\_url()** 참고).

- JDBC 응용 프로그램의 슬로우 쿼리 정보는 연결 URL의 **logSlowQueries** 속성을 **true**로 설정하고 **slowQueryThresholdMillis** 값을 설정한 후, **java.util.logging** 에서 **cubrid.jdbc** 의 레벨을 **FINEST** 로 설정하면 출력된다([드라이버 로그 설정](#) 참고).
- 브로커의 슬로우 쿼리 정보는 [브로커 설정](#) 의 **SLOW\_LOG** 값을 **ON**으로 설정하고 **LONG\_QUERY\_TIME** 값을 설정하면 **\$CUBRID/log/broker/sql\_log** 디렉터리에 저장된다.

## 서버 에러 로그

cubrid.conf에서 **error\_log\_level** 을 설정해서 서버 오류 로그로부터 다양한 정보를 얻을 수 있다. **error\_log\_level** 의 기본값은 **NOTIFICATION** 이다. 이 파라미터를 설정하는 방법은 [오류 메시지 관련 파라미터](#) 를 참고한다.

### 오버플로우 키 또는 오버플로우 페이지 감지

오버플로우 키나 오버플로우 페이지가 발생하면 서버 에러 로그 파일에 **NOTIFICATION** 메시지를 출력한다. 사용자는 이 메시지를 통해 오버플로우 키 또는 오버플로우 페이지로 인해 DB 성능이 느려졌음을 감지할 수 있다. 가능하다면 오버플로우 키나 오버플로우 페이지가 발생하지 않도록 하는 것이 좋다. 즉, 크기가 큰 칼럼에 인덱스를 사용하지 않는 것이 좋으며, 레코드의 크기를 너무 크게 잡지 않는 것이 좋다.

```
Time: 06/14/13 19:23:40.485 - NOTIFICATION *** file ../../src/
storage/btree.c, line 10617 CODE = -1125 Tran = 1, CLIENT =
testhost:csql(24670), EID = 6
Created the overflow key file. INDEX idx(B+tree: 0|131|540) ON CLASS
hoo(CLASS_OID: 0|522|2). key: 'z '(OID: 0|530|1).
.....

Time: 06/14/13 19:23:41.614 - NOTIFICATION *** file ../../src/
storage/btree.c, line 8785 CODE = -1126 Tran = 1, CLIENT =
testhost:csql(24670), EID = 9
Created a new overflow page. INDEX i_foo(B+tree: 0|149|580) ON CLASS
foo(CLASS_OID: 0|522|3). key: 1(OID: 0|572|578).
.....

Time: 06/14/13 19:23:48.636 - NOTIFICATION *** file ../../src/
storage/btree.c, line 5562 CODE = -1127 Tran = 1, CLIENT =
testhost:csql(24670), EID = 42
Deleted an empty overflow page. INDEX i_foo(B+tree: 0|149|580) ON
CLASS foo(CLASS_OID: 0|522|3). key: 1(OID: 0|572|192).
```

### 로그 회복 시간 감지

DB 서버 시작이나 백업 볼륨 복구 시 서버 에러 로그 또는 restoredb 에러 로그 파일에 로그 회복(log recovery) 시작 시간과 종료 시간에 대한 **NOTIFICATION** 메시지를 출력하여, 해당 작업의 소요 시간

을 확인할 수 있다. 해당 메시지에는 적용(redo)해야 할 로그의 개수와 로그 페이지 개수가 함께 기록된다.

```
Time: 06/14/13 21:29:04.059 - NOTIFICATION *** file ../../src/
transaction/log_recovery.c, line 748 CODE = -1128 Tran = -1, EID = 1
Log recovery is started. The number of log records to be applied:
96916. Log page: 343 ~ 5104.
.....
Time: 06/14/13 21:29:05.170 - NOTIFICATION *** file ../../src/
transaction/log_recovery.c, line 843 CODE = -1129 Tran = -1, EID = 4
Log recovery is finished.
```

## 교착 상태 감지

교착 상태 관련 잠금 정보는 서버 오류 로그에 기록된다. 서버 오류 로그 파일은 \$CUBRID/log/server 디렉터리에 <databases>\_<date>\_<time>.err 이름으로 저장된다.

```
demodb_20160202_1811.err

...

Your transaction (index 1, public@testhost|csql(21541)) timed out
waiting on X_LOCK lock on instance 0|650|3 of class t because of
deadlock. You are waiting for user(s) public@testhost|csql(21529) to
finish.

...
```

## HA 상태 변경 감지

HA 상태 변경은 cub\_master 프로세스의 로그 파일에서 확인할 수 있다. 로그 파일은 \$CUBRID/log 디렉터리에 <host\_name>.cub\_master.err 이름으로 저장된다.

## HA split-brain 감지

HA 환경에서 복제 구성된 두 개 이상의 장비 모두 마스터 역할을 맡게 되는 비정상적인 상황이 발생하는 것을 split-brain이라고 한다.

split-brain 상태를 해소하기 위해 스스로 종료하는 마스터 노드의 cub\_master 로그 파일은 다음과 같이 노드 정보를 포함한다.

```
Time: 01/10/25 13:46:34.782 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 873 ERROR CODE = -988, Tran = -1, EID = 35
Node event: [Failback] [Diagnosis] Multiple master nodes
(testhost02, testhost01) are detected.
```

```
Time: 01/10/25 13:46:34.782 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 876 ERROR CODE = -988, Tran = -1, EID = 36
Node event: HA Node Info
```

```
=====
=====
* group_id : hagrps host_name : testhost02 state : master

name priority state score missed

testhost02 2 master -32766 0
testhost01 1 master -32767 0
=====
=====
```

```
Time: 01/10/25 13:46:34.782 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 889 ERROR CODE = -988, Tran = -1, EID = 37
Node event: HA Ping Host Info
```

```
=====
=====
* PING check is enabled

hostname PING check result

8.8.8.8 SUCCESS
=====
=====
```

```
Time: 01/10/25 13:46:34.782 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 1371 ERROR CODE = -988, Tran = -1, EID = 38
Node event: [Failback] [Success] Current node has been successfully
demoted to slave.
```

위의 예는 testhost02 서버가 split-brain을 감지하고 스스로 종료될 때 cub\_master 로그에 출력하는 정보이다.

## Fail-over, Fail-back 감지

Fail-over 혹은 Fail-back이 발생하면 노드는 역할을 변경하게 된다.

Fail-over 후 마스터로 변경되는 노드 혹은 Fail-back 후 슬레이브로 변경되는 노드의 cub\_master 로그 메시지는 [Failover] 또는 [Failback] 접두어로 시작한다.

Fail-over 및 Fail-back 로그 메시지는 진단 로그 메시지와 결과 로그 메시지를 포함한다.

진단 로그 메시지는 fail-over 및 fail-back이 결정되는 경우 기록된다. 이 메시지는 fail-over 또는 fail-back이 시작된 원인을 기록하며, [Diagnosis] 접두어를 포함한다.

결과 로그 메시지는 fail-over 및 fail-back이 취소되거나, 성공적으로 완료될 때 기록된다. Fail-over 또는 Fail-back이 취소된 경우, 결과 메시지는 취소된 이유를 기록하며, [Canceled] 접두어를 포함한다. Fail-over 및 Fail-back이 성공적으로 완료된 경우, 결과 메시지는 결과를 기록하며, [Success] 접두어를 포함한다. 자세한 내용은 [Fail-over 원인 분석](#) 를 참고한다.

```
Time: 01/09/25 16:14:13.005 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 924 ERROR CODE = -988, Tran = -1, EID = 6
Node event: [Failover] [Diagnosis] The master node (testhost01) has
lost its role due to server process problem, such as disk failure.

Time: 01/09/25 16:14:13.614 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 1179 ERROR CODE = -988, Tran = -1, EID = 7
Node event: [Failover] [Success] Current node has been successfully
promoted to master.

Time: 01/09/25 16:14:13.614 - ERROR *** file ../../src/executables/
master_heartbeat.c, line 1196 ERROR CODE = -988, Tran = -1, EID = 8
Node event: HA Node Info
=====
=====
* group_id : hagrps host_name : testhost02 state : master

name priority state score missed
heartbeat

testhost02 2 to-be-master -4094 0
testhost01 1 unknown 32767 1
=====
=====
```

위의 예는 Fail-over로 인해 testhost02 서버가 슬레이브에서 마스터로 역할을 변경하는 도중 cub\_master 로그에 출력하는 정보이다.

## HA 구동 실패

사용자의 개입 없이 복제되는 DB 볼륨의 복구가 불가능한 경우의 예는 다음과 같다.

- copylogdb에서 복사하려는 로그가 원본 노드에서 삭제된 경우
- active 서버에서 반영해야 하는 보관 로그(archive log)가 이미 삭제된 경우
- 서버의 복구에 실패한 경우

이와 같이 복제 볼륨의 자동 복구가 불가능한 경우 “**cubrid heartbeat start**” 명령 수행에 실패하는데, 각각의 경우에 맞게 조치한다.

### 대표적인 복구 불가능 장애

사용자의 개입 없이 자동으로 복제되는 DB 볼륨의 복구가 불가능한 경우 중 서버 프로세스가 원인인 경우는 워낙 다양하므로 설명을 생략한다 **copylogdb** 또는 **applylogdb** 프로세스가 원인인 경우 에러 메시지는 다음과 같다.

#### · **copylogdb** 가 원인인 경우

| 원인                            | 에러 메시지                                                                                                 |
|-------------------------------|--------------------------------------------------------------------------------------------------------|
| 아직 복사되지 않은 로그가 대상 노드에서 이미 삭제됨 | log writer: failed to get log page(s) starting from page id 80.                                        |
| 이전 복사되던 DB와 다른 DB의 로그로 판단됨    | Log <code>"/home1/cubrid/DB/tdb01_cdb037.cub/tdb01_lgat"</code> does not belong to the given database. |

#### · **applylogdb** 가 원인인 경우

| 원인                                                                   | 에러 메시지                                                                                                                                |
|----------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| 복제 반영할 로그가 포함된 archive 로그가 이미 삭제됨                                    | Internal error: unable to find log page 81 in log archives.<br><br>Internal error: logical log page 81 may be corrupted.              |
| _db_ha_apply_info 카탈로그와 현재 복제 로그의 DB 생성 시간이 다름. 즉, 이전 반영하던 복제 로그가 아님 | HA generic: Failed to initialize _db_ha_apply_info.                                                                                   |
| 데이터베이스 로캘이 다름                                                        | Locale initialization: Active log file(/home1/cubrid/DB/tdb01_cdb037.cub/tdb01_lgat) charset is not valid (iso88591), expecting utf8. |



## HA 구동 실패 시 대처 방법

| 상황                          | 대처 방법                                    |
|-----------------------------|------------------------------------------|
| 실패 원인이 된 원본 노드가 마스터 상태인 경우  | 복제 재구성                                   |
| 실패 원인이 된 원본 노드가 슬레이브 상태인 경우 | 복제 로그 및 _db_ha_apply_info 카탈로그 초기화 후 재시작 |

## DDL Audit Log

### 개요

CUBRID는 테이블 생성/삭제/수정 등 데이터베이스 시스템 구성 및 테이블 액세스 권한을 변경하는 DDL(Data Definition Language)을 기록하는 기능을 가지고있다. CAS, csql 및 loaddb를 통해 수행된 DDL은 필요에 따라 실행 된 파일의 복사본과 함께 로그 파일에 기록 될 수 있다.

시스템 파라미터의 ddl\_audit\_log가 yes 이면 \$CUBRID/log/ddl\_audit 디렉토리에 DDL Audit log가 생성된다. 각 로그 파일의 크기는 ddl\_audit\_log\_size 매개 변수에 지정된 값을 초과 할 수 없다. DDL Audit와 관련된 시스템 매개 변수는 CUBRID 운영의 **시스템 설정** 을 참조 한다.

### 참고

**시스템 카탈로그** 의 메서드를 사용하는 경우에는 DDL Audit Log는 생성되지 않는다.

### DDL Audit 로그 파일 이름 규칙

- cas: {broker\_name}\_{app\_server\_num}\_ddl.log
- csql interactive: csql\_{db\_name}\_ddl.log
- loaddb: loaddb\_{db\_name}\_ddl.log

#### 추가 파일 이름 규칙:

- 파일의 csql : csql/{csql\_file}\_{YYYYMMDD}\_{HHMMSS}\_{pid}
- loaddb : loaddb/{loaddb\_file}\_{YYYYMMDD}\_{HHMMSS}\_{pid}

## CAS의 DDL Audit 로그 파일 형식

• [Time] [ip\_addr][db\_name][user\_name][result][elapsed time][auto commit/rollback][sql\_text]

### 설명:

- [Time] : DDL 실행 시작 시간 (예 : 20-12-18 12 : 08 : 32.327)
- [db\_name] : DDL이 실행된 데이터베이스 이름
- [ip\_addr] : 애플리케이션 클라이언트의 IP 주소 (예 : 172.31.0.70)
- [user\_name] : DDL을 발행 한 데이터베이스 사용자 이름
- [result] : 명령문 실행 결과. 성공하면 OK, 그렇지 않으면 오류 코드 (예 : ERROR : -494)
- [elapsed time] : 문 실행 경과 시간
- [auto commit/rollback] : 오류 코드와 함께 자동으로 커밋되거나 롤백된다.
- [sql\_text] : 실행 된 DDL 텍스트

## CSQL의 DDL Audit 로그 파일 형식

• [Time] [pid][user\_name][result][elapsed time][auto commit/rollback][sql\_text]

### 설명:

- [Time] : DDL 실행 시작 시간 (예 : 20-11-20 13 : 26 : 51.765)
- [pid] : csql 프로세스 ID
- [user\_name] : DDL을 발행 한 데이터베이스 사용자 이름
- [result] : 명령문 실행 결과. 성공하면 OK, 그렇지 않으면 오류 코드 (예 : ERROR : -272)
- [elapsed time] : 문 실행 경과 시간
- [auto commit/rollback] : 오류 코드와 함께 자동으로 커밋되거나 롤백된다.
- [sql\_text] : 실행 된 DDL 텍스트 또는 실행 된 csql 파일 이름

## LOADDB의 DDL Audit 로그 형식

• [Time] [pid][user\_name][result][log contents][file name]

**설명:**

- [Time] : DDL 실행 시작 시간 (예 : 20-12-18 12 : 08 : 32.327)
- [pid] : loaddb 프로세스 ID
- [user\_name] : DDL을 발행 한 데이터베이스 사용자 이름
- [result] : loaddb 실행 결과, 성공하면 OK, 그렇지 않으면 오류 코드 (예 : ERROR : -494)
- [log contents] : 총 문의 수 또는 오류 및 오류 행의 경우 커밋 수.
- [file name] : loaddb에서 로드 한 파일을 복사한다. 이때 복사 된 파일의 이름이다.

## 데이터베이스 오류 로그

### CUBRID 오류 메시지 개요

CUBRID 에러 메시지는 \$CUBRID/msg 디렉터리에 한글 및 영문으로 제공된다, CUBRID 사용자(DBA) 관점에서 이러한 오류 메시지는 시스템, 트랜잭션/락, 로그/백업, DDL/DML, 인덱스, 제약조건, 권한, SQL 실행, HA/복제 등 CUBRID 데이터베이스가 런타임 중 직면할 수 있는 주요 오류 상황을 체계적으로 분류한 핵심 에러 카탈로그를 구성한다

CUBRID 오류 메시지 파일별 구성 및 사용자 안내 메시지는 다음과 같다.

- cubrid.msg: CUBRID 데이터베이스 엔진의 런타임 에러 및 시스템 메시지
- csqll.msg: CSQL(CUBRID SQL 인터프리터) 클라이언트 도구의 사용법, 명령어, 에러 메시지
- utils.msg: CUBRID 관리 유틸리티 도구들의 메시지 및 사용법

### CUBRID 데이터 모델 용어 비교

CUBRID는 객체-관계 데이터베이스(ORDB)로 개발되어 클래스(테이블) 상속과 같은 객체 개념을 일부 지원하고 있다, 이로 인해 오류 메시지나 내부 개념에서는 객체(Object) 기반 용어가 사용되기도 한다. 따라서 아래와 같이 전통적인 관계형 데이터베이스(RDB) 용어와 객체-관계 데이터베이스(ORDB) 용어를 함께 이해할 필요가 있다.

| 관계형(RDB) 용어 | 객체-관계형(ORDB) 용어 |
|-------------|-----------------|
| 테이블 (table) | 클래스 (class)     |

| 관계형(RDB) 용어        | 객체-관계형(ORDB) 용어 |
|--------------------|-----------------|
| 칼럼 (column)        | 속성 (attribute)  |
| 레코드 (record)       | 인스턴스 (instance) |
| 데이터 타입 (data type) | 도메인 (domain)    |

### CUBRID 오류 메시지 카테고리

데이터베이스 주요 오류 메시지는 시스템 관리, SQL 질의, 데이터베이스 관리, 데이터베이스 파일, 트랜잭션, CUBRID HA 등 여섯 범주로 분류하였으며, 각 오류 코드와 메시지별로 설명과 서비스 영향도를 함께 정리하였다.

## 시스템 관리 오류

### ERROR CODE: -3, 'Out of virtual memory: unable to allocate %1\$zu memory bytes.'

- 이 메시지는 CUBRID 시스템이 운영체제로부터 특정 크기의 가상 메모리를 할당하려고 시도했지만, 시스템에 사용 가능한 가상 메모리가 부족하여 할당에 실패했음을 나타내는 오류입니다. 가상 메모리는 물리적 RAM과 디스크의 스왑 공간을 포함하며, CUBRID는 데이터베이스 작업(예: 쿼리 실행, 버퍼 캐시, 트랜잭션 관리, 내부 데이터 구조)을 위해 많은 메모리를 사용합니다. 이 메시지는 주로 시스템의 물리적 RAM이 부족하거나, 스왑 공간이 고갈되었거나, CUBRID 프로세스에 할당된 메모리 제한을 초과했을 때 발생하며, 이 영향으로 데이터베이스의 정상적인 서비스를 방해할 수 있습니다.

### ERROR CODE: -7, 'Trying to format disk volume "%1\$s" with an incorrect value %2\$d for number of pages.'

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨을 포맷하려고 시도했지만, 지정된 페이지 수가 유효하지 않거나 허용되지 않는 값이어서 포맷 작업에 실패했음을 나타내는 오류입니다. 주로 file open 단계에서 잘못된 설정, 시스템 제한 초과, 또는 내부 로직 오류로 인해 발생하며, 데이터베이스 생성이나 볼륨 확장 작업을 방해할 수 있습니다.

### ERROR CODE: -8, 'Unable to format disk volume "%1\$s" with %2\$d pages (%3\$lld bytes).'

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨을 지정된 크기로 포맷하려고 시도했지만, 파일 시스템 수준의 문제로 인해 포맷 작업에 실패했음을 나타내는 오류입니다. 주로 file open 단계에서 파일 시스템 오류, 권한 문제, 하드웨어 문제, 또는 볼륨 파일 경로 문제로 인해 발생하며, 데이터베이스 생성이나 볼륨 확장 작업을 방해할 수 있습니다.

**ERROR CODE: -9, 'Unable to format disk volume "%1\$s" with %2\$d pages (%3\$lld Kbytes) due to insufficient space. Current space available is %4\$d pages (%5\$lld Kbytes).'**

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨을 포맷하려고 시도했지만, 요청된 크기보다 사용 가능한 디스크 공간이 부족하여 포맷 작업에 실패했음을 나타내는 오류입니다, 주로 디스크 공간 부족, 볼륨 크기 설정 오류, 또는 파일 시스템 제한으로 인해 발생하며, 데이터베이스 생성이나 볼륨 확장 작업을 방해할 수 있습니다.

**ERROR CODE: -10, 'Unable to mount disk volume "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨을 마운트(mount)하려고 시도했지만, 운영체제 또는 파일 시스템 수준의 문제로 인해 해당 작업에 실패했음을 나타내는 오류입니다, 주로 파일 시스템 손상, 볼륨 파일 부재, 권한 부족, 또는 하드웨어 문제로 인해 발생하며, 데이터베이스의 정상적인 시작이나 복구 작업을 방해할 수 있습니다.

**ERROR CODE: -11, 'Unable to mount disk volume "%1\$s". The database "%2\$s", to which the disk volume belongs, is in use by user %3\$s on process %4\$d of host %5\$s since %6\$s.'**

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨을 마운트(mount)하려고 시도했지만, 해당 볼륨이 속한 데이터베이스가 이미 다른 CUBRID 프로세스에 의해 사용 중이어서 마운트 작업에 실패했음을 나타내는 오류입니다, 주로 lock file를 생성 시 발생하는 오류로 이미 다른 프로세스에 의해 lock file이 생성되었거나 lock file를 생성하지 못하는 경우 발생합니다, 이 메시지는 주로 동일한 데이터베이스에 대해 두 개 이상의 CUBRID 서버 인스턴스가 동시에 접근하려고 할 때 발생하며, 데이터베이스의 일관성을 보호하기 위한 중요한 메커니즘입니다.

**ERROR CODE: -12, 'Problems dismounting volume "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨을 마운트 해제(dismount)하려고 시도했지만, 운영체제 또는 파일 시스템 수준의 문제로 인해 해당 작업에 실패했음을 나타내는 오류입니다, 주로 file close 단계에서 볼륨이 여전히 사용 중이거나, 파일 시스템에 문제가 있거나, CUBRID 프로세스에 볼륨을 마운트 해제할 권한이 없는 경우에 발생하며, 데이터베이스의 정상적인 종료나 기타 작업을 방해할 수 있습니다.

**ERROR CODE: -13, 'An I/O error occurred while reading page %1\$d of volume "%2\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지를 디스크에서 읽으려고 시도했지만, I/O 작업 중 오류가 발생하여 읽기 작업에 실패했음을 나타내는 오류입니다, 주로 디스크 오류, 파일 시스템 문제, 권한 문제, 또는 하드웨어 문제로 인해 발생하며, 데이터베이스의 안정성과 데이터 무결성에 영향을 미칠 수 있습니다.

**ERROR CODE: -14, 'An I/O error occurred while writing page %1\$d of volume "%2\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지를 디스크에 쓰려고 시도했지만, I/O 작업 중 오류가 발생하여 쓰기 작업에 실패했음을 나타내는 오류입니다, 주로 디스크 오류, 파일 시스템 문제, 권한 문제, 또는 하드웨어 문제로 인해 발생하며, 데이터베이스의 안정성과 데이터 무결성에 영향을 미칠 수 있습니다.

**ERROR CODE: -15, 'Unable to write page %1\$d of volume "%2\$s" due to insufficient space in operating system device.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지를 디스크에 쓰려고 시도했지만, 운영체제 수준에서 디스크 공간이 부족하여 쓰기 작업에 실패했음을 나타내는 오류입니다, 주로 디스크 공간 부족, 파일 시스템 제한, 또는 볼륨 크기 제한으로 인해 발생하며, 데이터베이스의 안정성과 데이터 무결성에 영향을 미칠 수 있습니다.

**ERROR CODE: -16, 'Unable to rename disk volume "%1\$s" to "%2\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 디스크 볼륨의 이름을 변경하려고 시도했지만, 파일 시스템 수준의 문제로 인해 해당 작업에 실패했음을 나타내는 오류입니다, 주로 권한 부족, 대상 파일/디렉토리 충돌, 원본 파일 부재, 또는 다른 프로세스에 의해 파일이 사용 중인 경우에 발생하며, 데이터베이스 관리 작업(예: 볼륨 재구성)에 실패할 수 있습니다.

**ERROR CODE: -78, 'Internal error: an I/O error occurred while reading logical log page %1\$lld (physical page %2\$lld) of "%3\$s"'**

- 이 메시지는 CUBRID 시스템이 논리적 로그 페이지를 물리적 페이지에서 읽으려고 할 때, 지정된 운영체제 장치에서 I/O(입출력) 오류가 발생하여 읽기 작업이 실패했음을 나타내는 내부 오류입니다, 이는 주로 하드웨어 문제(디스크 고장), 파일 시스템 손상, 운영체제 수준의 I/O 문제, 또는 드라이버 오류 등으로 인해 발생합니다.

**ERROR CODE: -79, 'Internal error: an I/O error occurred while writing logical log page %1\$lld (physical page %2\$lld) of "%3\$s"'**

- 이 메시지는 CUBRID 시스템이 논리적 로그 페이지를 물리적 페이지에 쓰려고 할 때, 지정된 운영체제 장치에서 I/O(입출력) 오류가 발생하여 쓰기 작업이 실패했음을 나타내는 내부 오류입니다, 이는 주로 하드웨어 문제(디스크 고장), 파일 시스템 손상, 운영체제 수준의 I/O 문제, 또는 드라이버 오류 등으로 인해 발생합니다.

**ERROR CODE: -82, 'Unable to mount log disk volume/file "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 지정된 로그 디스크 볼륨 또는 파일을 "마운트"하거나 접근하는 데 실패했음을 나타내는 오류입니다, 이는 주로 파일 시스템 권한 문제, 파일의 부재 또는 손상, 디스크 공간 부족, 또는 I/O 오류와 같은 하위 수준의 문제로 인해 발생합니다.

**ERROR CODE: -101, 'Unknown user file "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 사용자 파일에 접근하거나 처리하려고 할 때, 해당 파일이 존재하지 않거나, 경로가 잘못되었거나, 시스템에서 인식할 수 없는 파일일 때 발생하는 오류입니다, 주로 `createdb` 명령어 사용 시 `-more-volume-file` 옵션에 기입한 파일을 못 찾는 경우나 볼륨 정보 파일(DBname\_vinf)을 못 찾는 경우 발생합니다.

**ERROR CODE: -102, 'File "%1\$s" does not have enough entries: %2\$d entries expected.'**

- 이 메시지는 CUBRID 시스템이 볼륨정보(DBname\_vinf) 파일을 읽으면서 발생하는 오류입니다, 주로 파일이 불완전하거나 손상되었거나, 데이터베이스 볼륨정보와 다르게 기록/입력되어 있을 때 발생합니다.

**ERROR CODE: -103, 'File "%1\$s" seems to have unordered entries. Entry %2\$d with values: %3\$d %4\$s %5\$s, was found when entry with values %6\$d %7\$s was expected.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨정보(DBname\_vinf) 파일에 첫번째 볼륨을 찾는 과정에서 발생하는 오류입니다, 볼륨정보 파일의 항목들이 순차적으로 정렬되어 있어야 하는데, 순서가 바뀌거나 누락된 항목이 있음을 의미합니다.

**ERROR CODE: -104, 'File "%1\$s" seems to have an incorrect entry for database name (main volume). Entry %2\$d with values: %3\$d %4\$s %5\$s, was found when entry with values %6\$d %7\$s %8\$s was expected.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨정보(DBname\_vinf) 파일에 첫번째 볼륨을 찾는 과정에서 발생하는 오류입니다, 이는 데이터베이스의 무결성 문제, 파일 손상, 또는 수동으로 파일이 변경되었을 가능성이 있습니다.

**ERROR CODE: -105, 'Cannot access backup file "%1\$s". Restore or Backup is cancelled.'**

- 이 메시지는 CUBRID 시스템이 백업 파일에 접근하려고 할 때, 파일 시스템 권한 문제, 파일 손상, 또는 파일이 존재하지 않아 접근할 수 없을 때 발생하는 오류입니다, 백업 또는 복원 작업이 이 파일 접근 실패로 인해 취소되었음을 의미합니다.

**ERROR CODE: -111, 'Your transaction has been aborted by the system due to server failure or mode change.'**

- 이 메시지는 CUBRID 시스템이 서버 장애나 모드 변경으로 인해 활성 트랜잭션을 일방적으로 중단했을 때 발생하는 오류입니다, 클라이언트의 요청이나 동의 없이 시스템이 트랜잭션을 강제로 종료했음을 의미하며, 주로 서버 장애나 시스템 모드 변경 상황에서 발생할 수 있습니다.

**ERROR CODE: -113, 'Unable to restart/initialize the database server. %1\$s'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 서버를 재시작하거나 초기화하려고 할 때 실패했을 때 발생하는 오류입니다, 서버 재시작이나 초기화 과정에서 시스템 리소스 부족, 권한 문제, 포트 충돌, 또는 기타 시스템 오류로 인해 실패했음을 의미합니다.

**ERROR CODE: -114, 'Unable to interpret "%1\$s" as a database. The database volumes/files may have been renamed/copied outside the database domain.'**

- 이 메시지는 CUBRID 시스템이 지정된 경로나 이름을 데이터베이스로 인식할 수 없을 때 발생하는 오류입니다, 데이터베이스 볼륨이나 파일이 데이터베이스 도메인 외부에서 이름이 변경되거나 복사되어, CUBRID가 올바른 데이터베이스 구조를 찾을 수 없음을 의미합니다.

**ERROR CODE: -118, 'Failed to locate current working directory.'**

- 이 메시지는 CUBRID 시스템이 현재 작업 디렉터리를 확인하는 과정에서 실패했을 때 발생하는 오류입니다, 주로 작업 디렉터리 확인 실패는 파일 시스템 권한 문제, 디렉터리 삭제, 또는 프로세스 권한 문제로 인해 발생할 수 있습니다.

**ERROR CODE: -123, 'Unable to create %1\$s for database "%2\$s". Please refer "%3\$s" for additional information.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스에 필요한 볼륨이나 파일을 생성하려고 할 때, 파일 시스템 권한, 디스크 공간 부족, 경로 문제 등으로 인해 생성에 실패했을 때 발생하는 오류입니다, 데이터베이스 볼륨 생성 실패로 인해 데이터베이스 초기화나 확장 작업이 중단되었음을 의미합니다.

**ERROR CODE: -130, 'Out of virtual memory.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 필요한 메모리를 할당할 수 없을 때 발생하는 오류입니다, 시스템의 가상 메모리가 부족하거나, 메모리 할당 요청이 실패했을 때 출력되며, 이는 시스템의 메모리 자원이 고갈되었음을 의미로 해당 오류는 주로 xasl 관련 루틴에서 발생합니다.

**ERROR CODE: -313, 'Object buffer underflow while reading.'**

- 이 메시지는 데이터베이스 객체 데이터를 읽어 오는 과정에서 발생합니다, 주로 객체 로딩이나 언로딩 과정에서 발생하는 중요한 메모리 관련 오류입니다.

**ERROR CODE: -320, 'Encountered corrupted disk representation of object.'**

- 이 메시지는 CUBRID 데이터베이스에서 디스크에 저장된 객체의 표현이 손상되었을 때 나타나는 오류입니다, B-tree 로딩 과정에서 객체 데이터를 읽어올 때 발생합니다, 이 메시지는 디스크 I/O 오류, 메모리 손상, 또는 데이터베이스 파일 손상으로 인해 발생할 수 있습니다

**ERROR CODE: -542, 'Number of free sectors for volume "%1\$s" is inconsistent. %2\$d and %3\$d were found according to volume header and bitmap, respectively.'**

- 이 메시지는 CUBRID 데이터베이스의 특정 볼륨(디스크 파일)에서, 볼륨 헤더(메타데이터)에 기록된 남은 섹터(Free Sector) 개수와 실제 비트맵(섹터 할당 상태를 나타내는 내부 테이블)에서 계산한 남은 섹터 개수가 서로 다를 때 발생합니다, 이런 불일치는 디스크 공간 관리의 일관성 오류를 의미하며, 데이터베이스의 저장 공간 할당/해제 과정에서 문제가 발생했거나, 물리적 손상, 비정상 종료, 버그 등으로 인해 메타데이터와 실제 상태가 어긋난 경우에 나타납니다.

**ERROR CODE: -543, 'Disk header for volume "%1\$s" is inconsistent.'**

- 이 메시지는 CUBRID 데이터베이스의 특정 볼륨(디스크 파일)에서, 디스크 헤더(메타데이터, 볼륨의 구조와 상태를 나타내는 정보)가 손상되었거나 내부적으로 기대하는 값과 실제 값이 다를 때 발생합니다, 디스크 헤더는 볼륨의 크기, 할당 정보, 버전, 체크섬 등 중요한 메타데이터를 포함하고 있어, 이 정보가 일치하지 않으면 데이터베이스의 저장 구조 전체에 영향을 줄 수 있습니다, 주로 비정상 종료, 디스크 오류, 파일 시스템 손상, 버그 등으로 인해 발생할 수 있습니다.



**ERROR CODE: -599, 'An I/O error occurred while synchronizing state of volume "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스가 볼륨의 상태를 디스크와 동기화하는 과정에서 I/O 오류가 발생했을 때 나타납니다, 볼륨은 CUBRID 데이터베이스의 물리적 저장 단위로, 데이터 파일을 의미합니다, 동기화는 메모리에 있는 데이터를 디스크에 강제로 쓰는 과정으로, 데이터 일관성과 내구성을 보장하기 위해 수행됩니다.

**ERROR CODE: -705, 'Inconsistent %1\$d permanent volumes were found when %2\$d were expected.'**

- 이 메시지는 CUBRID 데이터베이스가 부팅되거나 복구되는 과정에서, 부트 페이지(boot page)에 기록된 영구 볼륨의 개수와 실제로 시스템에서 발견된 영구 볼륨의 개수가 일치하지 않을 때 나타나는 오류입니다, 이는 데이터베이스의 볼륨 구성에 문제가 있음을 의미하며, 데이터베이스의 일관성 및 무결성에 영향을 미칠 수 있습니다, 이 메시지는 CUBRID 데이터베이스 서버가 시작되거나 크래시 후 복구될 때 발생합니다

**ERROR CODE: -708, 'Unable to expand temporary disk volume "%1\$s" with %2\$d pages (%3\$lld Kbytes) due to insufficient space. Current space available is %4\$d pages (%5\$lld Kbytes).'**

- 이 메시지는 CUBRID 데이터베이스가 임시 디스크 볼륨을 확장하려고 시도했지만, 디스크 공간이 부족하여 실패했을 때 나타나는 오류입니다, CUBRID는 대용량 쿼리 처리나 임시 테이블 생성 시 임시 볼륨을 확장하는 과정에서 발생할 수 있습니다.

**ERROR CODE: -780, 'Internal system failure: unable to get system time.'**

- 이 메시지는 CUBRID 데이터베이스에서 시스템 시간을 가져오려고 할 때, 시스템 시간을 얻을 수 없는 상황임을 나타냅니다, 이는 CUBRID의 시스템 시간 메커니즘에서 발생하며, 주로 운영체제 수준의 시간 관련 함수 호출 실패로 인해 발생할 수 있습니다.

**ERROR CODE: -792, 'Cannot allocate memory for css.'**

- 이 메시지는 CUBRID 시스템의 connection entry 관련 메모리 할당에 실패했을 때 발생하는 내부 오류입니다, 메모리 할당 실패는 모듈이 정상적으로 작동할 수 없게 되며, 이는 클라이언트 연결 관리와 서버 통신에 치명적인 영향을 미칩니다, 이는 주로 시스템 메모리 부족, 메모리 단편화, 또는 운영체제의 메모리 할당 제한으로 인해 발생합니다.

**ERROR CODE: -864, 'Cannot open backupdb/restoredb verbose file "%1\$s"'**

이 오류는 데이터베이스 백업 또는 복원 작업 중 상세 로그 파일을 생성하거나 열지 못할 때 발생합니다, 파일 시스템 권한, 디스크 공간 부족, 경로 문제 등으로 인해 발생할 수 있는 I/O 관련 오류입니다.

**ERROR CODE: -879, 'Cannot get permission: "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 파일 또는 디렉터리의 파일 시스템 권한 정보를 조회하려고 할 때 발생합니다, 이런 상황은 일반적으로 데이터베이스 파일이나 로그 파일의 접근 권한

을 확인해야 할 때 발생하며, CUBRID 프로세스를 실행하는 사용자 계정에 해당 파일에 대한 정보 조회 권한이 없을 때 주로 나타납니다.

#### **ERROR CODE: -880, 'Cannot change permission '%1\$s''**

- 이 메시지는 CUBRID 데이터베이스에서 특정 파일 또는 디렉터리의 파일 시스템 권한을 변경하려고 할 때 발생합니다. 이런 상황은 일반적으로 데이터베이스 파일이나 로그 파일의 접근 권한을 조정해야 할 때 발생하며, CUBRID 프로세스를 실행하는 사용자 계정에 해당 파일에 대한 변경 권한이 없을 때 주로 나타납니다.

#### **ERROR CODE: -881, 'Cannot get lock file '%1\$s(FILE ID:%2\$d)''**

- 이 메시지는 CUBRID 데이터베이스에서 특정 파일에 대한 락(lock)을 획득하려고 할 때 발생합니다. 이런 상황은 일반적으로 데이터베이스 볼륨을 열거나 마운트할 때 발생하며, 파일 시스템의 락 관리 메커니즘과 관련이 있습니다. 이는 파일 시스템 접근 문제, 권한 문제, 다른 프로세스에 의한 락 점유, 또는 시스템 리소스 부족 등으로 인해 발생할 수 있습니다.

#### **ERROR CODE: -882, 'Cannot release lock file 'FILE ID:%1\$d''**

- 이 메시지는 CUBRID 데이터베이스에서 특정 파일에 설정된 락(lock)을 해제하려고 할 때 발생합니다. 이런 상황은 일반적으로 데이터베이스 볼륨을 닫거나 언마운트할 때 발생하며, 파일 시스템의 락 관리 메커니즘과 관련이 있습니다. 이는 파일 시스템 접근 문제, 권한 문제, 또는 시스템 리소스 부족 등으로 인해 발생할 수 있습니다.

#### **ERROR CODE: -980, 'A thread is waiting in %1\$s more than %2\$d msec(%3\$d msec)'**

- 이 메시지는 CUBRID 데이터베이스에서 스레드가 특정 작업(임계 영역 진입, 잠금 획득, 파일 I/O 등)을 수행하는 동안 설정된 임계값보다 오래 대기할 때 발생하는 경고 메시지입니다. 이는 성능 모니터링 기능의 일부로, 스레드가 예상보다 오래 대기하는 상황을 감지하여 시스템 관리자에게 알려주는 역할을 합니다. 이 메시지는 오류가 아니라 성능 문제를 조기에 발견하고 대응할 수 있도록 도와주는 정보성 메시지입니다. 주로 임계 영역(critical section), 잠금 관리, 파일 I/O 작업 등에서 발생하며, 시스템의 병목 현상이나 리소스 경합 상황을 파악하는 데 도움이 됩니다.

#### **ERROR CODE: -1015, 'Specified directory '%1\$s' does not exist.'**

- 이 메시지는 데이터베이스 서버가 사용하는 볼륨파일들의 경로가 맞지 않거나 생성할 수 없는 경우 발생합니다.

#### **ERROR CODE: -1113, 'Element index out of range.'**

- 이 메시지는 CUBRID 데이터베이스에서 이진 힙(Binary Heap) 데이터 구조에 요소를 추가하려고 할 때, 힙의 최대 용량을 초과했을 때 발생합니다. 즉, 이진 힙이 이미 가득 찬 상태에서 새로운 요소를 삽입하려고 시도했음을 의미합니다. 이는 메모리 관리와 시스템 안정성을 보장하기 위한 보호성 오류입니다. 이진 힙은 정렬, 우선순위 큐, 힙 정렬 등에서 사용되는 중요한 데이터 구조로, 용량 제한을 초과하면 메모리 오버플로우나 예상치 못한 동작이 발생할 수 있습니다.

**ERROR CODE: -1185, 'Unexpected page refix on page %1\$d,%2\$d, while trying to fix page %3\$d, %4\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 페이지 버퍼 관리 시스템에서 발생합니다. 특정 페이지를 메모리에 고정(fix)하여 접근하려 할 때, 이미 다른 페이지가 예상치 못한 방식으로 재고정(refix)되는 상황이 발생했음을 나타냅니다. 이는 페이지 버퍼의 내부적인 일관성 또는 상태 관리에 문제가 생겼음을 의미하며, 일반적으로는 발생해서는 안 되는 비정상적인 상황입니다. 페이지 버퍼는 데이터베이스의 핵심 구성 요소이므로, 이러한 오류는 데이터 무결성 문제나 시스템 불안정으로 이어질 수 있습니다.

## 질의(SQL) 관련 오류

**ERRORCODE:-64 , 'Unknown class "%1\$s"'**

- 이 메시지는 CUBRID 시스템에서 지정된 테이블 이름을 찾을 수 없을 때 발생하는 오류입니다.

**ERROR CODE: -65, 'Class "%1\$s" already exists.'**

- 이 메시지는 CUBRID 시스템에서 이미 존재하는 테이블 이름으로 새로운 테이블을 생성하려고 시도했을 때 발생하는 오류입니다.

**ERROR CODE: -176, 'Conversion error in date format.'**

- 이 메시지는 CUBRID에서 날짜/시간 관련 데이터 타입의 형식 변환 과정에서 오류가 발생했을 때 나타나는 오류입니다, 문자열을 날짜/시간 타입으로 변환하거나, 날짜/시간 값의 유효성을 검증하는 과정에서 발생합니다, 잘못된 날짜 형식, 범위를 벗어난 값, 또는 변환할 수 없는 형식의 데이터로 인해 발생할 수 있습니다.

**ERROR CODE: -181, 'Cannot coerce value of domain "%1\$s" to domain "%2\$s"'**

- 이 메시지는 CUBRID에서 한 데이터 타입의 값을 다른 데이터 타입으로 변환할 때, 변환이 불가능하거나 호환되지 않을 때 발생하는 오류입니다, 데이터 타입 간의 호환성 문제나 변환 규칙에 맞지 않는 값으로 인해 발생합니다, 예를 들어, 문자열을 숫자로 변환하거나, 서로 다른 데이터 타입 간의 연산을 수행할 때 발생할 수 있습니다.

**ERROR CODE: -182, 'Cannot coerce value of domain "%1\$s" to domain "%2\$s" without overflow.'**

- 이 메시지는 CUBRID에서 한 데이터 타입의 값을 다른 데이터 타입으로 변환할 때, 오버플로우가 발생하여 변환이 불가능할 때 나타나는 오류입니다, 데이터 타입 변환 과정에서 값의 범위가 목표 데이터 타입의 허용 범위를 초과하여 변환할 수 없음을 의미합니다, 예를 들어, 매우 큰 정수 값을 작은 정수 타입으로 변환하려고 할 때 발생할 수 있습니다.

**ERROR CODE: -202, 'Attribute "%1\$s" was not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 지정된 이름의 컬럼을 찾을 수 없을 때 발생하는 오류입니다, 주로 잘못된 컬럼 이름 참조, 존재하지 않는 컬럼 접근, 또는 스키마 정의 오류로 인해 나타날 수 있습니다.

**ERROR CODE: -203, 'Value for attribute "%1\$s" must be of type "%2\$s", not type "%3\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 컬럼의 데이터 타입과 할당하려는 값의 타입이 일치하지 않을 때 발생하는 오류입니다, 주로 잘못된 타입의 값을 할당하려고 할 때, 데이터 타입 정의 또는 데이터 타입 변환 실패로 인해 나타날 수 있습니다.

**ERROR CODE: -205, 'Attribute "%1\$s" cannot be made NULL.'**

- 이 메시지는 CUBRID 데이터베이스에서 NOT NULL 제약 조건이 설정된 컬럼에 NULL 값을 할당하려고 할 때 발생하는 오류입니다, 주로 잘못된 데이터 삽입, 업데이트 시 NULL 값 할당, 또는 스키마 정의와 데이터 간의 불일치로 인해 나타날 수 있습니다.

**ERROR CODE: -208, 'Method "%1\$s" not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 지정된 이름의 메소드를 찾을 수 없을 때 발생하는 오류입니다, 주로 잘못된 메소드 이름 참조, 존재하지 않는 메소드 접근, 또는 스키마 정의 오류로 인해 나타날 수 있습니다.

**ERROR CODE: -218, 'Invalid object in db\_get path expression.'**

- 이 메시지는 CUBRID 데이터베이스에서 *db\_get* 함수의 경로 표현식(path expression) 처리 중 유효하지 않은 객체가 발견되었을 때 발생하는 오류입니다.

**ERROR CODE: -219, 'Invalid db\_get path expression.'**

- 이 메시지는 CUBRID 데이터베이스에서 *db\_get* 함수의 경로 표현식(path expression)이 유효하지 않을 때 발생하는 오류입니다.

**ERROR CODE: -220, 'Invalid set in db\_get path expression.'**

- 이 메시지는 CUBRID 데이터베이스에서 *db\_get* 함수의 경로 표현식(path expression)을 처리하는 도중, set 타입이 아닌 객체에 대해 set 관련 연산이 시도되었을 때 발생하는 오류입니다.

**ERROR CODE: -221, 'Invalid set index in db\_get path expression.'**

- 이 메시지는 CUBRID 데이터베이스에서 *db\_get* 함수의 경로 표현식(path expression)을 처리하는 도중, set, multiset, 또는 sequence 타입의 컬렉션 객체에 대해 유효하지 않은 인덱스가 사용되었을 때 발생하는 오류입니다.

**ERROR CODE: -222, 'Attempt to assign string value greater than %2\$d characters to attribute "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 객체의 특정 컬럼에 문자열 값을 할당하려 할 때, 할당하려는 문자열의 길이가 해당 컬럼이 허용하는 최대 문자열 길이를 초과하여 발생하는 오류입니다, 주로 VARCHAR 또는 CHAR 타입의 컬럼에 대해 정의된 최대 길이를 넘어서는 데이터를 삽입하거나 업데이트할 때 발생합니다.

**ERROR CODE: -225, 'Missing value for attribute "%1\$s" with the NOT NULL constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 NOT NULL 제약 조건이 설정된 컬럼에 대해 값이 제공되지 않았거나 NULL 값이 할당되려고 할 때 발생하는 오류입니다.

**ERROR CODE: -226, 'Empty component list in class.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블 또는 객체에 대한 작업을 수행하는 도중, 해당 테이블이나 객체가 필수적으로 가져야 할 컴포넌트(컬럼, 속성 등) 목록이 비어 있거나 유효하지 않을 때 발생하는 오류입니다, 주로 스키마 언로드, 쿼리 결과 처리, 또는 특정 객체 정보 조회와 같은 작업에서 발생할 수 있습니다.

**ERROR CODE: -227, 'Attempt to assign value to attribute "%1\$s" more than once.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체 템플릿을 사용하여 객체를 생성할 때, 동일한 컬럼에 값을 두 번 이상 할당하려고 할 때 발생하는 오류입니다.

**ERROR CODE: -228, 'Method "%1\$s" called with %2\$d arguments. The system allows a maximum of %3\$d arguments.'**

- 이 메시지는 CUBRID 데이터베이스에서 메소드를 호출할 때, 허용된 최대 인자 개수를 초과하는 인자를 전달했을 때 발생하는 오류입니다.

**ERROR CODE: -229, 'Method "%1\$s" requires argument %2\$d to be within the domain "%3\$s". An invalid value of domain "%4\$s" was given.'**

- 이 메시지는 CUBRID 데이터베이스에서 메소드를 호출할 때, 특정 인자의 값이 예상된 도메인을 벗어나는 경우 발생하는 오류입니다, 주로 잘못된 타입의 값을 메소드 인자로 전달했거나, 데이터 타입 변환(coercion)이 불가능할 때 나타날 수 있습니다.

**ERROR CODE: -233, 'Cannot create class with basic type name "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블을 생성할 때 기본 타입 이름을 테이블 이름으로 사용하려고 할 때 발생하는 스키마 정의 오류입니다, CUBRID는 기본 타입(primitive type)과 예약어(reserved word)를 테이블 이름으로 사용하는 것을 금지합니다. 이는 타입 시스템의 일관성을 유지하고 이름 충돌을 방지하기 위함입니다, 이는 잘못된 테이블 이름을 사용했거나, 기본 타입이나 예약어를 테이블 이름으로 사용하려고 할 때 나타날 수 있습니다.

**ERROR CODE: -234, 'Method file "%1\$s" not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 스키마 템플릿에서 특정 이름을 가진 메소드 파일을 찾을 수 없을 때 발생하는 스키마 정의 오류입니다, 잘못된 메소드 파일 이름을 사용했거나, 해당 메소드 파일이 이미 삭제되었거나, 스키마 템플릿이 올바르게 초기화되지 않았을 때 나타날 수 있습니다.

**ERROR CODE: -235, '%1\$d unresolved method functions after linking.'**

- 이 메시지는 CUBRID 데이터베이스에서 동적 링킹 과정이 완료된 후에도 여전히 해결되지 않은(구현을 찾을 수 없는) 메소드 함수가 존재함을 나타내는 스키마 정의 오류입니다, 메소드 파일에 문제가 있거나, 함수 이름이 잘못되었거나, 필요한 라이브러리가 로드되지 않았거나, 메소드 파일 내에서 참조하는 함수가 누락되었을 때 나타날 수 있습니다.

**ERROR CODE: -236, 'Unresolved method "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 동적 링킹 과정에서 특정 메소드(주로 Java 저장 함수/프로시저)의 함수 구현을 찾을 수 없을 때 발생하는 스키마 정의 오류입니다, 메소드 파일이 손상되었거나, 함수 이름이 잘못되었거나, 링커 설정에 문제가 있을 때 나타날 수 있습니다.

**ERROR CODE: -239, 'Attribute "%1\$s" was not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 이름을 가진 컬럼을 찾을 수 없을 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -240, 'Method "%1\$s" was not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 이름을 가진 메소드를 찾을 수 없을 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -241, 'There is no attribute or method with name "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 이름을 가진 컬럼나 메소드를 찾을 수 없을 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -242, 'Signature for %2\$s not found for method "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 메소드에 대해 지정된 시그니처를 찾을 수 없을 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -243, 'Argument %2\$d of method "%1\$s" is undefined.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 메소드의 지정된 인덱스에 해당하는 인자가 정의되지 않았을 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -244, 'Domain name "%1\$s" is not the name of a class or basic type.'**

- 이 메시지는 CUBRID 데이터베이스에서 컬럼이나 메소드의 데이터 타입으로 지정된 이름이 유효한 테이블 이름이나 기본 데이터 타입이 아닐 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -245, 'The name "%1\$s" has already been used as an attribute name.'**

- 이 메시지는 CUBRID 데이터베이스에서 새로운 메소드, 테이블 메소드, 또는 다른 스키마 구성 요소의 이름을 정의하려고 할 때, 해당 이름이 이미 기존 컬럼의 이름으로 사용되고 있어서 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -246, 'The name "%1\$s" has already been used as a method name.'**

- 이 메시지는 CUBRID 데이터베이스에서 새로운 컬럼, 테이블 컬럼, 또는 다른 스키마 구성 요소의 이름을 정의하려고 할 때, 해당 이름이 이미 기존 메소드의 이름으로 사용되고 있어서 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -248, 'The attribute domain "%1\$s" is not suitable for use with the %2\$s constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 컬럼에 대해 UNIQUE 제약 조건을 적용하려고 할 때, 해당 데이터 타입이 UNIQUE 제약 조건에 적합하지 않아서 발생하는 스키마 정의 오류입니다, CUBRID는 UNIQUE 제약 조건을 특정 데이터 타입에만 적용할 수 있도록 제한하고 있습니다. 인덱싱이 가능한 데이터 타입만이 UNIQUE 제약 조건을 가질 수 있습니다.

**ERROR CODE: -250, 'Cannot create index on attribute "%1\$s", defined with a shared value.'**

- 이 메시지는 CUBRID 데이터베이스에서 공유 값(shared value)으로 정의된 컬럼에 대해 인덱스를 생성하려고 할 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -251, 'Signature %2\$s already exists for method "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 메소드에 대해 동일한 시그니처를 가진 정의가 이미 존재할 때 발생하는 스키마 정의 오류입니다.

**ERROR CODE: -252, 'Attribute "%1\$s" must have a set domain for that operation.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 컬럼에 대해 SET 타입(SET, MULTISSET 등 컬렉션 타입)이 필요한 작업을 수행하려고 할 때, 해당 컬럼의 데이터 타입이 SET 타입이 아니어서 발생하는 스키마 정의 오류입니다, 주로 `CREATE TABLE` 또는 `ALTER TABLE` 문을 사용하여 컬럼을 정의하거나 수정할 때, 또는 특정 메소드나 함수가 SET 타입 컬럼을 인자로 요구할 때 발생할 수 있습니다.

**ERROR CODE: -253, 'Attribute "%1\$s" cannot be defined with a nested set domain.'**

- 이 메시지는 CUBRID 데이터베이스에서 컬럼을 중첩된 SET 데이터 타입으로 정의하려고 할 때 발생하는 스키마 정의 오류입니다, CUBRID는 SET 타입의 컬렉션(예: SET, MULTISSET) 내부에 또 다른 SET 타입을 중첩하는 것을 허용하지 않습니다, 주로 `CREATE TABLE` 또는 `ALTER TABLE` 문을 사용하여 컬럼을 정의하거나 수정할 때 발생합니다.

**ERROR CODE: -254, 'Unable to remove domain "%2\$s" from attribute "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 데이터 타입을 제거하려고 할 때 발생하는 오류입니다, 주로 `DROP TABLE` 문을 사용하여 컬럼의 데이터 타입을 제거하려고 할 때, 해당 데이터 타입이



컬럼에서 제거될 수 없는 상황(예: 데이터 타입이 존재하지 않거나, 다른 제약 조건에 의해 참조되거나, 컬럼의 기본 데이터 타입인 경우)에서 발생합니다.

**ERROR CODE: -260, 'Argument %2\$d of method "%1\$s" is not of a set domain.'**

- 이 메시지는 CUBRID 데이터베이스에서 메소드 정의 시 특정 인자의 도메인이 set 타입이 아닐 때 발생하는 스키마 정의 오류입니다, CUBRID에서 set 타입은 set, multiset 등의 컬렉션 타입을 의미합니다, 주로 *CREATE TABLE* 또는 *ALTER TABLE* 문을 사용하여 메소드를 정의하거나 수정할 때 발생할 수 있습니다.

**ERROR CODE: -269, 'Cannot define index on domain "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 데이터 타입에 인덱스를 생성하려고 할 때 발생하는 스키마 정의 오류입니다, CUBRID는 모든 데이터 타입에 대해 인덱스 생성을 허용하지 않으며, 인덱스 생성이 가능한 데이터 타입("SET", "MULTISET", "BLOB", "CLOB" 등)은 제한되어 있습니다.

**ERROR CODE: -272, 'Index "%2\$s" already defined for class "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 동일한 이름의 인덱스를 중복으로 생성하려고 할 때 발생하는 스키마 관리 오류를 나타냅니다, 특정 테이블에 이미 존재하는 인덱스와 동일한 이름으로 새로운 인덱스를 생성하려고 시도했을 때 발생합니다.

**ERROR CODE: -273, 'Index "%1\$s" does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 인덱스를 찾을 수 없을 때 발생하는 스키마 관리 오류를 나타냅니다, 이는 주로 *DROP INDEX*, *ALTER INDEX*, 또는 인덱스를 사용하는 쿼리 실행 등 인덱스 DDL 작업 중에 발생할 수 있습니다.

**ERROR CODE: -274, 'Illegal characters in name: "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 스키마 객체(테이블, 컬럼, 메소드, 인덱스 등)의 이름을 정의하거나 변경할 때, 해당 이름이 NULL 또는 empty string 인 경우 발생합니다, 이는 주로 *CREATE TABLE*, *ALTER TABLE*, *CREATE VIEW* 등의 DDL(Data Definition Language) 문을 실행할 때 발생합니다.

**ERROR CODE: -293, 'Operation not allowed for this class type.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 테이블 타입에 대해 허용되지 않는 작업을 시도했을 때 발생하는 오류입니다, 주로 스키마 정의, 테이블 변경, 또는 특정 객체 관련 작업을 수행하는 과정에서 발생하며, 해당 테이블의 특성을 이해하고 허용된 작업을 수행해야 함을 나타냅니다.

**ERROR CODE: -294, 'Undefined environment variable "%1\$s" referenced in method file.'**

- 이 메시지는 CUBRID 데이터베이스에서 메소드 파일의 경로에 포함된 환경 변수가 정의되지 않았을 때 발생하는 오류입니다, 메소드 파일은 CUBRID에서 사용자 정의 메소드(함수)를 구현한 외부 라이브러리 파일로 이 메시지는 주로 동적 링킹 과정에서 메소드 파일을 찾을 수 없어서 발생하는 문제입니다.



**ERROR CODE: -305, 'Sequence index %1\$d is out of bounds.'**

- 이 메시지는 CUBRID 데이터베이스에서 시퀀스(sequence) 컬렉션의 인덱스가 유효한 범위를 벗어났을 때 나타나는 오류입니다, 시퀀스 객체 관리 시스템에서 인덱스 범위 검증 과정에서 발생하며 주로 시퀀스 요소에 접근할 때 인덱스가 0보다 작거나 시퀀스 크기보다 큰 경우입니다.

**ERROR CODE: -308, 'Element value is not within the domain of the set.'**

- 이 메시지는 CUBRID 데이터베이스에서 set(집합) 타입의 컬렉션에 요소를 추가할 때, 해당 요소의 값이 set의 도메인과 호환되지 않을 때 나타나는 오류입니다, 주로 요소 값이 set의 도메인 규칙을 위반할 때 발생합니다.

**ERROR CODE: -309, 'Illegal set element index given: %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 set(집합) 타입의 컬렉션에 잘못된 인덱스가 전달되었을 때 나타나는 오류입니다, 주로 음수 인덱스나 컬렉션 크기를 초과하는 인덱스가 전달될 때 발생합니다.

**ERROR CODE: -311, 'Element not found in sequence.'**

- 이 메시지는 CUBRID 데이터베이스에서 sequence(순서가 있는 컬렉션)에서 특정 요소를 찾을 수 없을 때 나타나는 오류입니다, sequence는 순서가 있는 데이터 구조로 이 메시지는 주로 요소 검색 시 해당 값이 sequence에 존재하지 않을 때 발생합니다.

**ERROR CODE: -312, "'%1\$s' is not a valid set domain.'**

- 이 메시지는 CUBRID 데이터베이스에서 set(집합) 타입이 유효하지 않을 때 나타나는 오류입니다, 주로 잘못된 set(집합) 타입이 전달되거나 NULL인 경우 발생할 수 있습니다.

**ERROR CODE: -424, 'No statement to execute.'**

- 이 메시지는 CUBRID 데이터베이스에서 실행할 SQL 문장이 없을 때 나타나는 오류입니다, 문장 검증 과정에서 세션에 문장이 없거나 문장 배열이 비어있을 때입니다.

**ERROR CODE: -427, 'Data overflow on data type "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 데이터 타입에 대해 오버플로우가 발생했을 때 나타나는 오류입니다, 데이터 타입 변환 과정에서 데이터 값이 해당 데이터 타입의 허용 범위를 초과했을 때입니다.

**ERROR CODE: -428, 'Statement is not updatable.'**

- 이 메시지는 CUBRID 데이터베이스에서 수정할 수 없는 문장(statement)에 대해 수정 작업을 시도했을 때 나타나는 오류입니다, 문장의 수정 가능성 검사 과정에서 뷰테이블에서 수정할 수 없는 문장에 대해 수정을 시도할 때입니다.

**ERROR CODE: -435, 'Multiple statements not allowed.'**

- 이 메시지는 CUBRID 데이터베이스에서 여러 개의 SQL 문장이 동시에 실행되거나 처리될 수 없는 컨텍스트에서 여러 문장이 제공되었을 때 나타나는 오류입니다, SQL 문장 처리 과정에서 단일 문장만 허용되는 API 호출이나 실행 환경에서 세미콜론 등으로 구분된 여러 문장이 입력된 경우입니다.

**ERROR CODE: -454, 'Invalid data type referenced.'**

- 이 메시지는 CUBRID 데이터베이스에서 존재하지 않거나 잘못된 데이터 타입이 참조되었을 때 나타나는 오류입니다, 쿼리에서 데이터 타입 검증 과정에서 발생합니다, 즉, 정의되지 않은 데이터 타입이나 잘못된 데이터 타입 식별자가 사용된 경우입니다.

**ERROR CODE: -457, 'Invalid query result type.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 결과 구조체의 타입이 유효하지 않을 때 나타나는 오류입니다, 쿼리 결과 타입 검증 과정에서 발생합니다, 즉, 지원되지 않는 쿼리 결과 타입이 사용된 경우입니다.

**ERROR CODE: -458, 'Overflow occurred in addition context.'**

- 이 메시지는 CUBRID 데이터베이스에서 산술 연산 중 덧셈을 수행할 때 오버플로우가 발생했을 때 나타나는 오류입니다, 쿼리에서 산술 연산 처리 과정에서 발생합니다, 즉, 데이터 타입의 최대 값을 초과하는 덧셈 연산이 수행된 경우입니다.

**ERROR CODE: -459, 'Query result contains more than a single tuple.'**

- 이 메시지는 CUBRID 데이터베이스에서 단일 튜플(행)을 반환해야 하는 쿼리가 여러 개의 튜플을 반환했을 때 나타나는 오류입니다, 쿼리에서 단일 튜플 검증 과정에서 발생합니다, 즉, 서브쿼리나 특정 연산에서 정확히 하나의 결과만 반환되어야 하는데 여러 개가 반환된 경우입니다.

**ERROR CODE: -482, "'%1\$s' is not a virtual class.'**

- 이 메시지는 CUBRID 데이터베이스에서 'virtual class'(가상 테이블, 뷰테이블) 아닌 객체를 가상 테이블로 사용하려고 할 때 나타나는 오류입니다, 'virtual class'는 가상 테이블, 뷰테이블과 같은 개념으로 실제 데이터를 저장하지 않고 쿼리 결과를 보여주는 테이블입니다, 이 메시지는 가상 테이블 관련 작업을 수행하려고 할 때 해당 객체가 가상 테이블이 아닌 경우에 발생합니다.

**ERROR CODE: -486, 'Object is not an updatable virtual class instance.'**

- 이 메시지는 CUBRID 데이터베이스에서 가상 테이블(virtual class)의 인스턴스가 수정 가능하지 않을 때 나타나는 오류입니다, 수정 불가능한 가상 테이블 인스턴스(레코드)에 대해 수정 작업을 시도할 때 발생합니다.

**ERROR CODE: -492, '%1\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 파서에서 일반 오류가 발생할 때 나타나는 오류입니다, 이는 질의 파싱 중 오류가 발생할 경우 라인(line), 컬럼(column)을 표시하기 위해 사용하는 오류입니다.

**ERROR CODE: -493, 'Syntax: %1\$s %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 파서에서 구문 오류가 발생할 때 나타나는 오류입니다, 이는 파서의 구문 분석 단계에서 발생하는 오류로, SQL 쿼리의 문법적 분석 중에 문제가 발생했을 때 이 오류가 발생합니다.

**ERROR CODE: -494, 'Semantic: %1\$s %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 파서에서 시맨틱 오류가 발생할 때 나타나는 오류입니다, 파서의 시맨틱 분석 단계에서 발생하는 오류로, 쿼리의 의미적 분석 중에 문제가 발생했을 때 사용됩니다, 주로 쿼리의 의미적 오류나 타입 불일치 등의 상황에서 발생합니다.

**ERROR CODE: -495, 'Execute: %1\$s %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 파서에서 실행 오류가 발생할 때 나타나는 오류입니다, 이는 파서의 실행 단계에서 발생하는 오류로 XASL로 수행 처리를 할 수 없는 경우 발생합니다, 주로 쿼리 실행 실패나 파서 내부 오류 상황에서 발생합니다.

**ERROR CODE: -501, 'Invalid trigger priority.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 우선순위(priority)가 유효하지 않을 때 나타나는 오류입니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -502, 'Missing target class in trigger.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거를 정의할 때 대상 테이블이 지정되지 않았을 때 나타나는 오류입니다, 트리거는 특정 테이블의 이벤트(INSERT, UPDATE, DELETE)에 반응하도록 정의되므로, 대상 테이블은 필수적으로 지정되어야 합니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -503, 'Trigger "%1\$s" was not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 지정된 이름의 트리거를 찾을 수 없을 때 나타나는 오류입니다, 트리거가 존재하지 않거나, 사용자가 해당 트리거에 접근할 권한이 없을 때 발생합니다, 이는 트리거 조회 및 접근 과정에서 발생하는 오류입니다.

**ERROR CODE: -505, 'A trigger with the name "%1\$s" already exists.'**

- 이 메시지는 CUBRID 데이터베이스에서 동일한 이름의 트리거가 이미 존재할 때 나타나는 오류입니다, 트리거 이름은 전역적으로 고유해야 하므로, 중복된 이름으로 트리거를 생성하거나 이미 존재한 이름으로 변경할 수 없습니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -506, 'A trigger cannot be defined on "%1\$s" because it is a virtual class.'**

- 이 메시지는 CUBRID 데이터베이스에서 가상 테이블(virtual class)에 트리거를 정의하려고 할 때 나타나는 오류입니다, 가상 테이블은 뷰(view)와 같은 가상적인 테이블로, 실제 데이터를 저장하지 않기 때문에 트리거를 정의할 수 없습니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -507, 'Inappropriate trigger target class "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 대상 테이블이 부적절할 때 나타나는 오류입니다, 트리거가 특정 테이블을 대상으로 할 때, 해당 테이블이 존재하지 않거나 적절하지 않은 경우 발생합니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -508, 'Inappropriate trigger target attribute "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 대상 컬럼이 부적절할 때 나타나는 오류입니다, 트리거가 특정 컬럼을 대상으로 할 때, 해당 컬럼이 존재하지 않거나 적절하지 않은 경우 발생합니다, 이는 트리거 정의의 유효성을 검사하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -511, 'Not authorized to access trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 현재 사용자가 특정 트리거에 액세스(읽기/조회)할 권한이 없을 때 나타나는 오류입니다, 트리거의 정보를 조회하거나 참조하려고 할 때 권한이 부족한 경우 발생합니다.

**ERROR CODE: -512, 'Not authorized to delete trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 현재 사용자가 특정 트리거를 삭제할 권한이 없을 때 나타나는 오류입니다, 트리거를 완전히 제거하려고 할 때 권한이 부족한 경우 발생합니다.

**ERROR CODE: -514, 'Not authorized to alter trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 현재 사용자가 특정 트리거를 변경할 권한이 없을 때 나타나는 오류입니다, 트리거의 이름 변경, 상태 변경, 우선순위 변경, 주석 변경 등의 작업을 수행하려고 할 때 권한이 부족한 경우 발생합니다.

**ERROR CODE: -515, 'Trigger action execution time of %2\$s cannot be earlier than condition execution time of %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 액션 실행 시간이 조건 실행 시간보다 빠를 수 없다는 규칙을 위반했을 때 나타나는 오류입니다, 트리거의 조건(condition)이 실행된 후에 액션(action)이 실행되어야 하는데, 액션이 조건보다 먼저 실행되도록 설정된 경우 발생합니다, 이는 트리거의 논리적 실행 순서를 보장하기 위한 안전장치입니다.

**ERROR CODE: -516, 'Maximum trigger depth %1\$d exceeded at trigger "%2\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 재귀 호출 깊이가 최대 허용 깊이를 초과했을 때 나타나는 오류입니다, 트리거가 다른 트리거를 호출하고, 그 트리거가 또 다른 트리거를 호출하

는 과정에서 재귀 호출이 무한히 반복되거나 최대 깊이를 초과했을 때 발생합니다, 이는 트리거의 재귀 호출로 인한 무한 루프나 스택 오버플로우를 방지하기 위한 안전장치입니다.

**ERROR CODE: -517, 'he operation has been rejected by trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 작업(예: INSERT, UPDATE, DELETE)이 트리거의 *REJECT* 액션에 의해 거부되었을 때 나타나는 오류입니다, *REJECT* 액션은 트리거가 실행될 때 현재 트랜잭션을 롤백하고 오류를 발생시켜 작업을 중단시킵니다, 이는 트리거가 정의된 조건에 따라 데이터 변경 작업을 명시적으로 차단했음을 의미합니다.

**ERROR CODE: -518, 'Internal error: processing trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거를 처리하는 과정에서 내부 오류가 발생했을 때 나타나는 오류입니다, 트리거 컴파일 과정에서 파서 오류가 발생했지만 구체적인 오류 정보를 가져올 수 없는 상황에서 발생합니다, 이는 CUBRID의 트리거 시스템에서 예상치 못한 내부 상태 오류를 나타냅니다.

**ERROR CODE: -519, 'Illegal trigger condition type.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 조건 타입이 유효하지 않을 때 나타나는 오류입니다, 이는 CUBRID의 트리거 조건이 적합한 표현식이어야 한다는 제약사항을 나타냅니다.

**ERROR CODE: -520, 'The REJECT action cannot be used with an action time of AFTER or DEFERRED.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거를 생성할 때 *REJECT* 액션을 *AFTER* 또는 *DEFERRED* 액션 시간과 함께 사용하려고 할 때 나타나는 오류입니다, *REJECT* 액션은 트리거가 실행될 때 트랜잭션을 거부하는 액션으로, *AFTER*나 *DEFERRED* 액션 시간과는 호환되지 않습니다, 이는 CUBRID의 트리거 시스템에서 특정 액션과 액션 시간 조합이 허용되지 않는다는 제약사항을 나타냅니다.

**ERROR CODE: -521, 'The REJECT action cannot be used with the ABORT or TIMEOUT events.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거를 생성할 때 *REJECT* 액션을 *ABORT* 또는 *TIMEOUT* 이벤트와 함께 사용하려고 할 때 나타나는 오류입니다, *REJECT* 액션은 트리거가 실행될 때 트랜잭션을 거부하는 액션으로, *ABORT*나 *TIMEOUT* 이벤트와는 호환되지 않습니다, 이는 CUBRID의 트리거 시스템에서 특정 액션과 이벤트 조합이 허용되지 않는다는 제약사항을 나타냅니다.

**ERROR CODE: -523, 'Trigger action expression was not given.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거를 생성할 때 액션 타입이 *EXPRESSION*으로 설정되었지만, 액션 소스 문자열(*action\_source*)이 제공되지 않았을 때 나타나는 오류입니다, 트리거 액션은 트리거가 실행될 때 수행할 작업을 정의하는 부분이며, 액션 타입이 표현식(*EXPRESSION*)인 경우 반드시 액션 소스 문자열이 제공되어야 합니다, 이 메시지는 트리거 생성 과정에서 필수 파라미터가 누락되었을 때 발생합니다.

**ERROR CODE: -524, 'The specified activity cannot be removed because it is owned by another user.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 지연된 액티비티(deferred activity)를 제거하려고 할 때, 해당 액티비티가 현재 사용자가 아닌 다른 사용자가 소유하고 있어서 제거할 수 없다는 오류입니다, 트리거 액티비티는 트리거가 실행될 때 수행되는 작업을 나타내며, 지연된 액티비티는 트리거 실행이 지연된 상태에서 관리되는 액티비티입니다, 권한 문제로 인해 발생하며, 보안상의 이유로 다른 사용자가 소유한 액티비티를 제거할 수 없도록 제한합니다.

**ERROR CODE: -525, 'Error compiling condition for "%1\$s": %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 조건을 컴파일하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 주로 트리거의 조건 부분에 정의된 SQL 문이나 스크립트를 파싱하고 컴파일할 때 문법 오류나 컴파일 오류가 발생했을 때 나타납니다.

**ERROR CODE: -526, 'Error compiling action for "%1\$s", %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 액션을 컴파일하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 주로 트리거의 액션 부분에 정의된 SQL 문이나 스크립트를 파싱하고 컴파일할 때 문법 오류나 컴파일 오류가 발생했을 때 나타납니다.

**ERROR CODE: -527, 'Error evaluating condition for "%1\$s": %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 조건을 평가(실행)하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 주로 트리거의 조건 부분에서 SQL 문이나 스크립트를 실행할 때 문법 오류나 실행 오류가 발생했을 때 나타납니다.

**ERROR CODE: -528, 'Error evaluating action for "%1\$s", %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거의 액션을 평가(실행)하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 주로 트리거의 액션 부분에서 SQL 문이나 스크립트를 실행할 때 문법 오류나 실행 오류가 발생했을 때 나타납니다.

**ERROR CODE: -529, 'Transaction cannot be committed, it was invalidated by trigger "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션을 커밋하려고 할 때, 트리거에 의해 트랜잭션이 무효화되어 커밋할 수 없을 때 나타나는 오류입니다, 주로 트리거의 액션이 'INVALIDATE'로 설정되어 있을 때 발생합니다.

**ERROR CODE: -539, 'Attempt to divide by zero.'**

- 이 메시지는 CUBRID 데이터베이스에서 0으로 나누기 연산을 시도할 때 나타나는 오류입니다, 주로 쿼리 처리 과정에서 나누기 연산의 피제수가 0일 때 발생합니다, 수학적으로 정의되지 않은 연산이므로 CUBRID에서 오류로 처리합니다.

**ERROR CODE: -552, 'Date arithmetic underflow.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜 연산 또는 함수를 수행할 때 언더플로우가 발생했을 때 나타나는 오류입니다, 이는 주로 날짜 값에서 다른 값을 빼거나 연산할 때 결과가 날짜 범위를 벗어날 때 발생합니다, 날짜 연산 결과가 최소 날짜 값보다 작아질 때 발생하는 오류입니다.

**ERROR CODE: -553, 'Time arithmetic underflow.'**

- 이 메시지는 CUBRID 데이터베이스에서 시간 연산 또는 함수를 수행할 때 언더플로우가 발생했을 때 나타나는 오류입니다, 이는 주로 시간 값에서 다른 값을 빼거나 연산할 때 결과가 시간 범위를 벗어날 때 발생합니다, 시간 연산 결과가 최소 시간 값보다 작아질 때 발생하는 오류입니다.

**ERROR CODE: -618, 'Attribute is not updatable.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 컬럼을 업데이트하려고 시도했지만, 해당 컬럼이 업데이트 불가능한 유형일 때 나타납니다, 뷰테이블(view table)의 특정 컬럼이나 시스템에 의해 관리되는 컬럼을 직접 업데이트할 때 발생할 수 있습니다.

**ERROR CODE: -621, 'Invalid data type. The arguments contain an invalid data type.'**

- 이 메시지는 CUBRID 데이터베이스에서 함수나 연산에 전달된 인수의 데이터 타입이 유효하지 않을 때 나타납니다, 예를 들어, 문자열 함수에 숫자 타입을 전달하거나, 숫자 연산에 문자열 타입을 전달할 때 발생할 수 있습니다.

**ERROR CODE: -622, 'Incompatible code sets. The code sets of the arguments are incompatible.'**

- 이 메시지는 CUBRID 데이터베이스에서 문자열 비교 또는 연산 시 인수로 전달된 문자열들의 문자셋(character set)가 서로 호환되지 않을 때 나타납니다, 예를 들어, UTF-8 문자열과 EUC-KR 문자열을 직접 비교하거나 연산하려고 할 때 발생할 수 있습니다.

**ERROR CODE: -623, 'Invalid escape sequence. An invalid escape sequence has caused an error in the regular expression compiler.'**

- 이 메시지는 CUBRID 데이터베이스에서 LIKE 연산자의 ESCAPE 절에 잘못된 이스케이프 시퀀스를 사용할 때 나타납니다, SQL 구문의 LIKE 연산자에서 특수 문자를 이스케이프할 때 사용하는 시퀀스가 유효하지 않을 때 발생합니다.

**ERROR CODE: -624, 'Invalid escape character. The escape character is non-NULL with a length greater than 1.'**

- 이 메시지는 CUBRID 데이터베이스에서 LIKE 연산자의 ESCAPE 절에 잘못된 이스케이프 문자를 사용할 때 나타납니다.

**ERROR CODE: -628, 'Invalid currency type: %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 통화(monetary) 데이터 타입을 처리할 때 발생하는 오류입니다, 이는 지원되지 않는 통화 타입이 지정되었을 때 발생합니다.



**ERROR CODE: -631, 'SQL statement violated NOT NULL constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 SQL 문장이 NOT NULL 제약 조건을 위반했을 때 발생하는 오류입니다, 테이블의 컬럼에 NOT NULL 제약 조건이 설정되어 있는데, INSERT나 UPDATE 문장에서 해당 컬럼에 NULL 값을 입력하려고 할 때 나타납니다.

**ERROR CODE: -649, 'Given precision of %1\$d is invalid; it should be greater than %2\$d and no greater than %3\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 데이터 타입의 정밀도(precision) 값이 유효하지 않을 때 발생하는 오류입니다, 지정된 정밀도 값이 해당 데이터 타입에서 허용되는 범위를 벗어났을 때 발생합니다, 이는 데이터 타입 정의 시 정밀도 값이 너무 작거나 너무 클 때 발생하는 검증 실패를 의미합니다.

**ERROR CODE: -670, 'Operation would have caused one or more unique constraint violations. INDEX %1\$s%2\$s ON CLASS %3\$s%4\$s. key: %5\$s%6\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 UNIQUE 제약 조건을 위반하는 작업이 시도되었을 때 발생합니다, B+tree 인덱스에서 중복된 키 값이 발견되어 UNIQUE 제약 조건을 위반하게 되는 상황을 나타냅니다, 이는 데이터 무결성을 보장하기 위해 중복된 값을 허용하지 않는 UNIQUE 제약 조건이 위반되었음을 의미합니다.

**ERROR CODE: -683, 'Attempted to create string with illegal length %1\$d'**

- 이 메시지는 CUBRID 데이터베이스에서 문자열을 생성하려고 할 때, 지정된 길이가 유효하지 않을 때 발생하는 오류입니다, 문자열 길이는 CUBRID에서 정의된 최대 길이를 초과하거나 음수일 수 없습니다, 주로 clob\_to\_char() 함수에서 LOB(Large Object) 데이터를 문자열로 변환하는 과정에서 발생합니다.

**ERROR CODE: -685, 'Class "%1\$s" is invalid.'**

- 이 메시지는 CUBRID 데이터베이스에서 owner 변경 시 지정된 테이블이 유효하지 않을 때 발생하는 오류입니다.

**ERROR CODE: -704, 'Evaluation of generic server function failed.'**

- 이 메시지는 CUBRID 데이터베이스에서 자바 프로시저를 평가(evaluation)하는 과정에서 실패했을 때 발생합니다,

**ERROR CODE: -709, 'Constraint "%1\$s" not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 이름의 제약조건(constraint)을 찾을 수 없을 때 나타나는 오류입니다, CUBRID는 제약조건을 삭제하거나 수정할 때 해당 제약조건이 존재하는지 확인합니다, 이 메시지는 CUBRID 데이터베이스에서 제약조건을 삭제하거나 수정하는 과정에서 발생합니다



**ERROR CODE: -710, 'Invalid constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 제약조건(constraint)이 유효하지 않을 때 나타나는 오류입니다, CUBRID는 제약조건을 생성하거나 관리할 때 해당 제약조건이 유효한지 검증합니다, 이 메시지는 CUBRID 데이터베이스에서 제약조건을 생성하거나 관리하는 과정에서 발생할 수 있습니다.

**ERROR CODE: -711, 'An object with the supplied attribute value doesn't exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 컬럼 값을 가진 객체를 찾을 수 없을 때 나타나는 오류입니다, CUBRID는 객체 검색이나 쿼리 실행 중에 해당 조건을 만족하는 객체가 존재하지 않을 때 이 오류를 발생시킵니다.

**ERROR CODE: -712, 'Constraint" %1\$s" already exists.'**

- 이 메시지는 CUBRID 데이터베이스에서 동일한 이름의 제약조건(constraint)이 이미 존재할 때 나타나는 오류입니다, CUBRID는 테이블이나 테이블에 제약조건을 추가할 때 중복된 이름을 허용하지 않습니다.

**ERROR CODE: -713, 'An index does not exist on this attribute.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 컬럼에 대해 인덱스가 존재하지 않을 때 나타나는 오류입니다, CUBRID 데이터베이스에서 객체 검색이나 인덱스 관련 작업 중 발생할 수 있습니다.

**ERROR CODE: -716, 'NOT NULL constraints are not allowed in virtual or proxy classes.'**

- 이 메시지는 CUBRID 데이터베이스에서 가상 테이블(virtual class), 뷰테이블에 NOT NULL 제약 조건을 추가하려고 할 때 나타나는 오류입니다.

**ERROR CODE: -717, 'UNIQUE constraints are not allowed in virtual or proxy classes.'**

- 이 메시지는 CUBRID 데이터베이스에서 가상 테이블(virtual class), 뷰테이블에 UNIQUE 제약 조건을 추가하려고 할 때 나타나는 오류입니다.

**ERROR CODE: -721, 'Insert query yields more than one row.'**

- 이 메시지는 CUBRID 데이터베이스에서 INSERT 문을 실행할 때 결과가 하나의 행이 아닌 여러 행을 반환했을 때 나타나는 오류입니다, INSERT 문은 일반적으로 하나의 행만 삽입해야 하는데, 서브쿼리나 VALUES 절에서 여러 행이 반환되면 이 오류가 발생합니다, 이 메시지는 주로 INSERT 문의 VALUES 절이 여러 개이거나, 서브쿼리가 여러 행을 반환할 때 발생합니다.

**ERROR CODE: -722, 'Wrong number of attributes for NOT NULL constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 NOT NULL 제약 조건을 생성하거나 삭제할 때 컬럼의 개수가 잘못되었을 때 나타나는 오류입니다, NOT NULL 제약 조건은 하나의 컬럼에만 적용할 수 있는 제약 조건입니다, 이 메시지는 여러 컬럼에 NOT NULL 제약 조건을 동시에 적용하려고 하거나, 컬럼이 지정되지 않았을 때 발생합니다.

**ERROR CODE: -723, 'Cannot modify the domain of an existing attribute.'**

- 이 메시지는 CUBRID 데이터베이스에서 이미 존재하는 컬럼의 데이터 타입을 변경하려고 시도했을 때 나타나는 오류입니다.

**ERROR CODE: -724, 'Invalid parameter(s) for default constraint name generation.'**

- 이 메시지는 CUBRID 데이터베이스에서 인덱스를 삭제 시 인덱스 이름이 NULL인 경우 발생할 수 있습니다.

**ERROR CODE: -727, 'Invalid trigger event.'**

- 이 메시지는 CUBRID 데이터베이스에서 트리거(trigger)를 생성하거나 관리하는 과정에서 유효하지 않은 트리거 이벤트가 사용되었을 때 나타나는 오류입니다, 트리거 이벤트는 트리거가 언제 실행될지를 결정하는 조건으로, INSERT, UPDATE, DELETE 등의 데이터베이스 작업과 연관됩니다, 이 메시지는 유효하지 않은 이벤트 타입이 트리거에 지정되었을 때 발생합니다.

**ERROR CODE: -729, 'Overflow occurred in subtraction context.'**

- 이 메시지는 CUBRID 데이터베이스에서 뺄셈 연산을 수행하는 과정에서 오버플로우가 발생했을 때 나타나는 오류입니다, 오버플로우는 연산 결과가 해당 데이터 타입이 표현할 수 있는 최대 범위를 초과할 때 발생합니다, 이는 주로 두 숫자의 차이가 정수(INT, BIGINT, SHORT) 또는 부동 소수점(FLOAT, DOUBLE) 타입의 최대값을 초과할 때 발생합니다, 또한 복합 연산에서 뺄셈 부분이 오버플로우를 유발할 때도 발생할 수 있습니다.

**ERROR CODE: -730, 'Overflow occurred in multiplication context.'**

- 이 메시지는 CUBRID 데이터베이스에서 곱셈 연산을 수행하는 과정에서 오버플로우가 발생했을 때 나타나는 오류입니다, 오버플로우는 연산 결과가 해당 데이터 타입이 표현할 수 있는 최대 범위를 초과할 때 발생합니다, 이는 주로 두 숫자의 곱이 정수(INT, BIGINT, SHORT) 또는 부동 소수점(FLOAT, DOUBLE) 타입의 최대값을 초과할 때 발생합니다, 또한 NUMERIC 타입의 곱셈에서도 결과가 표현 가능한 범위를 초과할 때도 발생할 수 있습니다.

**ERROR CODE: -731, 'Overflow occurred in division context.'**

- 이 메시지는 CUBRID 데이터베이스에서 나눗셈 연산을 수행하는 과정에서 오버플로우가 발생했을 때 나타나는 오류입니다, 오버플로우는 연산 결과가 해당 데이터 타입이 표현할 수 있는 최대 범위를 초과할 때 발생합니다, 이는 주로 부동 소수점 나눗셈에서 매우 큰 값이 생성되거나, 정수 나눗셈에서 특정 조건(예: INT\_MIN / -1)에서 결과가 범위를 초과할 때 발생할 수 있습니다, 또한 NUMERIC 타입의 나눗셈에서도 결과가 표현 가능한 범위를 초과할 때도 발생할 수 있습니다.

**ERROR CODE: -732, 'Overflow occurred in unary minus context.'**

- 이 메시지는 CUBRID 데이터베이스에서 단항 마이너스 연산을 수행하는 과정에서 오버플로우가 발생했을 때 나타나는 오류입니다, 단항 마이너스는 숫자 값의 부호를 바꾸는 연산으로, 양수를 음수로, 음수를 양수로 변환합니다, 이 메시지는 특정 데이터 타입의 최소값에 단항 마이너스 연산

을 적용할 때 발생합니다, 예를 들어, INT 타입의 최소값(-2,147,483,648)에 단항 마이너스 연산을 적용하면 양수 최대값(2,147,483,648)이 되어 INT 타입의 범위를 초과합니다.

**ERROR CODE: -751, 'The OID of a new object isn't available in a BEFORE INSERT trigger.'**

- 이 메시지는 CUBRID 데이터베이스에서 BEFORE INSERT 트리거를 정의할 때, 트리거 내에서 새 객체를 직접 참조하려고 할 때 발생합니다, BEFORE INSERT 트리거는 데이터가 실제로 테이블에 삽입되기 전에 실행되므로, 새 객체의 OID는 아직 할당되지 않은 상태입니다, 트리거 내에서 "new" 객체 자체를 참조하는 것은 허용되지 않으며, "new.attribute\_name" 형태로 개별 속성만 참조할 수 있습니다.

**ERROR CODE: -772, 'db\_serial class not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 db\_serial 카탈로그 테이블을 찾으려고 할 때, 해당 테이블이 존재하지 않는 상황임을 나타냅니다, 이는 CUBRID의 serial 객체 관리 메커니즘에서 발생하며, 주로 db\_serial 테이블이 시스템에 등록되지 않았을 때 발생할 수 있습니다.

**ERROR CODE: -773, 'Serial "%1\$s" not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 serial 객체를 조회하려고 할 때, 해당 이름의 serial 객체가 존재하지 않는 상황임을 나타냅니다.

**ERROR CODE: -774, 'Serial "%1\$s" already exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 serial 객체를 생성하려고 할 때, 동일한 이름의 serial 객체가 이미 존재하는 상황임을 나타냅니다.

**ERROR CODE: -775, 'Next value exceeds MAX(MIN) value.'**

- 이 메시지는 CUBRID 데이터베이스에서 serial 객체의 다음 값(nextval)을 계산할 때, 계산된 값이 최대값 또는 최소값을 초과하는 상황임을 나타냅니다.

**ERROR CODE: -776, 'Cannot fetch serial object.'**

- 이 메시지는 CUBRID 데이터베이스에서 serial 객체를 가져오려고 할 때, serial 객체를 읽을 수 없는 상황임을 나타냅니다.

**ERROR CODE: -777, 'Cannot update serial object.'**

- 이 메시지는 CUBRID 데이터베이스에서 serial 객체를 업데이트하려고 할 때, serial 객체를 수정할 수 없는 상황임을 나타냅니다.

**ERROR CODE: -779, 'Result exceeds limit: year of Date MUST be in 1 and 9999.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜 관련 함수를 사용할 때, 결과 날짜의 연도가 1년에서 9999년 범위를 벗어나는 경우 발생합니다, 이는 CUBRID의 날짜 처리 메커니즘에서 발생하며, 주로 날짜 연산 결과가 유효한 날짜 범위를 초과할 때 발생할 수 있습니다.

**ERROR CODE: -781, 'Format string is too long. It must be less than 16K.'**

- 이 메시지는 CUBRID 데이터베이스에서 문자열 포맷 변환 함수를 사용할 때, 포맷 문자열의 길이가 16K(16,000 바이트)를 초과하는 경우 발생합니다, 이는 CUBRID의 문자열 포맷 변환 메커니즘에서 발생하며, 주로 포맷 문자열 길이 제한을 초과할 때 발생할 수 있습니다.

**ERROR CODE: -782, 'Empty string not allowed here.'**

- 이 메시지는 빈 문자열("")이 허용되지 않는 문자열 연산이나 함수에서 빈 문자열이 사용될 때 발생합니다, 특정 문자열 함수나 연산에서 의미 있는 입력값이 필요한 경우 발생하는 데이터 유효성 검사 오류입니다.

**ERROR CODE: -783, 'Invalid format.'**

- 이 메시지는 문자열 형식이나 데이터 포맷이 예상된 형식과 일치하지 않을 때 발생합니다. 날짜/시간 형식, 숫자 형식, 또는 특정 패턴을 요구하는 문자열 함수에서 잘못된 형식이 입력될 때 발생하는 포맷 검증 오류입니다.

**ERROR CODE: -784, 'Two arguments are mismatched.'**

- 이 메시지는 CUBRID 데이터베이스의 문자열 처리 모듈에서 두 개의 인자가 서로 맞지 않을 때 발생합니다, 즉, 날짜 변환을 수행할 때, 소스 문자열과 포맷 문자열이 일치하지 않을 때 발생하는 오류 메시지입니다, 이 메시지는 주로 숫자 파싱이 실패했을 때 발생하며, 이는 소스 문자열의 형식이 예상된 포맷과 맞지 않음을 의미합니다.

**ERROR CODE: -785, 'First argument string is too long. It must be less than 16K.'**

- 이 메시지는 CUBRID 데이터베이스의 문자열 처리 모듈에서 날짜/시간 변환 함수를 호출할 때, 첫 번째 인자로 전달된 문자열의 길이가 16KB(16000바이트)를 초과할 경우 발생합니다, 즉, 원본 문자열의 길이를 검사하는 로직에서 이 제한을 초과하면 오류가 설정됩니다, 이 제한은 데이터베이스 시스템이 문자열 처리 시 발생할 수 있는 메모리 오버플로우나 성능 저하를 방지하기 위해 설정된 것입니다.

**ERROR CODE: -786, 'Format is duplicated.'**

- 이 메시지는 CUBRID 데이터베이스의 문자열 처리 모듈에서 날짜/시간 변환 함수를 호출할 때, 형식 문자열 내에 동일한 형식 지정자가 중복되어 사용된 경우 발생합니다, 즉, 날짜/시간 형식 문자열을 파싱하는 과정에서, 이미 한 번 사용된 형식 지정자(예: YYYY, MM, DD 등)가 다시 나타날 때 오류가 설정됩니다, 이는 날짜/시간 형식 문자열의 문법적 오류를 방지하기 위한 CUBRID의 내부 검증 메커니즘입니다.

**ERROR CODE: -787, 'Conversion error in time format.'**

- 이 메시지는 CUBRID 데이터베이스의 시간 처리 모듈에서 시간 형식 변환 과정에서 발생하는 오류입니다, 즉, 시간 데이터를 다른 형식으로 변환하거나, 시간대를 변환하거나, 시간 값의 유효성을 검사하는 과정에서 문제가 발생했을 때 오류가 설정됩니다, 이 메시지는 여러 모듈에서 발생할 수 있으며, 주로 시간 데이터의 무결성과 일관성을 보장하기 위한 검증 과정에서 발생합니다.

**ERROR CODE: -788, 'Conversion error in timestamp format.'**

- 이 메시지는 CUBRID 데이터베이스의 타임스탬프 처리 모듈에서 타임스탬프 형식 변환 과정에서 발생하는 오류입니다, 즉, 타임스탬프 데이터를 다른 형식으로 변환하거나, 날짜/시간 문자열을 타임스탬프로 파싱하거나, 타임스탬프 값의 유효성을 검사하는 과정에서 문제가 발생했을 때 오류가 설정됩니다, 이 메시지는 여러 모듈에서 발생할 수 있으며, 주로 타임스탬프 데이터의 무결성과 일관성을 보장하기 위한 검증 과정에서 발생합니다.

**ERROR CODE: -791, 'Invalid serial value.'**

- 이 메시지는 CUBRID 데이터베이스의 serial 객체 처리 모듈에서 serial 값의 유효성 검사에 실패했을 때 발생하는 오류입니다, 즉, serial 객체의 속성 값을 검증하는 과정에서, 필수 속성 값이 NULL 이거나 올바른 데이터 타입이 아닐 때 오류가 설정됩니다, serial 객체의 무결성을 보장하기 위한 검증 과정에서 발생하며, 주로 스키마 내보내기나 SERIAL 객체 관리 과정에서 나타납니다.

**ERROR CODE: -833, 'The index name "%1\$s" is ambiguous. Specify the class name of the index.'**

- 이 메시지는 CUBRID 데이터베이스의 스키마 관리 모듈에서 인덱스 이름으로 테이블을 찾으려 할 때 발생합니다, 즉, 특정 인덱스 이름과 타입으로 해당 인덱스를 소유한 테이블을 찾으려 할 때, 동일한 이름과 타입의 인덱스가 여러 테이블에 존재하여 어떤 테이블을 반환해야 할지 결정할 수 없을 때 발생하는 오류 메시지입니다, 데이터베이스 설계 시 여러 테이블에서 동일한 인덱스 이름을 사용했거나, 인덱스 참조 시 테이블 이름을 명시하지 않았을 때 발생합니다. CUBRID는 인덱스 이름만으로는 어떤 테이블의 인덱스인지 구분할 수 없으므로 테이블 이름을 명시하도록 요구합니다.

**ERROR CODE: -834, 'There are some mismatches between source string "%1\$s" and format string "%2\$s" given in to\_number().'**

- 이 메시지는 CUBRID 데이터베이스의 쿼리 처리 모듈에서 `to_number()` 함수를 사용하여 문자열을 숫자로 변환하는 과정에서 발생합니다, 즉, 원본 문자열과 형식 문자열을 비교하여 숫자로 변환을 시도할 때, 두 문자열의 구조나 내용이 서로 호환되지 않아 변환 규칙을 따를 수 없을 때 발생하는 오류 메시지입니다, 주로 SQL 쿼리에서 `to_number()` 함수를 잘못 사용했거나, 입력 데이터의 형식이 예상과 다를 때 발생합니다. 예를 들어, 숫자 형식으로 변환해야 할 문자열에 숫자가 아닌 문자가 포함되어 있거나, 지정된 형식 문자열이 실제 데이터의 형식과 맞지 않을 때 발생합니다.

**ERROR CODE: -865, 'Cannot execute statement due to unknown type input markers.'**

- 이 메시지는 CUBRID 데이터베이스에서 SQL 문장을 실행하는 과정에서 발생합니다, 이는 주로 호스트 변수의 타입을 결정할 수 없거나, 타입 변환이 불가능한 경우에 나타납니다.

**ERROR CODE: -874, 'Argument of power() is out of range.'**

- 이 메시지는 CUBRID 데이터베이스에서 거듭제곱(power) 연산을 수행하는 과정에서 발생합니다, 구체적으로는 음수 밑수에 비정수 지수를 사용하는 경우에 발생합니다. 이는 수학적으로 정의되지 않은 연산이므로 CUBRID에서 오류로 처리합니다.

**ERROR CODE: -875, 'Overflow happened in power context.'**

- 이 메시지는 CUBRID 데이터베이스에서 거듭제곱(power) 연산을 수행하는 과정에서 발생합니다, 이런 상황은 일반적으로 매우 큰 수의 거듭제곱 연산을 수행할 때 발생하며, 수학적으로는 유효하지만 컴퓨터의 부동소수점 표현 범위를 초과하는 경우에 나타납니다.

**ERROR CODE: -886, "'%1\$s' caused unique constraint violation.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블에 정의된 유니크(Unique) 제약 조건을 위반하는 데이터를 삽입(INSERT)하거나 갱신(UPDATE)하려고 할 때 발생합니다, 이 메시지는 이미 존재하는 값과 동일한 값을 삽입하거나, 기존 값을 업데이트하여 다른 레코드와 중복되는 값을 만들려고 할 때 발생합니다.

**ERROR CODE: -887, 'Stored procedure '%1\$s' is already exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 저장 프로시저를 생성하려고 할 때, 동일한 이름의 저장 프로시저가 이미 존재하여 발생합니다, 즉, *CREATE PROCEDURE* 문 실행 시 저장 프로시저 이름이 중복되었을 때 오류가 설정됩니다, CUBRID의 저장 프로시저 관리 메커니즘에서 발생하며, 주로 저장 프로시저 이름 중복으로 인해 발생할 수 있습니다.

**ERROR CODE: -888, 'Parameter count is invalid. expected: %1\$d, actual: %2\$d'**

- 이 메시지는 CUBRID 데이터베이스에서 저장 프로시저를 호출할 때, 전달된 매개변수 개수가 저장 프로시저 정의에서 기대하는 매개변수 개수와 일치하지 않아 발생합니다, 즉, 저장 프로시저 호출 시 매개변수 개수가 맞지 않을 때 오류가 설정됩니다, CUBRID의 저장 프로시저 호출 메커니즘에서 발생하며, 주로 매개변수 개수 불일치로 인해 발생할 수 있습니다.

**ERROR CODE: -889, 'Stored procedure execute error: %1\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 저장 프로시저(Stored Procedure)를 실행하는 과정에서 발생하는 일반적인 실행 오류입니다, 즉, 저장 프로시저의 실행 중에 다양한 원인으로 인해 오류가 발생했음을 나타내는 오류입니다, 이런 상황은 일반적으로 저장 프로시저 내부의 SQL 문 실행 실패, 데이터 타입 불일치, 권한 부족, 메모리 부족, 또는 기타 런타임 오류 등으로 발생합니다.

**ERROR CODE: -890, 'Partition failed.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블 파티션 생성, 변경, 삭제 등 파티셔닝 관련 작업을 수행하는 도중 내부 오류, 파라미터 오류, 데이터 불일치, 리소스 부족, 제약 조건 위반 등으로 인해 작업이 실패했을 때 발생합니다, 주로 파티션 정의가 잘못되었거나, 데이터가 파티션 조건에 맞지 않거나, 시스템 리소스 부족, 내부 버그, 스키마 제약 조건 위반 등 다양한 원인으로 인해 나타날 수 있습니다.

**ERROR CODE: -891, 'Appropriate partition does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블 파티션 작업(예: 데이터 삽입, 조회, 이동, 파티션 변경 등의 탐색 과정)을 수행할 때, 지정된 조건이나 값에 해당하는 파티션이 정의되어 있지 않거나, 이미 삭제되었거나, 파티션 정의가 잘못되어 접근할 수 없는 경우에 발생합니다, 주로 파티션 키



값이 파티션 범위에 포함되지 않거나, 파티션 이름/ID가 잘못 지정된 경우, 파티션이 삭제/변경된 경우, 내부 메타데이터 불일치 등으로 인해 나타날 수 있습니다.

**ERROR CODE: -892, 'Primary key "%2\$s" already defined for class "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블에 기본 키를 추가하거나 정의하려고 할 때, 해당 테이블에 이미 기본 키가 존재하여 발생합니다, 즉, `CREATE TABLE` 또는 `ALTER TABLE ADD PRIMARY KEY`와 같은 DDL(Data Definition Language) 문을 실행하여 기본 키를 정의하려 할 때, 기본 키 중복 검사에서 이미 존재할 경우 오류가 설정됩니다, 이 메시지는 CUBRID의 스키마 관리 메커니즘에서 발생하며, 주로 기본 키 중복 정의로 인해 발생할 수 있습니다.

**ERROR CODE: -893, 'Attribute '%1\$s', a member of the primary key, could not be dropped.'**

- 이 메시지는 CUBRID 데이터베이스에서 기본 키의 일부인 속성을 삭제하려고 할 때 발생합니다, 즉, DDL(Data Definition Language) 문을 실행하여 속성을 삭제하려 할 때, 해당 속성이 기본 키의 멤버인 경우 오류가 설정됩니다. CUBRID의 스키마 관리 메커니즘에서 발생하며, 주로 기본 키 무결성 보호로 인해 발생할 수 있습니다.

**ERROR CODE: -894, 'Stored procedure/function '%1\$s' does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 존재하지 않는 저장 프로시저나 함수를 호출하려고 할 때 발생합니다, 즉, `CALL` 문이나 함수 호출을 통해 저장 프로시저나 함수를 실행하려 할 때, 지정된 이름의 저장 프로시저나 함수가 데이터베이스에 존재하지 않을 경우 오류가 설정됩니다, CUBRID의 저장 프로시저 관리 메커니즘에서 발생하며, 주로 존재하지 않는 저장 프로시저/함수 호출로 인해 발생할 수 있습니다.

**ERROR CODE: -895, 'Invalid stored procedure type: '%1\$s' is the '%2\$s'.'**

- 이 메시지는 CUBRID 데이터베이스에서 저장 프로시저나 함수의 타입이 일치하지 않을 때 발생합니다, 즉, DDL(예를 들어 `function`으로 선언하고 `drop procedure`를 하는 경우) 문을 실행할 때, 실제 저장 프로시저의 타입과 요청된 타입이 다를 경우 오류가 설정됩니다.

**ERROR CODE: -899, 'Invalid partition requests.'**

- 이 메시지는 CUBRID 데이터베이스에서 파티션 관련 작업을 수행할 때 잘못된 요청이 발생하여 나타납니다, 즉, 파티션 테이블에 대한 부적절한 작업 요청이나 파티션 관련 제약 조건을 위반한 작업을 시도할 경우 오류가 설정됩니다, CUBRID의 파티션 관리 메커니즘에서 발생하며, 주로 파티션 관련 제약 조건 위반으로 인해 발생할 수 있습니다.

**ERROR CODE: -900, 'Java VM library is not found: %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 JVM(Java Virtual Machine)을 시작하기 위해 필요한 JVM 라이브러리를 찾거나 로드하는 데 실패했을 때 발생합니다, 즉, CUBRID 서버 프로세스가 자바 기반의 저장 프로시저(Stored Procedure)나 사용자 정의 함수(User Defined Function) 등을 실행하기 위해 JVM 프로세스를 생성하려고 시도했으나, 시스템 환경 변수(예: `JAVA_HOME`, `JVM_PATH`)에 JVM 라이브러리 경로가 올바르게 설정되어 있지 않거나, 라이브러리 파일 자체가 존

재하지 않거나, 접근 권한 문제 등으로 인해 로드할 수 없을 때 발생합니다. (11.4 버전부터는 pl\_server가 사용하는 JVM library를 찾지 못하는 경우 발생)

#### **ERROR CODE: -901, 'Java VM can not be started: %1\$s'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 JVM(Java Virtual Machine) 프로세스를 시작하려고 시도했지만 실패했을 때 발생합니다, 즉, CUBRID 서버 프로세스가 자바 기반의 저장 프로시저(Stored Procedure)나 사용자 정의 함수(User Defined Function) 등을 실행하기 위해 JVM 프로세스를 생성하고 초기화하려고 했으나, JVM 라이브러리 로딩 실패, JVM 생성 실패, 필요한 자바 테이블 찾기 실패, 메모리 할당 실패, 또는 JVM 초기화 과정에서 발생하는 다양한 문제로 인해 JVM 프로세스가 정상적으로 시작되지 못했음을 의미합니다. (11.4버전부터는 sp 처리를 위한 pl\_server를 구동할 수 없는 경우)

#### **ERROR CODE: -902, 'Java VM is not running.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 JVM(Java Virtual Machine) 프로세스가 실행되지 않은 상태에서 자바 기반 기능을 사용하려고 할 때 발생합니다., 즉, CUBRID 서버 프로세스가 자바 기반의 저장 프로시저(Stored Procedure)나 사용자 정의 함수(User Defined Function) 등을 실행하기 위해 JVM과의 연결을 시도했으나, JVM 프로세스 자체가 시작되지 않았거나, 이미 종료되었거나, CUBRID의 PL 서버 관리자가 초기화되지 않은 상태임을 의미합니다. (11.4버전부터는 sp 처리를 위한 pl\_server가 구동되어 있지 않은 경우)

#### **ERROR CODE: -903, 'Can't connect Java VM: %1\$s'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 JVM(Java Virtual Machine) 연결을 시도했지만 실패했을 때 발생합니다, 즉, CUBRID 서버 프로세스가 자바 기반의 저장 프로시저(Stored Procedure)나 사용자 정의 함수(User Defined Function) 등을 실행하기 위해 JVM 프로세스와 IPC(Inter-Process Communication) 또는 소켓 통신을 시도했으나, JVM 프로세스가 시작되지 않았거나, 비정상적으로 종료되었거나, CUBRID 서버가 JVM 프로세스를 찾을 수 없거나, 통신 설정에 문제가 있어 초기 연결 단계에서 실패했음을 의미합니다. (11.4버전부터는 sp 처리를 위한 pl\_server와 연결이 되지 않는 경우)

#### **ERROR CODE: -904, 'Invalid stored procedure name.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 저장 프로시저(Stored Procedure)나 함수(Function)의 이름이 유효하지 않다고 판단했을 때 발생합니다, 즉, CUBRID 서버가 저장 프로시저를 생성(CREATE), 수정(ALTER), 삭제(DROP), 또는 호출(CALL)하려고 할 때, 지정된 이름이 CUBRID의 명명 규칙을 만족하지 않거나, 이름 처리 과정에서 문제가 발생했음을 의미합니다, 이는 CUBRID의 저장 프로시저 관리 시스템에서 이름의 유효성을 검증하는 과정에서 발생하는 보호성 오류입니다.

#### **ERROR CODE: -905, 'Networking with JVM failed: %1\$d'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 JVM(Java Virtual Machine)과 통신을 시도하는 과정에서 네트워크 관련 문제가 발생하여 실패했을 때 발생합니다., 즉, CUBRID 서버 프로세스가 자바 기반의 저장 프로시저(Stored Procedure)나 사용자 정의 함수(User Defined Function) 등을 실행하기 위해 JVM 프로세스와 소켓 통신을 시도했으나, 연결 실패, 데이터 전송 오류, 타임아웃 등의 네



트위크 수준의 문제로 인해 통신이 중단되었음을 의미합니다, 이런 상황은 일반적으로 CUBRID 서버와 JVM 프로세스 간의 네트워크 설정 문제, 방화벽, JVM 프로세스의 비정상 종료, 시스템 자원 부족, 또는 CUBRID 내부의 JVM 연동 모듈 오류로 인해 발생합니다. (11.4버전부터는 sp 처리를 위한 pl\_server에 오류가 발생해서 결과 등을 주지 않는 경우)

**ERROR CODE: -907, 'The owner or a member of DBA group can drop stored procedure.'**

- 이 메시지는 현재 로그인한 사용자가 특정 저장 프로시저(Stored Procedure)를 변경(ALTER), 제거(DROP)하려고 시도했지만, 해당 저장 프로시저를 삭제할 수 있는 충분한 권한을 가지고 있지 않을 때 발생합니다, CUBRID에서는 저장 프로시저의 소유자(Owner)이거나 DBA(Database Administrator) 그룹의 멤버인 사용자만이 해당 저장 프로시저를 삭제할 수 있도록 엄격한 권한 정책을 적용합니다.

**ERROR CODE: -908, 'Too many arguments of stored procedure '%1\$s''**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 저장 프로시저(Stored Procedure)나 함수(Function)를 정의하거나 호출할 때, 해당 프로시저에 지정된 인자의 개수가 CUBRID가 허용하는 최대 개수를 초과했을 때 발생합니다, CUBRID 서버가 *CREATE PROCEDURE* 또는 *CREATE FUNCTION* 문을 처리하는 과정에서, 사용자가 정의한 저장 프로시저의 인자 수를 넘어섰음을 의미합니다, 이는 CUBRID의 저장 프로시저 관리 시스템에서 인자 개수의 유효성을 검증하는 과정에서 발생하는 보호성 오류입니다.

**ERROR CODE: -911, 'Invalid call: it can not return ResultSet.'**

- 이 메시지는 CUBRID 데이터베이스에서 저장 프로시저(Stored Procedure)를 호출할 때, 해당 저장 프로시저가 ResultSet(쿼리 결과 집합)을 반환할 수 없는 방식으로 정의되었거나, ResultSet을 반환할 수 없는 컨텍스트에서 호출되었을 때 발생합니다, 즉, CUBRID의 특정 저장 프로시저는 ResultSet을 직접 반환하도록 설계되지 않았거나, 현재 실행 환경(예: 다른 저장 프로시저 내에서 호출 등)이 ResultSet 반환을 지원하지 않는 경우에 이 오류가 발생합니다, 이런 상황은 일반적으로 *CALL* 문을 사용하여 저장 프로시저를 호출할 때, 프로시저의 정의나 호출 방식이 ResultSet 반환 규칙을 위반할 때 발생합니다, 이는 저장 프로시저의 실행 규칙과 데이터베이스 시스템의 일관성을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -912, 'ResultSet can not be used input parameter.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 저장 프로시저(Stored Procedure)나 함수(Function)를 정의 시 'RESULTSET' 타입을 입력 파라미터로 사용하려고 할 때 발생합니다, CUBRID의 저장 프로시저 시스템에서 *RESULTSET* 타입은 출력 파라미터로만 사용할 수 있도록 제한하고 있기 때문입니다. 이는 데이터의 흐름과 메모리 관리 측면에서 시스템의 안정성을 보호하기 위한 제약 조건입니다.

**ERROR CODE: -913, 'Too many nested stored procedure call.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 저장 프로시저(Stored Procedure)가 다른 저장 프로시저를 호출하는 중첩(nested) 호출의 깊이가 시스템에서 허용하는 최대치를 초과했을 때 발생

합니다, 이는 무한 재귀 호출이나 과도한 중첩 호출로 인해 시스템 자원이 고갈되거나 스택 오버플로우가 발생하는 것을 방지하기 위한 보호성 오류입니다.

**ERROR CODE: -914, 'A serial object already exists as an auto increment constraint.'**

- 이 메시지는 CUBRID 데이터베이스에서 자동 증가(Auto Increment) 제약조건을 생성하려고 할 때, 동일한 이름의 시리얼 객체가 이미 시스템에 존재할 때 발생합니다.

**ERROR CODE: -915, 'Increase value of an auto increment constraint should be greater than 0.'**

- 이 메시지는 CUBRID 데이터베이스에서 자동 증가(Auto Increment) 제약조건을 생성하거나 수정할 때, 증가 값이 0 이하로 설정되었을 때 발생합니다, 즉, 자동 증가 시퀀스의 증가 값은 반드시 양수여야 하며, 0이나 음수는 허용되지 않습니다. 이는 자동 증가의 논리적 의미와 수학적 일관성을 보장하기 위한 제약조건입니다, 이런 상황은 일반적으로 *CREATE TABLE* 또는 *ALTER TABLE* 문을 사용하여 자동 증가 컬럼을 정의할 때, *INCREMENT BY* 절에 0 이하의 값을 지정했을 때 발생합니다.

**ERROR CODE: -918, 'The class '%1\$s' referred by the foreign key does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건을 정의하거나 수정할 때, 외래 키가 참조하려는 부모 테이블이 데이터베이스 내에 존재하지 않을 때 발생합니다, 이런 상황은 일반적으로 *CREATE TABLE* 또는 *ALTER TABLE* 문을 사용하여 외래 키를 정의할 때, 참조하려는 부모 테이블의 이름을 잘못 지정했거나, 해당 테이블이 이미 삭제된 경우에 발생합니다, 이는 데이터베이스 설계의 논리적 일관성과 참조 무결성을 보호하기 위한 제약조건 검증 오류입니다.

**ERROR CODE: -919, 'The class '%1\$s' referred by the foreign key does not have the primary key.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건을 정의하거나 수정할 때, 외래 키가 참조하려는 부모 테이블에 기본 키(Primary Key) 제약조건이 정의되어 있지 않을 때 발생합니다, 이런 상황은 일반적으로 *CREATE TABLE* 또는 *ALTER TABLE* 문을 사용하여 외래 키를 정의할 때, 참조하려는 부모 테이블에 기본 키가 정의되지 않았거나, 기본 키가 삭제된 후에 발생합니다, 이는 데이터 무결성과 참조 무결성을 보호하기 위한 제약조건 검증 오류입니다.

**ERROR CODE: -920, 'The foreign key '%1\$s' does not include the primary key member '%2\$s'.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건을 정의하거나 수정할 때, 외래 키를 구성하는 컬럼 목록에 참조되는 기본 키(Primary Key)의 모든 컬럼이 포함되어 있지 않을 때 발생합니다, 이런 상황은 일반적으로 *CREATE TABLE* 또는 *ALTER TABLE* 문을 사용하여 외래 키를 정의할 때, 참조하는 키의 컬럼 목록을 잘못 지정하거나, 오타 등으로 인해 일부 컬럼이 누락되었을 때 발생합니다, 이는 데이터 무결성과 참조 무결성을 보호하기 위한 제약조건 검증 오류입니다.

**ERROR CODE: -921, 'The domain of the foreign key member '%1\$s' is different from that of the primary key member '%2\$s'. (%3\$s: %4\$s vs. %5\$s)'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건을 정의하거나 수정할 때, 외래 키를 구성하는 컬럼의 도메인(데이터 타입 및 관련 속성)이 참조하는 기본 키(Primary Key)를 구성하는 컬럼의 도메인과 일치하지 않을 때 발생합니다, 이런 상황은 일반적으로 *CREATE*

TABLE 또는 ALTER TABLE 문을 사용하여 외래 키를 정의할 때, 참조하는 키의 컬럼 도메인을 잘못 이해하거나 오타 등으로 인해 호환되지 않는 도메인을 지정했을 때 발생합니다, 이는 데이터 무결성과 참조 무결성을 보장하기 위한 제약조건 검증 오류입니다.

**ERROR CODE: -922, 'The constraint of the foreign key '%1\$s' is invalid, due to value '%2\$s'.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건이 유효하지 않은 값을 참조하거나, 제약조건 자체의 논리적 오류로 인해 데이터 무결성을 위반할 때 발생합니다, 즉, 외래 키 컬럼에 삽입되거나 업데이트된 값이 참조하는 부모 테이블의 기본 키(Primary Key) 또는 유니크 키(Unique Key)에 존재하지 않거나, 외래 키 제약조건 정의 자체가 잘못되어 유효성 검사를 통과하지 못했음을 의미합니다, 이런 상황은 일반적으로 INSERT 또는 UPDATE 문을 사용하여 자식 테이블에 데이터를 추가하거나 변경할 때, 부모 테이블에 존재하지 않는 값을 외래 키 컬럼에 지정하려 할 때 발생합니다. 또한, 외래 키를 정의하는 과정에서 참조하는 컬럼의 데이터 타입 불일치나 기타 스키마 불일치로 인해 발생할 수도 있습니다, 이는 데이터 무결성과 참조 무결성을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -923, 'The primary key '%1\$s' referred by a foreign key '%2\$s' is not supposed to be dropped.'**

- 이 메시지는 CUBRID 데이터베이스에서 기본 키(Primary Key)를 삭제하려고 할 때, 해당 기본 키가 다른 테이블의 외래 키(Foreign Key)에 의해 참조되고 있기 때문에 삭제 작업이 허용되지 않을 때 발생합니다, 즉, 참조 무결성(Referential Integrity)을 유지하기 위해, 자식 테이블에서 참조하고 있는 부모 테이블의 기본 키는 직접 삭제할 수 없도록 시스템이 제한하는 것입니다. 기본 키를 삭제하려면 먼저 해당 기본 키를 참조하는 모든 외래 키 제약조건을 제거해야 합니다, 이런 상황은 일반적으로 ALTER TABLE ... DROP PRIMARY KEY 또는 DROP TABLE 문을 사용하여 기본 키가 설정된 테이블을 삭제하려 할 때, 해당 기본 키를 참조하는 외래 키가 남아있을 때 발생합니다, 이는 데이터 무결성을 보호하고 데이터베이스의 논리적 일관성을 유지하기 위한 보호성 오류입니다.

**ERROR CODE: -924, 'Update/Delete operations are restricted by the foreign key '%1\$s'.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건이 설정된 테이블의 부모 레코드를 업데이트하거나 삭제하려고 할 때 발생합니다. 이때 외래 키의 ON UPDATE 또는 ON DELETE 액션이 RESTRICT 로 설정되어 있어, 자식 테이블에 해당 부모 레코드를 참조하는 레코드가 존재하는 경우 작업이 허용되지 않습니다, 즉, 참조 무결성(Referential Integrity)을 유지하기 위해, 자식 레코드가 존재하는 부모 레코드는 수정하거나 삭제할 수 없도록 시스템이 제한하는 것입니다, 이런 상황은 일반적으로 UPDATE 또는 DELETE 문을 사용하여 부모 테이블의 레코드를 변경하려 할 때, 해당 레코드를 참조하는 자식 테이블의 레코드가 남아있을 때 발생합니다, 이는 데이터 무결성을 보호하고 데이터베이스의 논리적 일관성을 유지하기 위한 보호성 오류입니다.

**ERROR CODE: -926, 'The instance having the foreign key '%1\$s' cannot be dropped.'**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건에 의해 다른 테이블에서 참조되고 있는 인스턴스(레코드)를 삭제하려고 할 때 발생합니다, 즉, 부모 테이블의 인스턴스를 삭제하려 하지만, 자식 테이블에 해당 부모 인스턴스를 참조하는 외래 키가 존재하여 참조 무결성이 깨질 수 있기 때문에 삭제 작업이 허용되지 않는다는 의미입니다, 이런 상황은 일반적으로 DELETE 문을 사용하여 부모 테이블의 레코드를 삭제할 때, 해당 레코드를 참조하는 자식 테이블의 레코드

가 남아있을 때 발생합니다, 이는 데이터 무결성을 보호하고 데이터베이스의 논리적 일관성을 유지하기 위한 보호성 오류입니다.

**ERROR CODE: -927, 'The number of keys of the foreign key '%1\$s' is different from that of the primary key '%2\$s''**

- 이 메시지는 CUBRID 데이터베이스에서 외래 키(Foreign Key) 제약조건을 정의하거나 수정할 때, 외래 키를 구성하는 컬럼의 개수가 참조하는 기본 키(Primary Key)를 구성하는 컬럼의 개수와 일치하지 않을 때 발생합니다, 즉, 외래 키는 참조하는 부모 테이블의 기본 키 또는 유니크 키와 동일한 수의 컬럼으로 구성되어야 하며, 각 컬럼의 데이터 타입도 호환되어야 합니다. 이 메시지는 주로 컬럼 개수의 불일치로 인해 발생합니다, 이런 상황은 일반적으로 `CREATE TABLE` 또는 `ALTER TABLE` 문을 사용하여 외래 키를 정의할 때, 참조하는 키의 구조를 잘못 이해하거나 오타 등으로 인해 컬럼 개수를 다르게 지정했을 때 발생합니다, 이는 데이터 무결성과 관계형 데이터베이스의 논리적 일관성을 보장하기 위한 제약조건 검증 오류입니다.

**ERROR CODE: -931, 'Start value of an auto increment constraint should be less then max value.'**

- 이 메시지는 CUBRID 데이터베이스에서 자동 증가(Auto Increment) 제약조건을 생성하거나 수정할 때, 시작 값이 최대 값보다 크거나 같을 때 발생합니다, 즉, 자동 증가 시퀀스의 시작 값이 해당 데이터 타입의 최대 허용 범위를 초과하거나, 시작 값과 최대 값이 동일한 경우에 발생하는 제약조건 위반 오류입니다, 이런 상황은 일반적으로 테이블 스키마 정의 시 잘못된 자동 증가 설정, 데이터 타입과 시작 값 간의 불일치, 또는 사용자가 명시적으로 설정한 시작 값이 데이터 타입의 한계를 초과할 때 발생합니다.

**ERROR CODE: -935, 'Argument of %1\$s function is out of range.'**

- 이 메시지는 CUBRID 데이터베이스에서 수학 함수나 시스템 함수의 인수가 해당 함수가 처리할 수 있는 유효한 범위를 벗어났을 때 발생합니다, 즉, 함수가 정의된 도메인(domain)이나 수학적으로 유효하지 않은 입력값을 받았을 때 발생하는 오류입니다, 이런 상황은 일반적으로 수학적으로 정의되지 않은 연산이나 시스템 함수의 범위를 벗어난 접근을 시도할 때 발생합니다.

**ERROR CODE: -936, 'Overflow occurred in exp context.'**

- 이 메시지는 CUBRID 데이터베이스의 쿼리 프로세서에서 다양한 데이터 타입의 값에 대해 지수 함수를 계산하는 역할할 때 매우 큰 양수나 음수 값에 대해 지수 함수를 계산할 때 발생합니다.

**ERROR CODE: -965, 'Cannot use duplicate attribute names in index key list. Attribute name '%1\$s' listed more than once.'**

- 이 메시지는 CUBRID 데이터베이스에서 인덱스를 생성하거나 수정할 때, 인덱스 키 목록에 동일한 속성(컬럼) 이름이 두 번 이상 지정되었을 때 발생합니다, CUBRID는 인덱스의 효율성과 일관성을 위해 동일한 컬럼을 인덱스 키에 중복으로 포함하는 것을 허용하지 않습니다. 이는 불필요한 중복을 방지하고, 인덱스의 성능을 최적화하며, 데이터베이스 스키마의 무결성을 유지하기 위한 보호성 오류입니다, 예를 들어, `CREATE INDEX idx_name ON table_name (col1, col2, col1)`과 같이 동일한 컬럼을 두 번 지정하면 이 오류가 발생합니다.

**ERROR CODE: -979, 'CONNECT BY loop in user data.'**

- 이 메시지는 connect by 구문에서 순환 참조에 의해 무한 루프가 발생한 경우 발생합니다.

**ERROR CODE: -996, 'Incorrect number of values given. %1\$d values were given but %2\$d values are required.'**

- 이 메시지는 CUBRID 데이터베이스에서 prepare from 구문에 사용한 host 변수 개수(?) 와 execute using 구문의 값의 개수가 맞지 않는 경우 발생합니다.

**ERROR CODE: -997, 'Cannot add the foreign key constraint %1\$s to the partitioned %2\$s. Referential actions that update attributes are not yet supported on partitioned classes.'**

- 이 메시지는 CUBRID 데이터베이스에서 파티션 테이블에 외래키 제약 조건을 추가하려고 할 때 발생합니다. CUBRID는 현재 파티션된 테이블에서 참조 동작(Referential Action)이 속성을 업데이트 하는 외래키 제약 조건을 지원하지 않습니다. 참조 동작은 ON UPDATE CASCADE, ON UPDATE SET NULL, ON UPDATE SET DEFAULT 등과 같이 부모 테이블의 키 값이 변경될 때 자식 테이블의 외래키 값을 자동으로 업데이트하는 기능입니다. 파티션된 테이블은 데이터가 여러 파티션에 분산되어 저장되므로, 이러한 복잡한 참조 동작을 수행할 때 성능 문제나 데이터 일관성 문제가 발생할 수 있어 CUBRID에서 제한하고 있습니다.

**ERROR CODE: -998, 'Cannot add the foreign key constraint %1\$s on the attribute %2\$s. The attribute cannot have a NOT NULL constraint because it has to match the SET NULL referential action.'**

- 이 메시지는 SET NULL을 참조 작업으로 사용하는 외래 키를 정의할 때, 해당 컬럼에 NOT NULL 제약조건인 경우 발생합니다. SET NULL 참조 작업은 참조 무결성을 위해 NULL 값을 허용해야 하므로, NOT NULL 제약조건과 논리적으로 충돌합니다.

**ERROR CODE: -999, 'Cannot create table '%1\$s' like table '%2\$s' because the existing table has features that will not be duplicated (%3\$s).'**

이 오류 코드는 CREATE TABLE LIKE 구문 사용 시, 원본 테이블이 가진 특정 기능이나 속성을 새 테이블에 복제할 수 없을 때 발생합니다.

**ERROR CODE: -1000, '%1\$s' is a CLASS or SHARED attribute but only normal attributes are allowed in this context.'**

- 이 메시지는 CREATE TABLE AS SELECT 구문에 발생하는 오류로 column 속성에 shared 또는 class 속성을 사용할 수 없다는 오류입니다. 예를들면, CREATE TABLE x (name varchar); CREATE TABLE y (name varchar shared '1111') as SELECT name FROM x; 이와 같은 DDL(shared 복제) 구문 형식을 사용할 경우 발생할 수 있습니다.

**ERROR CODE: -1003, 'The constraint '%1\$s' exists but has a different type.'**

- 이 메시지는 데이터베이스 스키마 관리 중에 발생하는 제약조건 타입 불일치 오류로 primary key 를 `alter table .. drop index`로 삭제하려고 할때 발생합니다.

**ERROR CODE: -1004, 'Please note that when using ALTER TABLE DROP FOREIGN KEY, CUBRID drops both the constraint and its associated index.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 `ALTER TABLE DROP FOREIGN KEY` 문을 실행할 때 사용자에게 제공하는 알림(notification)입니다, 이는 CUBRID의 외래키 관리 시스템의 특성을 설명하는 정보성 메시지이며, 오류가 아닌 사용자에게 주의사항을 알려주는 알림입니다.

**ERROR CODE: -1011, 'Cannot add the foreign key constraint %1\$s on the shared attribute %2\$s.'**

이 오류 코드는 SHARED 속성에 외래 키 제약조건을 추가할 수 없음을 나타냅니다.

**ERROR CODE: -1012, 'Cannot create table with NULL column type.'**

이 오류 코드는 `CREATE TABLE AS SELECT` 구문 실행 시 NULL 타입의 컬럼이 포함된 경우 발생하는 오류입니다, 예를들면 `CREATE TABLE tbl AS SELECT NULL AS a` 이와 같은 DDL 복제 구문 형식을 사용할 경우 발생할 수 있습니다.

**ERROR CODE: -1013, 'Set DEFAULT or SHARED on %1\$s typed column is not allowed.'**

이 오류 코드는 BLOB, CLOB 데이터 타입의 컬럼에 DEFAULT 값이나 SHARED 속성을 설정할 수 없음을 나타냅니다.

**ERROR CODE: -1014, 'Set NOT NULL on %1\$s typed column is not allowed.'**

이 오류 코드는 BLOB, CLOB 데이터 타입의 컬럼에 NOT NULL 제약조건을 설정할 수 없음을 나타냅니다.

**ERROR CODE: -1041, 'A string was truncated in context of "%1\$s".'**

- 이 메시지는 CUBRID에서 쿼리 처리(Query Processing)나 데이터 저장/조회 과정 중에 문자열 값이 할당된 버퍼나 대상 컬럼의 크기보다 커서 뒷부분이 잘렸음을 나타냅니다, 이는 `ALTER TABLE` 에서 컬럼의 문자열 크기를 줄이는 경우나 `group_concat` 함수와 같이 문자열 병합 시 발생할 수 있는 경고성 오류 메세지입니다.

**ERROR CODE: -1042, 'Trying to create a string requiring %1\$d bytes of memory, while the maximum allowed is %2\$d bytes.'**

- 이 메시지는 CUBRID 쿼리 처리 중 특정 문자열 연산(주로 문자열 함수)의 결과로 생성되는 문자열의 크기가 시스템에서 허용하는 단일 문자열 객체의 최대 크기 제한을 초과했을 때 발생합니다, 이는 `$CUBRID/conf/cubrid.conf` 환경설정에서 `string_max_size_bytes` 파라미터 설정값 연관성이 매우 높습니다.

**ERROR CODE: -1044, 'Cannot change attribute "%1\$s". CUBRID cannot change attributes that are partitioning keys.'**

- 이 메시지는 사용자가 `ALTER TABLE ... CHANGE` 구문을 사용하여 테이블의 속성 정의를 변경하려고 시도했지만, 해당 속성이 테이블 파티셔닝(Partitioning)의 키(Key)로 사용되고 있기 때문에 변경이 거부되었음을 의미합니다, CUBRID는 파티션된 테이블의 데이터 분산 규칙과 무결성을 유지



하기 위해, 파티셔닝 키로 사용되는 컬럼의 정의(예: 데이터 타입, 제약 조건 등)를 변경하는 것을 허용하지 않습니다. 파티셔닝 키는 데이터가 어떤 파티션에 저장될지를 결정하는 기준이므로, 이 키 컬럼의 정의가 변경되면 기존 파티션 구조 전체가 무효화되고 데이터 정합성이 깨질 수 있기 때문입니다.

**ERROR CODE: -1045, 'Cannot change attribute "%1\$s". CUBRID cannot change attributes used in foreign keys (either referenced or referencing).'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼을 변경하려고 할 때 발생합니다. 해당 컬럼이 외래키(Foreign Key) 관계에서 사용되고 있는 경우, CUBRID는 데이터 무결성과 참조 일관성을 보장하기 위해 이러한 변경을 허용하지 않습니다. 외래키는 데이터베이스의 참조 무결성을 유지하는 중요한 제약 조건으로, 참조되는 컬럼(Parent Table의 Primary Key 또는 Unique Key)과 참조하는 컬럼(Child Table의 Foreign Key) 간의 관계를 정의합니다. 이러한 컬럼의 변경은 참조 무결성을 깨뜨릴 수 있으므로 CUBRID에서 제한하고 있습니다.

**ERROR CODE: -1046, 'Cannot change attribute "%1\$s". CUBRID cannot change the attribute's domain. This change of domain is not supported.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼의 데이터 타입을 변경하려고 할 때 발생합니다. CUBRID가 현재 해당 데이터 타입 간의 변환을 지원하지 않는 경우에 발생합니다. 모든 데이터 타입 변환이 항상 가능한 것은 아니며, 특히 복잡하거나 호환되지 않는 데이터 타입 간의 변경은 데이터 무결성이나 시스템 안정성에 문제를 일으킬 수 있어 CUBRID에서 명시적으로 지원하지 않을 수 있습니다. 이는 데이터베이스의 안정성과 일관성을 유지하기 위한 CUBRID의 설계상의 제한입니다.

**ERROR CODE: -1047, 'Cannot change attribute "%1\$s". CUBRID cannot change the attribute's domain. The new domain may not correctly represent all the existing attribute values.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼의 데이터 타입을 변경하려고 할 때 발생합니다. CUBRID가 기존 컬럼에 저장된 데이터를 분석한 결과, 요청된 새로운 데이터 타입으로 변환할 수 없는 값이 존재하거나, 데이터 손실이 발생할 가능성이 높다고 판단했을 때 발생합니다. 이는 데이터 무결성을 보장하고 잠재적인 데이터 손실을 방지하기 위한 CUBRID의 보호 메커니즘입니다. 예를 들어, `VARCHAR` 컬럼에 숫자가 아닌 문자열이 포함되어 있는데 이를 `INT` 타입으로 변경하려고 할 때, 또는 `INT` 컬럼에 `BIGINT` 범위를 초과하는 값이 있는데 이를 `INT`로 변경하려고 할 때 발생할 수 있습니다.

**ERROR CODE: -1048, 'Cannot change attribute "%1\$s". CUBRID cannot currently change an attribute's AUTO\_INCREMENT if the attribute's domain also changes.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE` 문을 사용하여 특정 컬럼의 `AUTO_INCREMENT` 속성과 데이터 타입을 동시에 변경하려고 할 때 발생합니다. CUBRID는 데이터 무결성을 유지하고 스키마 변경의 복잡성을 줄이기 위해 이러한 동시 변경 작업을 허용하지 않습니다. `AUTO_INCREMENT` 속성은 값 생성 로직과 관련이 있고, 데이터 타입 변경은 기존 데이터의 변환 및 저장 방식과 관련이 있으므로, 이 두 가지를 한 번에 변경하는 것은 잠재적인 데이터 손

실, 일관성 문제 또는 예측 불가능한 동작을 초래할 수 있습니다. 따라서 CUBRID는 이러한 변경을 별도의 단계로 나누어 수행하도록 요구합니다.

**ERROR CODE: -1049, ‘Cannot change attribute “%1\$s”. CUBRID cannot currently change the attribute’s domain. The domain conversion is possible but this feature is not allowed by configuration.’**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼의 데이터 타입을 변경하려고 할 때 발생합니다. CUBRID가 기술적으로는 해당 도메인 변환이 가능하다고 판단했지만, 현재 시스템 설정에서 이 기능을 허용하지 않도록 구성되어 있어 발생하는 제한적 오류입니다. 이는 데이터 타입 변환의 복잡성, 잠재적인 데이터 손실 위험, 또는 시스템 성능에 미칠 수 있는 영향을 고려하여 관리자가 의도적으로 제한한 기능일 가능성이 높습니다, 이는 `$CUBRID/conf/cubrid.conf` 환경설정에서 `alter_table_change_type_strict` 설정 값이 `true` 인 경우 발생할 수 있습니다.

**ERROR CODE: -1050, ‘Cannot change attribute “%1\$s”. CUBRID cannot currently change the attribute’s domain. The attribute has an index on it.’**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼의 데이터 타입을 변경하려고 할 때 발생합니다. 특히, 변경하려는 컬럼에 이미 인덱스(Index)가 생성되어 있는 경우 CUBRID는 데이터 무결성 및 인덱스 구조의 복잡성 때문에 해당 도메인 변경을 허용하지 않습니다. 인덱스는 특정 컬럼의 값에 기반하여 데이터를 빠르게 검색할 수 있도록 돕는 구조인데, 컬럼의 데이터 타입이 변경되면 인덱스 구조 자체가 유효하지 않게 되거나 재구축이 필요하게 됩니다. CUBRID는 이러한 복잡성과 잠재적인 데이터 손상을 방지하기 위해 인덱스가 있는 컬럼의 도메인 변경을 직접적으로 제한합니다.

**ERROR CODE: -1052, ‘Attribute “%1\$s” was not changed. CUBRID did not detect any effective changes in the attribute representation.’**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 또는 `ALTER TABLE ... MODIFY` 문을 사용하여 특정 컬럼의 속성을 변경하려고 시도했지만, CUBRID가 요청된 변경 사항이 기존 컬럼의 정의와 실질적으로 동일하다고 판단했을 때 발생하는 경고성 알림입니다. 즉, 사용자가 변경하려는 컬럼의 데이터 타입, 길이, 제약 조건(NOT NULL, DEFAULT 등) 등이 현재 컬럼의 정의와 동일하여 실제적인 변경이 발생하지 않았음을 의미합니다. 이는 오류가 아니며, 시스템이 불필요한 작업을 수행하지 않았음을 알려주는 정보성 메시지입니다.

**ERROR CODE: -1054, ‘The new value of an auto increment constraint should be greater than or equal to the current value.’**

이 메시지 CUBRID 데이터베이스 시스템에서 `ALTER TABLE` 문을 사용하여 자동 증가(AUTO\_INCREMENT) 제약조건의 값을 변경하려고 할 때, 변경 값이 현재 값보다 작은 경우에 발생합니다, 즉, CUBRID 서버가 `ALTER TABLE ... AUTO_INCREMENT = x` 문을 처리하는 과정에서, 사용자가 지정한 새로운 값이 해당 컬럼의 현재 자동 증가 값보다 작을 때 발생합니다. 이는 데이터 무결성을 보호하기 위한 제약 조건으로, 자동 증가 값이 감소하면 기존 데이터와의 일관성 문제가 발생할 수 있기 때문입니다, 이는 CUBRID의 자동 증가 시스템에서 데이터 일관성을 유지하기 위한 보호성 오류입니다.



**ERROR CODE: -1055, 'The new value of an auto increment constraint should be lower than the maximum value.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE` 문을 사용하여 `AUTO_INCREMENT` 값을 최대 값보다 크게 설정하려고 할 때 발생합니다. 특히, 사용자가 지정한 새로운 `AUTO_INCREMENT` 시작 값(예: `ALTER TABLE table_name AUTO_INCREMENT = N;` 에서 `N` 값)이 해당 컬럼의 데이터 타입이 허용하는 최대값보다 큰 경우 발생합니다. `AUTO_INCREMENT` 값은 항상 증가해야 하며, 데이터 타입의 범위를 초과하거나 기존 데이터의 최대값을 침범할 수 없으므로, CUBRID는 데이터 무결성을 보장하기 위해 이러한 설정을 거부합니다.

**ERROR CODE: -1056, 'To avoid ambiguity, the AUTO\_INCREMENT table option requires the table to have exactly one AUTO\_INCREMENT column and no seed/increment specification.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블 생성 또는 변경시 `auto_increment` 컬럼이 두개 이상인 경우, `auto_increment=seed` 값을 주려고 할때, 어떤 `auto_increment` 컬럼을 사용할지 판단하기 힘들어서 발생하는 오류입니다. 참고로 한 테이블내에 여러개의 `auto_increment` 컬럼을 설정할 수 있습니다.

**ERROR CODE: -1057, 'Cannot add a NOT NULL '%1\$s' typed attribute without a DEFAULT value.'**

- 이 메시지는 CUBRID 데이터베이스에서 `CREATE TABLE` 또는 `ALTER TABLE` 문을 사용하여 `NOT NULL` 제약 조건을 가진 타입 속성을 추가하려고 경우나 컬럼의 타입의 `default` 값을 지정할 수 없는 경우 발생합니다.

**ERROR CODE: -1058, 'ALTER TABLE .. CHANGE : changing to new domain : cast failed, current configuration doesn't allow truncation or overflow.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 문을 사용하여 컬럼의 데이터 타입을 변경할 때 발생합니다. 기존 컬럼의 데이터를 새로운 데이터 타입으로 변환(캐스팅)하는 과정에서 일부 데이터가 새로운 도메인의 범위를 초과하거나 잘림이 발생할 수 있는 상황이 감지되었습니다. 현재 CUBRID의 설정에서는 데이터 손실을 방지하기 위해 이러한 변환을 허용하지 않도록 구성되어 있습니다. 이는 데이터 무결성을 보장하고 의도치 않은 데이터 손실을 방지하기 위한 CUBRID의 보호 메커니즘입니다.

**ERROR CODE: -1059, 'ALTER TABLE .. CHANGE : changing to new domain : cast failed, default value of new domain was set.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 문을 사용하여 컬럼의 데이터 타입(숫자, 문자열이 아닌 경우)을 변경할 때 발생하는 정보성 알림입니다. 기존 컬럼의 데이터를 새로운 데이터 타입으로 변환(캐스팅)하는 과정에서 일부 데이터가 새로운 도메인의 범위에 맞지 않거나 변환할 수 없는 경우, CUBRID는 해당 데이터를 새로운 도메인의 "기본값(default value)"으로 자동 설정합니다. 이는 데이터 손실을 최소화하면서 스키마 변경을 완료하기 위한 CUBRID의 자동 데이터 변환 메커니즘입니다. 예를 들어, `VARCHAR` 컬럼을 `INT`로 변경할 때 숫자로 변환할 수 없는 문자열은 `INT`의 기본값인 0으로 설정될 수 있습니다. 이 메시지는 시스템이 데이터 일관성을 위해 성공적으로 수행한 자동 변환 작업에 대한 보고입니다.

**ERROR CODE: -1060, 'ALTER TABLE .. CHANGE : changing to new domain : cast failed, minimum value of new domain was set.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 문을 사용하여 컬럼의 데이터 타입(도메인 값이 음수인 경우)을 변경할 때 발생하는 정보성 알림입니다. 기존 컬럼의 데이터를 새로운 데이터 타입으로 변환(캐스팅)하는 과정에서 일부 데이터가 새로운 도메인의 범위를 초과하거나 변환할 수 없는 경우, CUBRID는 해당 데이터를 새로운 도메인의 최소값으로 자동 설정합니다. 이는 데이터 손실을 최소화하면서 스키마 변경을 완료하기 위한 CUBRID의 자동 데이터 변환 메커니즘입니다.

**ERROR CODE: -1061, 'ALTER TABLE .. CHANGE : changing to new domain : cast failed, maximum value of new domain was set.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 문을 사용하여 컬럼의 데이터 타입(도메인 값이 양수인 경우)을 변경할 때 발생하는 정보성 알림입니다. 기존 컬럼의 데이터를 새로운 데이터 타입으로 변환(캐스팅)하는 과정에서 일부 데이터가 새로운 도메인의 범위를 초과하거나 변환할 수 없는 경우, CUBRID는 해당 데이터를 새로운 도메인의 최대값으로 자동 설정합니다. 이는 데이터 손실을 최소화하면서 스키마 변경을 완료하기 위한 CUBRID의 자동 데이터 변환 메커니즘입니다.

**ERROR CODE: -1062, 'Cannot change attribute "%1\$s". The change would result in multiple primary keys being defined.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE` 문을 사용하여 특정 컬럼을 변경하려고 할 때 발생합니다. 요청된 변경 사항이 테이블에 여러 개의 기본키(Primary Key)가 정의되도록 할 경우, CUBRID는 데이터 무결성과 테이블 구조의 일관성을 보장하기 위해 해당 변경을 거부합니다. 관계형 데이터베이스에서는 각 테이블에 하나의 기본키만 허용되며, 여러 개의 기본키가 존재하면 데이터의 고유성과 참조 무결성에 문제가 발생할 수 있습니다. 이는 데이터베이스 설계의 기본 원칙을 위반하는 상황입니다.

**ERROR CODE: -1063, 'Cannot add NOT NULL constraint for attribute "%1\$s": there are existing NULL values for this attribute.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE` 문을 사용하여 특정 컬럼에 `NOT NULL` 제약 조건을 추가하려고 할 때 발생합니다. 해당 컬럼에 이미 하나 이상의 `NULL` 값이 존재하는 경우, CUBRID는 데이터 무결성을 보장하기 위해 `NOT NULL` 제약 조건 추가를 거부합니다. `NOT NULL` 제약 조건은 컬럼에 `NULL` 값이 저장되는 것을 완전히 금지하므로, 기존의 `NULL` 값들을 먼저 처리하거나 제거해야 합니다. 이는 데이터베이스의 일관성과 무결성을 보호하기 위한 CUBRID의 안전장치입니다.

**ERROR CODE: -1064, 'ALTER TABLE .. CHANGE : adding NOT NULL : all existing NULL values were changed to domain's hard default.'**

- 이 메시지는 CUBRID 데이터베이스에서 `ALTER TABLE ... CHANGE` 문을 사용하여 특정 컬럼에 `NOT NULL` 제약 조건을 추가할 때 발생하는 정보성 알림입니다. 해당 컬럼에 이미 `NULL` 값이 존재하고, 사용자가 명시적인 `DEFAULT` 값을 지정하지 않은 경우, CUBRID는 데이터 무결성을 유지하기 위

해 기존의 모든 `NULL` 값들을 해당 컬럼 데이터 타입의 “하드 기본값(hard default value)”으로 자동 변환합니다. 예를 들어, 숫자 타입은 0, 문자열 타입은 빈 문자열(“”) 등으로 변경됩니다. 이 메시지는 오류가 아니라, 시스템이 데이터 일관성을 위해 성공적으로 수행한 자동 변환 작업에 대한 보고입니다.

**ERROR CODE: -1067, ‘Column ‘%1\$s’ cannot have a NOT NULL constraint and a NULL default value.’**

- 이 메시지는 CUBRID 데이터베이스에서 테이블을 생성하거나 변경할 때, 특정 컬럼에 NOT NULL 제약 조건을 부여하면서 동시에 NULL을 기본값으로 설정하려고 시도할 때 발생합니다. NOT NULL 제약 조건은 해당 컬럼에 NULL 값이 저장되는 것을 허용하지 않으므로, NULL 기본값은 이 제약 조건과 논리적으로 충돌하여 허용되지 않습니다. 이는 데이터 무결성을 보장하고 데이터 정의의 일관성을 유지하기 위한 시스템의 보호 메커니즘입니다.

**ERROR CODE: -1068, ‘Maximum trigger depth cannot exceed %1\$d.’**

- 이 메시지는 CUBRID가 허용하는 최대 트리거 깊이 제한을 초과할 때 발생합니다. CUBRID는 트리거의 무한 루프나 과도한 리소스 소모를 방지하기 위해 트리거 호출의 최대 깊이를 제한하며, 이 메시지는 설정 자체가 시스템의 내부적인 한계를 넘어섰음을 의미합니다. 이는 데이터베이스의 안정성과 성능을 보호하기 위한 중요한 보호성 오류입니다.

**ERROR CODE: -1080, ‘The maximum length of filter predicate string must be %1\$d.’**

- 이 메시지는 CUBRID 데이터베이스에서 function index 생성시 발생하는 오류로 function에 사용된 인자(수식)의 길이가 시스템 내부에서 정한 길이보다 큰 경우 발생합니다.

**ERROR CODE: -1085, ‘Cannot compile regular expression: ‘%1\$s’**

- 이 메시지는 CUBRID 데이터베이스에서 정규 표현식(Regular Expression)을 컴파일하는 과정에서 문제가 발생하여 정상적으로 처리할 수 없을 때 나타납니다. 이는 주로 정규 표현식 자체의 문법 오류, 유효하지 않은 패턴, 또는 정규 표현식 엔진의 내부적인 문제로 인해 발생할 수 있습니다. 데이터베이스가 쿼리 내에서 정규 표현식 함수를 처리하기 전에 패턴을 컴파일할 때 이 오류가 발생하면 해당 쿼리 실행이 실패하게 됩니다.

**ERROR CODE: -1086, ‘Cannot execute regular expression: ‘%1\$s’**

- 이 메시지는 CUBRID 데이터베이스에서 정규 표현식(Regular Expression)을 실행하는 과정에서 문제가 발생하여 정상적으로 처리할 수 없을 때 나타납니다. 이는 주로 정규 표현식 자체의 문법 오류, 유효하지 않은 패턴, 또는 정규 표현식 엔진의 내부적인 문제로 인해 발생할 수 있습니다. 데이터베이스가 쿼리 내에서 정규 표현식 함수를 처리할 때 이 오류가 발생하면 해당 쿼리 실행이 실패하게 됩니다.

**ERROR CODE: -1092, ‘Cannot prepare statement of type ‘%1\$s’ with host variables.’**

- 이 메시지는 CUBRID 데이터베이스에서 특정 타입의 SQL 문장을 호스트 변수와 함께 PREPARE 문으로 준비하려고 할 때 발생합니다. PREPARE 문은 SQL 문장을 미리 컴파일하여 실행 계획을 수립하는 기능으로, 호스트 변수를 사용하여 동적으로 SQL을 구성할 수 있습니다. 하지만 일부 특정 타

입의 문장(예: DDL 문, 특정 시스템 명령어 등)은 호스트 변수와 함께 사용할 수 없도록 제한되어 있습니다. 이는 데이터베이스의 보안과 안정성을 보호하기 위한 제약 조건입니다.

**ERROR CODE: -1096, 'Altering partitioning schema is not allowed when the partitioned class is referenced by a foreign key.'**

- 이 메시지는 CUBRID 데이터베이스에서 파티션된 테이블의 파티셔닝 스키마를 변경하려고 할 때, 해당 테이블이 다른 테이블의 외래 키 제약 조건에 의해 참조되고 있는 경우 발생합니다. 데이터 무결성을 보호하기 위해 외래 키 관계가 존재하는 파티션 테이블의 구조 변경을 제한하는 보호 메커니즘입니다.

**ERROR CODE: -1097, 'Column '%1\$s' cannot have a PRIMARY KEY constraint and a NULL default value.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블을 생성하거나 변경할 때, 특정 컬럼에 PRIMARY KEY 제약 조건을 부여하면서 동시에 NULL을 기본값으로 설정하려고 시도할 때 발생합니다. PRIMARY KEY는 고유성과 NOT NULL 속성을 필수적으로 요구하므로, NULL 기본값은 이 제약 조건과 충돌하여 허용되지 않습니다. 이는 데이터 무결성을 보장하기 위한 시스템의 보호 메커니즘입니다.

**ERROR CODE: -1098, 'Locales for language '%1\$s' are not available with charset '%2\$s'.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜/시간 포매팅 함수(예: DATE\_FORMAT, TIME\_FORMAT)를 사용할 때 발생하는 오류입니다, 즉, 특정 언어와 문자셋의 조합에 대한 로케일 데이터가 시스템에 설치되어 있지 않거나 지원되지 않을 때 발생합니다, 이는 다국어 지원과 관련된 오류로, 날짜나 시간을 특정 언어 형식으로 출력하려고 할 때 해당 언어의 로케일 정보가 현재 문자셋과 호환되지 않는 경우 발생합니다.

**ERROR CODE: -1106, 'Multiple rows in source table match the same row in destination table.'**

- 이 메시지는 CUBRID 데이터베이스에서 group by 처리 중 결과에 2개 이상의 동일한 record가 존재할 경우와 sort 후 결과를 병합하는 과정에서 발생하는 오류입니다.

**ERROR CODE: -1107, 'Invalid bucket number for NTILE. Should be greater than zero and no more than 2,147,483,647.'**

- 이 메시지는 CUBRID 데이터베이스에서 SQL 윈도우 함수인 NTILE()을 사용할 때, 인수로 지정된 버킷(그룹)의 개수가 유효한 범위를 벗어났을 때 발생합니다, 즉, NTILE(n)에서 n 값이 0보다 작거나 같거나, 또는 CUBRID의 정수형이 표현할 수 있는 최대값(2,147,483,647)을 초과하는 경우에 이 오류가 발생합니다. NTILE 함수는 행을 균등하게 나누기 위해 양의 정수 버킷 개수를 필요로 하며, 시스템의 정수형 한계를 넘어서는 값을 처리할 수 없습니다, 이는 NTILE 함수의 올바른 사용법과 데이터 타입의 한계를 벗어나는 입력을 방지하기 위한 보호성 오류입니다.

**ERROR CODE: -1108, 'Invalid bucket number for WIDTH\_BUCKET. Should be greater than zero and less than 2,147,483,647.'**

- 이 메시지는 CUBRID 데이터베이스에서 WIDTH\_BUCKET 함수의 네 번째 인자(버킷 수)가 유효한 범위를 벗어났을 때 발생합니다, 즉, WIDTH\_BUCKET 함수는 데이터를 지정된 개수의 균등한 너비 버킷으로 분할하는 함수인데, 버킷 수가 1보다 작거나 32비트 정수의 최대값(2,147,483,647) 이상일 때 발생합니다, 이는 함수의 정상적인 동작을 보장하기 위한 보호성 오류입니다, WIDTH\_BUCKET 함수는 주로 데이터 분석이나 통계 작업에서 사용되며, 버킷 수가 너무 크거나 작으면 메모리 사용량이나 성능에 문제가 발생할 수 있습니다.

**ERROR CODE: -1109, 'Invalid value for partition definition.'**

- 이 메시지는 CUBRID 데이터베이스에서 파티션 테이블의 분할 키(partition key) 값이 파티션 정의와 일치하지 않을 때 발생합니다, 즉, 데이터를 특정 파티션에 삽입하거나 업데이트하려고 할 때, 해당 데이터의 분할 키 값이 어떤 파티션의 범위에도 속하지 않거나 파티션 정의에 맞지 않는 경우 발생합니다, 이는 파티션 테이블의 데이터 무결성을 보장하기 위한 보호성 오류입니다, 주로 INSERT, UPDATE 작업에서 분할 키 값이 파티션 정의의 범위를 벗어나거나, 파티션 정의가 잘못되어 데이터를 적절한 파티션에 배치할 수 없을 때 발생합니다.

**ERROR CODE: -1118, 'The argument of "%1\$s" can not be coerced to desired domain "%2\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 함수나 연산의 인자를 목표 도메인 타입으로 변환할 수 없을 때 발생합니다, 즉, 데이터 타입 변환(타입 캐스팅) 과정에서 호환되지 않는 데이터 타입 간의 변환을 시도했음을 의미합니다, 이는 주로 집계 함수(MEDIAN, AVG, SUM 등)나 산술 연산에서 서로 다른 데이터 타입 간의 연산을 수행할 때 발생합니다, 시스템은 DOUBLE, DATETIME, TIME 등의 호환 가능한 타입으로 자동 변환을 시도하지만, 변환이 불가능한 경우 이 오류를 발생시킵니다.

**ERROR CODE: -1121, 'NOT NULL constraints do not allow NULL value.'**

- 이 메시지는 CUBRID 데이터베이스에서 data insert 후 index 생성시 NOT NULL 제약 조건에 NULL 데이터가 존재하는 경우 발생합니다, 이는 데이터베이스의 제약조건 위반으로 인한 보호성 오류입니다, 이 메시지는 주로 인덱스 생성, 데이터 삽입, 업데이트 등의 작업 중에 발생하며, 특히 B-tree 인덱스 로딩 과정에서 NOT NULL 제약조건이 있는 컬럼에 NULL 값이 포함된 데이터를 처리하려고 할 때 발생합니다.

**ERROR CODE: -1124, 'Query execution error. ERROR\_CODE = %1\$d, /\* SQL\_ID: %2\$s \*/ %3\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 SQL 쿼리 실행 중에 발생한 일반적인 실행 오류를 나타냅니다, 즉, 쿼리 파싱이나 컴파일은 성공했지만, 실제 실행 단계에서 문제가 발생했음을 의미합니다, 이 메시지는 쿼리 실행 엔진에서 발생하는 내부 오류, 데이터 타입 변환 오류, 메모리 부족, 시스템 리소스 문제 등 다양한 상황에서 발생할 수 있습니다.

**ERROR CODE: -1137, 'The query has been rejected due to attempt to exceed the maximum allowed nesting level(%1\$d).'**

- 이 메시지는 SQL 쿼리의 중첩 레벨(서브쿼리, 재귀 쿼리 등)이 시스템에서 허용하는 최대 깊이(기본값: 400)를 초과했을 때 발생합니다. 시스템 자원을 보호하고 무한 재귀를 방지하기 위한 안전장치입니다.

**ERROR CODE: -1150, 'Context requires compatible collations.'**

- 이 메시지는 CUBRID 데이터베이스에서 문자열 데이터를 처리하는 과정(예: 비교, 정렬, 조인 등)에서 서로 호환되지 않는 콜레이션(Collation)이 사용되거나 요구될 때 발생합니다, 즉, 데이터베이스 시스템이 특정 연산을 수행하기 위해 필요한 문자열 데이터의 콜레이션이 현재 컨텍스트에서 기대하는 콜레이션과 일치하지 않음을 감지했을 때 발생하는 오류입니다, 이런 상황은 일반적으로 서로 다른 콜레이션 설정을 가진 테이블이나 컬럼 간의 연산, 또는 잘못된 콜레이션 지정으로 인해 발생할 수 있습니다, 이는 데이터의 일관성과 정확한 문자열 처리를 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -1153, 'Attempt to use 'zero date''**

- 이 메시지는 CUBRID 데이터베이스에서 날짜/시간 타입의 데이터가 'zero date'(값이 0인 날짜)일 때 산술 연산이나 함수를 수행하려고 할 때 발생합니다, 즉, TIMESTAMP, DATE, DATETIME 등의 날짜/시간 타입에서 내부적으로 0으로 저장된 값(유효하지 않은 날짜)에 대해 연산을 시도할 때 발생하는 검증 오류입니다, 이런 상황은 일반적으로 날짜/시간 데이터가 초기화되지 않았거나, 잘못된 값이 저장되었거나, NULL 대신 0이 저장된 경우에 발생합니다.

**ERROR CODE: -1169, 'Partition key attributes must be present in the index key.'**

- 이 메시지는 CUBRID 데이터베이스에서 파티션 테이블에 유니크(unique) 인덱스를 생성하거나 수정할 때 발생합니다. CUBRID는 파티션된 테이블의 유니크 인덱스에 대해 특별한 제약 조건을 가집니다. 즉, 유니크 인덱스를 구성하는 컬럼(인덱스 키)에는 해당 테이블의 파티션 키로 사용되는 모든 컬럼이 반드시 포함되어야 합니다, 이 제약은 데이터의 유니크성을 파티션 경계를 넘어 전체 테이블에서 보장하기 위함입니다. 만약 유니크 인덱스를 정의할 때 파티션 키 컬럼이 인덱스 키에 포함되지 않으면, 데이터베이스는 이 오류를 발생시켜 인덱스 생성을 거부합니다.

**ERROR CODE: -1193, 'The length of the processed partition expression exceeds the maximum allowed(%1\$d).'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블을 파티셔닝할 때 사용되는 파티션 표현식(예: RANGE, LIST, HASH 파티션 키로 사용되는 컬럼 또는 함수)의 길이가 시스템에서 허용하는 최대 길이를 초과할 때 발생합니다. 파티션 표현식은 내부적으로 처리되고 저장되어야 하므로, 너무 길면 시스템 자원 소모나 내부적인 제약으로 인해 문제가 발생할 수 있습니다. 이는 주로 CREATE TABLE ... PARTITION... 문을 실행할 때 발생하며, 파티션 키로 사용된 표현식이 복잡하거나 문자열이 긴 경우에 나타날 수 있습니다.



**ERROR CODE: -1195, 'The recursive part of a CTE should not contain aggregate functions without GROUP BY clause.'**

- 이 메시지는 CUBRID 데이터베이스에서 재귀적 CTE(Common Table Expression)를 정의할 때, 재귀 멤버(recursive member) 부분에 GROUP BY 절 없이 집계 함수(예: SUM, COUNT, AVG, MAX, MIN)를 사용했을 때 발생합니다. 재귀적 CTE는 반복적으로 자신을 참조하여 결과를 생성하는데, 재귀 부분에서 집계 함수를 GROUP BY 없이 사용하면 각 반복 단계에서 예상치 못한 결과나 무한 루프를 초래할 수 있어 CUBRID가 이를 문법적으로 제한합니다. 이는 SQL 표준 또는 특정 데이터베이스 시스템의 재귀 쿼리 처리 규칙에 따른 제약 사항입니다.

**ERROR CODE: -1196, 'Maximum recursions %1\$d reached executing CTE.'**

- 이 메시지는 CUBRID 데이터베이스에서 재귀적 CTE(Common Table Expression)를 실행하는 동안, 쿼리가 허용된 최대 재귀 횟수를 초과했을 때 발생합니다. 재귀적 CTE는 계층적 데이터를 처리하거나 반복적인 계산을 수행할 때 사용되는데, 무한 루프를 방지하고 시스템 자원을 보호하기 위해 데이터베이스 시스템은 기본적으로 최대 재귀 횟수를 제한합니다. 이 제한에 도달하면 쿼리 실행이 중단되고 이 오류 메시지가 반환됩니다.

**ERROR CODE: -1197, 'Invalid json: %1\$s (%2\$d)'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 형식의 데이터를 파싱하거나 처리하는 과정에서, 입력된 JSON 문자열이 유효한 JSON 문법을 따르지 않을 때 발생합니다. JSON 파서가 문자열을 분석하는 중에 문법 오류를 감지하면, 오류의 종류와 발생 위치를 포함한 상세한 정보와 함께 이 오류를 반환합니다. 이는 주로 JSON 데이터를 인자로 받는 함수 호출, JSON 형식의 설정 파일 로드, 또는 JSON 데이터를 포함하는 컬럼에 값을 삽입할 때 발생할 수 있습니다.

**ERROR CODE: -1199, 'The provided JSON has been invalidated by the JSON schema (Invalid schema path: %1\$s, Keyword: %2\$s, Invalid provided JSON path: %3\$s)'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON 형식이 맞지 않은 경우 발생합니다.

**ERROR CODE: -1200, 'Invalid JSON path'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON 경로 표현식(JSON Path Expression)의 문법이 유효하지 않을 때 발생합니다. JSON 경로를 사용하여 JSON 문서 내의 특정 요소를 선택하거나 접근하려고 할 때, 제공된 경로 문자열이 JSON Path 표준 또는 CUBRID가 지원하는 문법 규칙을 따르지 않으면 이 오류가 반환됩니다. 이는 주로 JSON\_EXTRACT, JSON\_SET, JSON\_INSERT, JSON\_REPLACE 등 JSON 경로를 인자로 받는 함수를 사용할 때 발생할 수 있습니다.

**ERROR CODE: -1201, 'Json object name cannot be NULL.'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 객체를 생성하거나 조작할 때, 객체의 멤버(속성) 이름이 NULL로 지정되었을 때 발생합니다. JSON 표준에 따르면 객체의 멤버 이름은 반드시 문자열이어야 하며 NULL이 될 수 없습니다. 이 메시지는 주로 JSON\_OBJECT 함수나 JSON\_OBJECTAGG 집계 함수를 사용할 때, 키(key) 인자로 NULL 값이 전달되거나 키로 사용되는 표현식의 결과가 NULL 일 때 나타날 수 있습니다.

**ERROR CODE: -1202, 'Json PATH "%1\$s" does not exist in %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON 함수들 사용시 존재하지 않은 json path 입력 시 발생합니다.

**ERROR CODE: -1203, 'Invalid JSON type at path %1\$s. Expected %2\$s, but found %3\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON\_INSERT 또는 JSON\_SET 함수 사용시 json object가 아닌 값을 입력시 발생합니다.

**ERROR CODE: -1204, 'Json KEY "%1\$s" is duplicate'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 객체를 생성하거나 조작할 때, 동일한 이름의 키(key)가 두 번 이상 사용되었을 때 발생합니다. JSON 표준에 따르면 JSON 객체 내의 모든 키는 고유해야 합니다. 따라서, JSON\_OBJECT 함수나 JSON\_OBJECTAGG와 같은 함수를 사용하여 JSON 데이터를 구성할 때, 중복된 키를 포함하려고 시도하면 시스템이 이를 감지하고 이 오류를 반환하여 데이터의 유효성을 보장합니다.

**ERROR CODE: -1205, 'Invalid type %1\$s. Expecting JSON document compatible data type (STRING or JSON).'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON 함수들을 사용 시 json doc로 변경할 수 없는 데이터 타입이 주어진 경우 발생합니다.

**ERROR CODE: -1206, 'Json PATH "%1\$s" does not address a cell in an Json ARRAY.'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON\_ARRAY\_INSERT 함수 사용시 발생할 수 있는 오류입니다.

**ERROR CODE: -1207, 'Invalid JSON path. Index is larger than allowed by system parameter json\_max\_array\_idx.'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON\_ARRAY\_INSERT 함수 사용시 두번째 인자인 json path에 index가 숫자가 아니거나 json\_max\_array\_idx 설정값을 초과할때 발생합니다.

**ERROR CODE: -1208, 'Invalid one/all argument. The one/all argument may only take one of these two values: "one", "all".'**

- 이 메시지는 CUBRID 데이터베이스에서 JSON 데이터를 다룰 때, JSON\_CONTAINS\_PATH 함수를 사용할 때, 두 번째 인자로 전달되는 "one" 또는 "all" 값이 유효하지 않을 때 발생합니다.

**ERROR CODE: -1284, 'Cannot truncate the table due to the primary key referred to on %1\$s. Try again with CASCADE option.'**

- 이 메시지는 CUBRID 데이터베이스에서 FOREIGN KEY로 참조하는 PRIMARY KEY가 존재하는 테이블에 대해 cascade option 없이 `truncate table`를 수행하는 경우 발생합니다.



**ERROR CODE: -1285, 'Cannot cascade truncate because the ON DELETE action of the foreign key (%1\$s) is not set to CASCADE.'**

- 이 메시지는 CUBRID 데이터베이스에서 FOREIGN KEY로 참조되는 테이블에 `truncate table cascade` 구문수행 시 FOREIGN KEY의 ``delete action``이 ``cascade``로 설정되지 않아 발생합니다.

**ERROR CODE: -1339, 'Unsupported argument type of stored procedure: %1\$s'**

- 이 메시지는 CUBRID에서 JAVA Stored procedure에서 지원하지 않는 인자 data type을 사용 시 발생합니다.

**ERROR CODE: -1340, 'Unsupported return type of stored procedure: %1\$s'**

- 이 메시지는 CUBRID에서 JAVA Stored procedure에서 지원하지 않는 return data type을 사용 시 발생합니다.

**ERROR CODE: -1343, 'RENAME does not allow change of owner.'**

- 이 메시지는 CUBRID에서 RENAME 명령을 사용할 때, 단순히 객체의 이름만 변경해야 하는데, 소유자까지 함께 변경하려고 시도할 경우 발생합니다. CUBRID의 보안 및 무결성 정책상, RENAME 명령은 객체의 소유자를 변경하는 기능을 지원하지 않으며, 오직 이름만 변경할 수 있습니다. 소유자 변경은 별도의 권한 관리 또는 DDL 명령을 통해서만 가능합니다.

**ERROR CODE: -1344, 'DBA, members of DBA group, and owner can perform CREATE SERIAL.'**

- 이 메시지는 CUBRID에서 시리얼(Serial, 자동 증가 시퀀스) 객체를 생성하려고 할 때, 해당 명령을 실행하는 사용자가 DBA, DBA 그룹의 멤버, 또는 시리얼을 생성하려는 대상 객체의 소유자가 아닐 경우 발생합니다. 즉, 시리얼 생성 권한이 없는 사용자가 CREATE SERIAL을 시도할 때 발생하는 권한(Privilege) 관련 오류입니다.

**ERROR CODE: -1348, 'Synonym "%1\$s" already exists.'**

- 이 메시지는 CUBRID에서 CREATE SYNONYM 구문을 실행할 때, 지정한 시노님 이름이 이미 데이터베이스에 존재할 경우 발생합니다. 즉, 동일한 이름의 시노님이 이미 등록되어 있어 중복 생성이 불가능할 때 발생하는 제약 조건 위반 오류입니다.

**ERROR CODE: -1349, 'Synonym "%1\$s" does not exist.'**

- 이 메시지는 CUBRID에서 DROP SYNONYM, ALTER SYNONYM, 또는 시노님을 참조하는 쿼리 등에서, 지정한 시노님 이름이 데이터베이스에 존재하지 않을 때 발생합니다. 즉, 사용자가 입력한 시노님이 현재 데이터베이스에 등록되어 있지 않거나, 이미 삭제된 경우입니다.

**ERROR CODE: -1351, 'The maximum length of the function expression string must be %1\$d.'**

- 이 메시지는 CUBRID에서 함수 기반 인덱스(Function-based Index) 생성 시 함수 인자로 사용된 expression 문자열에 최대값을 초과했을 때 발생합니다. 즉, 너무 긴 함수 식을 정의하려고 할 때 발생하는 제약 조건 위반 오류입니다.

**ERROR CODE: -1360, 'In line %1\$d, column %2\$d\nStored procedure compile error: %3\$s'**

- 이 메시지는 CUBRID PL/CSQL 생성 시 컴파일 오류입니다.

**ERROR CODE: -1363, 'Dropping system generated stored procedure is not allowed.'**

- 이 메시지는 CUBRID에서 생성한 JAVA Stored procedure/function 사용자가 삭제하려고 시도할때 발생합니다.

**ERROR CODE: -1365, 'Only DBA and the owner can grant the %1\$s privilege.'**

- 이 메시지는 CUBRID에서 JAVA Stored procedure/function 권한(grant) 설정 시 owner 또는 dba (dba group) 아닌 경우 발생합니다.

**ERROR CODE: -1366, 'String representation of a default value of a Stored procedure/function parameter may not exceed 255 characters.'**

- 이 메시지는 CUBRID에서 JAVA Stored procedure/function 생성하면서 인자의 default값 설정 시 char type의 문자열 길이 255 이상인 경우 발생합니다.

## 데이터베이스 관리 오류

**ERROR CODE: -1, 'Missing message for error code %1\$d.'**

- 이 메시지는 CUBRID 시스템이 규정할 수 없는 오류인 경우에 사용하는 오류로 발생한 원인에 대한 오류 메시지를 함께 출력되며 특정하게 규정할 수 없는 경우에 다양하게 사용하는 오류입니다.

**ERROR CODE: -2, 'Internal system failure: no more specific information is available.'**

- 이 메시지는 CUBRID 시스템 내부에서 예측하지 못한 오류가 발생했거나, 특정 오류 상황에 대한 더 구체적인 오류 코드를 설정할 수 없을 때 발생하는 일반적인(Generic) 시스템 오류입니다, CUBRID는 다양한 내부 컴포넌트와 모듈로 구성되어 있으며, 각 컴포넌트는 특정 실패 상황에 대해 고유한 오류 코드를 반환하도록 설계되어 있습니다. 그러나 때로는 예외적인 상황이나 복잡한 내부 상태로 인해 특정 오류를 명확히 식별하기 어려울 때 이 오류가 사용됩니다, 이 메시지는 심각한 내부 문제의 징후일 수 있으며, 데이터베이스의 안정성과 일관성에 영향을 미칠 가능성이 매우 높습니다.

**ERROR CODE: -4, 'Has been interrupted.'**

- 이 메시지는 CUBRID 시스템이 실행 중인 작업이 외부 신호나 시스템 이벤트에 의해 중단되었음을 나타내는 오류입니다, CUBRID는 다양한 작업(쿼리 실행, 데이터 로딩, 백업, 복구 등)을 수행하는 동안 중단 신호를 받을 수 있으며, 이 경우 현재 작업을 안전하게 중단하고 이 오류를 반환합니다.

**ERROR CODE: -83, 'Length of path "%1\$s" plus length of prefix logname "%2\$s" is too long; the combined length must be less than %3\$d.'**

- 이 메시지는 CUBRID 시스템이 로그 파일의 경로와 로그 이름을 합친 전체 길이가 허용된 최대 길이를 초과했을 때 발생하는 오류입니다, CUBRID는 로그 파일의 전체 경로 길이에 대한 제한하며 이 메시지는 주로 데이터베이스 경로나 로그 파일 이름이 너무 길어서 전체 경로가 시스템 제한을 초과할 때 발생합니다.

**ERROR CODE: -84, 'Length of prefix logname "%1\$s" is too long; the length must be less than %2\$d.'**

- 이 메시지는 CUBRID 시스템이 로그 파일의 이름의 길이가 허용된 최대 길이를 초과했을 때 발생하는 오류입니다, CUBRID는 로그 파일의 이름에 대한 길이 제한하며 이 메시지는 주로 데이터베이스 이름이 너무 길거나, 로그 파일 경로가 시스템 제한을 초과할 때 발생합니다.

**ERROR CODE: -85, 'The prefix name "%1\$s" is not the same as "%2\$s" on the log disk. The log may have been renamed outside the database domain.'**

- 이 메시지는 CUBRID 시스템이 로그 디스크에서 발견한 로그 파일의 이름이 데이터베이스가 예상하는 이름과 일치하지 않을 때 발생하는 오류입니다, 이는 데이터베이스의 트랜잭션 로그 파일이 데이터베이스 시스템 외부에서 수동으로 이름이 변경되었거나, 로그 파일이 잘못된 데이터베이스에 연결되었을 때 발생할 수 있습니다.

**ERROR CODE: -86, 'Database is incompatible with current "%1\$s" release "%2\$s"'**

- 이 메시지는 현재 사용 중인 CUBRID 릴리스(엔진) 버전이 데이터베이스와 호환되지 않음을 나타냅니다, 데이터베이스가 다른 버전의 CUBRID로 생성되었거나, 현재 CUBRID 버전이 데이터베이스의 형식과 호환되지 않는 경우 발생합니다, 이는 주로 CUBRID 버전 업그레이드/다운그레이드 과정에서 발생하는 호환성 문제입니다.

**ERROR CODE: -87, 'There are recovery actions that must recovered using "%1\$s" release "%2\$s" instead of release "%3\$s". After the recovery, the database can be run on release "%4\$s".'**

- 이 메시지는 데이터베이스 복구 작업이 현재 사용 중인 CUBRID 릴리스와 호환되지 않는 로그 레코드를 포함하고 있음을 나타냅니다, CUBRID 버전 호환성 문제로 주로 데이터베이스 업그레이드/다운그레이드 과정이나, 다른 버전의 CUBRID로 생성된 로그 파일을 복구하려 할 때 발생합니다.

**ERROR CODE: -88, 'Internal error: Release string "%1\$s" cannot exceed %2\$d. Must change header log.'**

- 이 메시지는 CUBRID 시스템이 로그 헤더에 저장된 "릴리스 문자열"의 길이가 허용된 최대 길이를 초과했음을 감지했을 때 발생하는 내부 오류입니다, 즉, 로그 헤더 저장시 릴리스 문자열 길이가 허용된 길이를 초과 했을때 발생합니다.

**ERROR CODE: -89, 'Log "%1\$s" does not belong to the given database.'**

- 이 메시지는 CUBRID 시스템이 특정 로그 파일이 현재 작업 중인 데이터베이스에 속하지 않는다고 판단할 때 발생하는 오류입니다, 로그 파일과 데이터베이스 간의 일치성 검증 과정에서 발생하는 오류로, 이는 주로 로그 파일이 잘못된 데이터베이스와 연결되었거나, 로그 파일이 손상되었거나, 데이터베이스 설정이 잘못되었을 때 발생합니다.

**ERROR CODE: -99, 'Unable to create backup directory information file "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 백업 작업을 수행할 때, 백업 디렉터리 정보를 저장하는 백업 정보 파일(DBname\_bkvinf)을 생성하려고 할 때 실패할 경우 발생하는 오류입니다.

**ERROR CODE: -100, 'Unable to backup database "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 백업 작업을 수행하려고 할 때, 백업 과정에서 오류가 발생하여 백업이 실패했을 때 발생하는 오류입니다, 백업 실패는 파일 시스템 권한 문제, 디스크 공간 부족, 파일 접근 오류, 또는 시스템 리소스 부족 등 다양한 원인으로 발생할 수 있습니다.

**ERROR CODE: -115, 'Database "%1\$s" already exists.'**

- 이 메시지는 CUBRID 시스템이 새로운 데이터베이스를 생성하려고 할 때, 동일한 이름의 데이터베이스가 이미 존재할 때 발생하는 오류입니다, CUBRID 데이터베이스 이름에 대/소문자를 구분하며 이 메시지는 이미 존재하는 데이터베이스와 같은 이름으로 새 데이터베이스를 생성할 수 없음을 의미합니다.

**ERROR CODE: -116, 'Database "%1\$s" is unknown, or the file "databases.txt" cannot be accessed.'**

- 이 메시지는 CUBRID 시스템이 지정된 데이터베이스를 찾을 수 없거나, 데이터베이스 목록을 관리하는 databases.txt 파일에 접근할 수 없을 때 발생하는 오류입니다, 주로 데이터베이스 이름이 잘못되었거나, databases.txt 파일이 손상되었거나, 파일 시스템 권한 문제로 인해 접근이 불가능할 때 발생하는 오류입니다.

**ERROR CODE: -117, 'Absolute pathname for the database is too long. The combined length of the path "%1\$s" plus the name "%2\$s" is %3\$d; the combined length must be less than %4\$d.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스의 전체 경로명(절대 경로 + 데이터베이스 이름)을 처리하려고 할 때, 그 길이가 시스템에서 허용하는 최대 길이를 초과했을 때 발생하는 오류입니다, 파일 시스템이나 CUBRID 내부적인 제한으로 인해 너무 긴 경로명은 처리할 수 없음을 의미합니다.

**ERROR CODE: -119, 'Cannot find hostname ("%1\$s") in "%2\$s". Please check permissions of the file or if there is the hostname.'**

- 이 메시지는 CUBRID 시스템이 특정 호스트 이름을 찾으려고 시도했지만 실패했을 때 발생하는 오류입니다, 주로 네트워크 통신을 설정하거나 호스트 정보를 조회하는 과정에서 발생하며, 지정된 호스트 이름이 유효하지 않거나, 호스트 정보를 담고 있는 파일(예: /etc/hosts)에 접근할 수 없을 때 나타납니다.

**ERROR CODE: -120, 'The maximum number of volumes (%1\$d) has been exceeded.'**

- 이 메시지는 CUBRID 시스템이 새로운 볼륨을 추가하려고 할 때, 시스템에서 허용하는 최대 볼륨 개수를 초과했을 때 발생하는 오류입니다. CUBRID는 시스템 안정성과 성능을 위해 데이터베이스 당 최대 볼륨 개수를 제한하고 있습니다. 이 제한을 초과하면 볼륨 추가가 실패합니다.

**ERROR CODE: -121, 'Trying to remove "%1\$s" a permanent volume from the database.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스에서 임시볼륨(temp temp volume)을 제거하려고 시도할 때 발생하는 오류입니다. 임시볼륨은 데이터베이스에서 쿼리를 정리하는 공간으로, 데이터베이스 구동(운영) 중 삭제가 허용되지 않습니다. 이 오류는 영구 볼륨을 제거하려는 시도가 감지되었음을 사용자에게 안내하는 이벤트 메시지입니다.

**ERROR CODE: -122, 'Unable to access system message catalog.'**

- 이 메시지는 CUBRID 시스템이 메시지 카탈로그 파일들을 초기화하려고 할 때, 메시지 카탈로그 파일에 접근할 수 없을 때 발생하는 오류입니다. 메시지 카탈로그는 CUBRID의 모든 에러 메시지와 로케일별 메시지를 포함하는 파일들로, 시스템 초기화 시 필수적으로 로드되어야 합니다.

**ERROR CODE: -124, 'Volume "%1\$s" already exists.'**

- 이 메시지는 CUBRID 시스템이 새로운 볼륨을 생성하려고 할 때, 동일한 이름의 볼륨이 이미 존재할 때 발생하는 오류입니다. 볼륨 확장이나 새 볼륨 생성 시 중복된 볼륨 이름으로 인해 생성이 실패했음을 의미합니다.

**ERROR CODE: -126, 'Unknown purpose "%1\$s" given on line %2\$d.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨 추가 설정 파일을 파싱하는 과정에서 특정 라인에 지정된 '볼륨 용도(PURPOSE)' 값이 유효하지 않거나 인식할 수 없을 때 발생하는 오류입니다. 주로 `createdb` 명령어에서 `-more-volume-file` 옵션 사용 시 발생하며 이는 볼륨 확장 설정 파일에서 볼륨의 용도를 지정하는 PURPOSE 파라미터에 허용되지 않는 문자열이 입력되었음을 의미합니다.

**ERROR CODE: -127, 'Incorrect value %1\$d for number of pages given on line %2\$d.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨 추가 설정 파일을 파싱하는 과정에서 특정 라인에 지정된 '페이지 수' 값이 유효하지 않을 때 발생하는 오류입니다. 주로 `createdb` 명령어에서 `-more-volume-file` 옵션 사용 시 발생하며 이는 볼륨 확장 설정 파일에서 볼륨의 크기를 지정하는 페이지 수가 허용되지 않는 값으로 지정되었음을 의미합니다.

**ERROR CODE: -128, 'Number of pages was not given on line %1\$d.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨 추가 설정 파일을 파싱하는 과정에서 특정 라인에 필요한 '페이지 수' 정보가 누락되었거나 유효하지 않을 때 발생하는 오류입니다. 주로 `createdb` 명령어에서 `-more-volume-file` 옵션 사용 시 발생하며 이는 볼륨 확장 설정 파일에서 볼륨의 크기를 지정하는 페이지 수가 정의되지 않았거나 올바르지 않은 형식으로 지정되었음을 의미합니다.

**ERROR CODE: -129, 'Unknown token "%1\$s" was found on line %2\$d.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨 추가 설정 파일을 파싱하는 과정에서 예상치 못한, 정의되지 않은 토큰(키워드)을 발견했을 때 발생하는 오류입니다, 볼륨 확장 설정 파일에서 지원되지 않는 키워드나 잘못된 구문이 사용되었음을 의미합니다.

**ERROR CODE: -131, 'Database system error.'**

- 이 메시지는 CUBRID 데이터베이스 시스템 내부에서 예측하지 못한 오류가 발생했음을 나타냅니다, 일반적으로 데이터베이스 엔진의 핵심 구성 요소에서 문제가 발생했을 때 출력되며 주로 관련 또는 질의 수행 요청시 관련 질의 또는 결과 정보가 없는 경우 발생합니다.

**ERROR CODE: -137, 'Corrupted authorization objects.'**

- 이 메시지는 CUBRID 데이터베이스에서 권한 정보가 맞지 않거나 컬럼이 없는 경우 발생합니다.

**ERROR CODE: -138, 'Authorization class "%1\$s" not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 권한 문제로 테이블을 찾을 수 없는 경우 발생합니다.

**ERROR CODE: -139, 'Access error on authorization attribute "%2\$s" of class "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 권한 문제로 관련 컬럼을 접근할 수 없는 경우 발생합니다.

**ERROR CODE: -140, 'Operation "%1\$s" can only be performed by the DBA or a DBA group member.'**

- 이 메시지는 CUBRID 시스템에서 특정 작업을 수행하려고 할 때, 해당 작업이 DBA 또는 DBA 그룹의 멤버만 수행할 수 있는 권한이 필요한 작업일 경우 발생하는 오류입니다, 일반 사용자가 시스템 관리자 권한이 필요한 작업을 시도했을 때 발생하며, 보안상의 이유로 접근이 거부되었음을 의미합니다.

**ERROR CODE: -141, 'Cannot add "%1\$s" as a member of "%2\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 user 생성시 public의 member로 등록하는 과정에서 발생하는 오류로 경고성 메시지입니다.

**ERROR CODE: -142, 'Adding member causes the user hierarchy to become a cyclic graph.'**

- 이 메시지는 CUBRID 시스템에서 사용자 또는 그룹에 멤버를 추가하는 작업이 사용자 계층 구조에 순환 참조(cyclic dependency)를 발생시킬 때 발생하는 오류입니다, 예를 들어, 그룹 A에 그룹 B를 멤버로 추가하고, 동시에 그룹 B에 그룹 A를 멤버로 추가하려고 할 때 발생할 수 있습니다. 이는 무한 루프나 논리적 모순을 야기할 수 있으므로 시스템에서 허용되지 않습니다.

**ERROR CODE: -143, 'Encountered a class with no owner.'**

- 이 메시지는 CUBRID 데이터베이스에서 권한을 수정하려는 객체의 owner가 없는 경우 발생하는 오류입니다.

**ERROR CODE: -146, 'Cannot issue GRANT/REVOKE to owner of a %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 owner의 object를 grantt 수행 시도시 발생하는 오류입니다.

**ERROR CODE: -147, 'No GRANT option.'**

- 이 메시지는 CUBRID 데이터베이스에서 grant 수행시 grant option을 주지 않아서 발생하는 오류입니다.

**ERROR CODE: -148, 'Cannot obtain write lock on authorization object.'**

- 이 메시지는 CUBRID 데이터베이스에서 grant 또는 revoke 처리시 write lock을 얻지 못해서 update를 할 수 없는 경우 발생하는 오류입니다.

**ERROR CODE: -150, 'Cannot revoke privileges from self.'**

- 이 메시지는 CUBRID 데이터베이스에서 자신의 object를 revoke 수행 시도시 발생하는 오류입니다.

**ERROR CODE: -151, 'Cannot revoke privileges from owner of a %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 owner의 object를 revoke 수행 시도시 발생하는 오류입니다.

**ERROR CODE: -152, 'GRANT not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 revoke 실행시 관련 grant 정보를 찾지 못할 경우 발생하는 오류입니다.

**ERROR CODE: -153, 'No authorization privileges in effect for the database.'**

- 이 메시지는 CUBRID 데이터베이스에서 해당 DB에 권한관련 catalog class가 존재하지 않을 경우 발생하는 오류입니다.

**ERROR CODE: -154, 'Incomplete authorization installation no authorization privileges in effect for the database.'**

- 이 메시지는 CUBRID 데이터베이스에서 해당 DB에 dba, public user가 존재하지 않은 경우 발생하는 오류입니다.

**ERROR CODE: -156, 'Authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 알수 없는 권한이 요청되었을때 발생하는 오류입니다.

**ERROR CODE: -157, 'SELECT authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 select 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.



**ERROR CODE: -158, 'ALTER authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 alter 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -159, 'UPDATE authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 update 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -160, 'INSERT authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 insert 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -161, 'DELETE authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 delete 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -162, 'INDEX authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 index 처리 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -163, 'EXECUTE authorization failure.'**

- 이 메시지는 CUBRID 데이터베이스에서 execute 권한이 해당 테이블에 없는 경우 발생하는 오류입니다.

**ERROR CODE: -164, 'User "%1\$s" already exists.'**

- 이 메시지는 CUBRID 시스템에서 새로운 사용자를 생성하려고 할 때, 동일한 이름의 사용자가 이미 시스템에 존재하는 경우 발생하는 오류입니다, CREATE USER 문을 실행할 때 중복된 사용자 이름으로 인해 사용자 생성이 실패했음을 의미합니다.

**ERROR CODE: -165, 'User "%1\$s" is invalid.'**

- 이 메시지는 CUBRID 시스템에서 특정 작업을 수행하려고 할 때, 지정된 사용자 이름이 유효하지 않거나 시스템에 존재하지 않는 사용자일 경우 발생하는 인증 관련 오류입니다, 데이터베이스에 접속하거나, 사용자 권한을 변경하거나, 특정 객체의 소유자를 지정하는 등의 작업에서 잘못된 사용자 이름이 사용되었음을 의미합니다.

**ERROR CODE: -166, 'Invalid user specified.'**

- 이 메시지는 CUBRID 시스템에서 사용자 권한(GRANT) 작업을 수행할 때, 지정된 사용자가 유효하지 않거나 존재하지 않을 때 발생하는 오류입니다, 주로 사용자 이름 오류나 존재하지 않는 사용자를 참조할 때 발생합니다.



**ERROR CODE: -167, 'DBA, members of DBA group and %1\$s owner can perform the operation.'**

- 이 메시지는 CUBRID 시스템에서 특정 작업을 수행하려고 할 때, 해당 작업을 수행할 권한이 없는 사용자에게 발생하는 인증 관련 오류입니다, DBA나 DBA 그룹의 멤버, 또는 특정 객체의 소유자만이 해당 작업을 수행할 수 있음을 알려줍니다.

**ERROR CODE: -168, '168 Member not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 drop member 시 member가 아닌 user를 삭제 요청할 때 발생하는 오류입니다.

**ERROR CODE: -169, '169 Cannot remove user %1\$s from the database.'**

- 이 메시지는 CUBRID 데이터베이스에서 drop user시 현재 login된 사용자를 삭제 요청할 때 발생하는 오류입니다.

**ERROR CODE: -171, 'Incorrect or missing password.'**

- 이 메시지는 CUBRID 시스템에서 사용자 인증을 시도할 때 제공된 비밀번호가 올바르지 않거나, 비밀번호가 아예 제공되지 않았을 때 발생하는 오류입니다, 데이터베이스에 접속하거나 큐브리드 명령어 작업을 수행하기 위해 필요한 인증 과정에서 비밀번호 불일치 또는 누락이 감지되었음을 의미합니다.

**ERROR CODE: -172, '172 Password string cannot have more than 31 bytes.'**

- 이 메시지는 CUBRID 데이터베이스 db user의 암호 길이가 31자를 넘을 경우 발생하는 오류입니다.

**ERROR CODE: -173, 'Could not locate database file "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 databases.txt 파일을 찾을 수 없을 때 발생하는 오류입니다.

**ERROR CODE: -174, 'Could not obtain write access to database file "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 특정 databases.txt 파일에 대한 쓰기 권한을 얻지 못했을 때 발생하는 오류입니다.

**ERROR CODE: -183, 'Unexpected amount of received data; %1\$d expected, %2\$d received.'**

- 이 메시지는 CUBRID 클라이언트가 서버로부터 데이터를 수신할 때, 예상했던 데이터의 크기와 실제로 수신된 데이터의 크기가 일치하지 않을 때 발생하는 오류입니다, 네트워크 통신 중 데이터 무결성 문제, 프로토콜 불일치, 또는 데이터 전송 오류로 인해 발생할 수 있습니다, 클라이언트가 서버로부터 받은 데이터의 양이 예상과 다른 잘못된 통신을 의미할 수 있습니다.

**ERROR CODE: -184, 'Cannot allocate communications buffer.'**

- 이 메시지는 CUBRID 시스템이 네트워크 통신을 위한 버퍼를 할당할 수 없을 때 발생하는 오류입니다, 메모리 부족, 시스템 리소스 한계, 또는 버퍼 할당 실패로 인해 발생합니다, 클라이언트와 서버 간의 데이터 전송을 위한 통신 버퍼를 생성하지 못했을 때 발생합니다.

**ERROR CODE: -185, 'Error receiving data from client.'**

- 이 메시지는 CUBRID 서버가 클라이언트로부터 데이터를 수신하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 네트워크 통신 중 클라이언트 데이터 수신 실패, 프로토콜 불일치, 클라이언트 응답 오류 등으로 인해 발생합니다, 서버가 클라이언트로부터 예상한 데이터를 받지 못했거나, 받은 데이터가 올바르지 않을 때 발생합니다.

**ERROR CODE: -186, 'Error receiving data from server.'**

- 이 메시지는 CUBRID 클라이언트가 서버로부터 데이터를 수신하는 과정에서 오류가 발생했을 때 나타나는 오류입니다, 네트워크 통신 중 데이터 수신 실패, 프로토콜 불일치, 서버 응답 오류 등으로 인해 발생합니다, 클라이언트가 서버로부터 예상한 데이터를 받지 못했거나, 받은 데이터가 올바르지 않을 때 발생합니다.

**ERROR CODE: -187, 'Communications buffer not used.'**

- 이 메시지는 CUBRID 클라이언트가 서버로부터 데이터를 수신할 때, 예상했던 버퍼와 실제로 받은 버퍼가 다를 때 발생하는 오류입니다, 네트워크 통신 과정에서 버퍼 관리에 문제가 있거나 예상과 다른 데이터가 수신되었을 때 발생합니다.

**ERROR CODE: -188, 'Unknown database "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 지정된 데이터베이스 이름이 NULL 또는 최대 길이를 초과한 경우 발생합니다.

**ERROR CODE: -189, 'Invalid host name "%1\$s"'**

- 이 메시지는 CUBRID 클라이언트가 서버에 연결할 때 제공된 호스트 이름이 유효하지 않거나 잘못된 형식일 때 발생하는 오류입니다, 호스트 이름이 NULL이거나, 최대 허용 길이를 초과하거나, 올바른 형식이 아닐 때 발생합니다, 클라이언트 초기화 과정에서 서버 호스트 정보의 유효성을 검증할 때 발생합니다.

**ERROR CODE: -190, 'Server host not identified.'**

- 이 메시지는 CUBRID 클라이언트가 서버 호스트 정보를 찾을 수 없을 때 발생하는 오류입니다, 클라이언트 초기화 과정에서 서버 호스트명이나 서버 이름이 설정되지 않았거나 비어있는 상태에서 발생합니다, 클라이언트가 어떤 서버에 연결해야 하는지 알 수 없는 상태를 나타냅니다.

**ERROR CODE: -191, 'Cannot connect to server "%1\$s" on "%2\$s"'**

- 이 메시지는 CUBRID 클라이언트가 지정된 호스트의 데이터베이스 서버 프로세스(cub\_server)에 연결할 수 없을 때 발생하는 오류입니다, 네트워크 연결 실패, 서버가 실행되지 않음, 방화벽 차단, 포트 문제 등으로 인해 발생할 수 있습니다, 클라이언트가 서버와의 통신을 설정할 수 없는 상태를 나타냅니다.

**ERROR CODE: -193, 'Server received shutdown command from client.'**

- 이 메시지는 CUBRID 데이터베이스가 클라이언트로부터 종료(shutdown) 명령을 받았을 때 발생하는 오류입니다. 데이터베이스 서버가 클라이언트의 요청에 따라 정상적으로 종료 프로세스를 시작했음을 알립니다, 이는 오류가 아니라 서버의 정상적인 종료를 알리는 이벤트 메시지입니다.

**ERROR CODE: -194, 'Unknown server request id %1\$d.'**

- 이 메시지는 CUBRID 서버가 클라이언트로부터 받은 요청 ID (서비스 ID) 가 유효하지 않거나 정의되지 않은 요청일 때 발생하는 오류입니다, 서버가 처리할 수 없는 요청 ID를 받았을 때 발생하며, 이는 주로 클라이언트와 서버 간의 프로토콜 버전 불일치나 잘못된 요청 형식으로 인해 발생합니다.

**ERROR CODE: -195, 'Server communications error: %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 내/외부 클라이언트와 서버 간의 통신 과정에서 발생하는 오류를 나타냅니다, 서버와의 통신이 실패하거나 연결이 끊어졌을 때, 또는 서버 통신 프로토콜에 문제가 있을 때 발생합니다, 주로 네트워크 연결 문제, 서버 상태 문제, 또는 통신 프로토콜 오류와 관련이 있습니다.

**ERROR CODE: -196, 'Server name not identified.'**

- 이 메시지는 CUBRID 데이터베이스 서버에 연결을 시도할 때, 서버 이름이 설정되지 않았거나 유효하지 않을 때 발생하는 오류입니다, 주로 클라이언트가 연결하려는 서버의 이름이나 호스트 정보가 올바르게 설정되지 않은 상태를 나타냅니다.

**ERROR CODE: -197, 'Could not make contact with master server.'**

- 이 메시지는 CUBRID 데이터베이스가 마스터(cub\_master) 프로세스로 연결을 시도했지만 연결에 실패했음을 나타냅니다, 마스터(cub\_master)는 CUBRID 시스템에서 여러 데이터베이스를 관리 역할을 하는 핵심 프로세스로 데이터베이스 작업을 수행하기 위해서는 마스터 서버와의 연결이 필요합니다, 주로 마스터 프로세스가 비정상적인 동작 상황이나 환경(방화벽등)에 있을 때 발생합니다.

**ERROR CODE: -199, 'Server no longer responding.'**

- 이 메시지는 CUBRID 데이터베이스 서버가 클라이언트의 요청에 응답하지 않거나, 서버 프로세스가 예기치 않게 종료되었을 때 발생하는 오류입니다, 서버가 더 이상 클라이언트와 통신할 수 없는 상태를 나타내며, 이는 DB와 접속 문제, 연결 끊김, 또는 서버 프로세스 중단 등의 원인으로 발생할 수 있습니다.

**ERROR CODE: -204, 'Function called with missing or invalid arguments.'**

- 이 메시지는 CUBRID 데이터베이스 내부에서 함수를 호출할 때, 필수 인자가 누락되었거나 전달된 인자의 값이 유효하지 않을 때 발생하는 오류입니다, 일반적으로 사용자 직접적인 SQL 오류라기 보다는 CUBRID 엔진 내부의 문제나 개발자가 CUBRID API를 잘못 사용했을 때 발생할 가능성이 높습니다.

**ERROR CODE: -210, 'Internal error: processing object template.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체 템플릿 처리 중 내부 오류가 발생했을 때 나타나는 오류입니다, 주로 테이블 캐시 불일치, 락 업그레이드 실패, 또는 템플릿 상태 검증 오류로 인해 발생할 수 있습니다.

**ERROR CODE: -217, 'Operation can only be performed on class objects.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 작업이 테이블 객체에 대해서만 유효할 때, 해당 작업이 테이블 객체가 아닌 다른 유형의 객체에 대해 시도되었음을 나타냅니다, 주로 스키마 관리, 메타데이터 조회, 또는 객체 템플릿 처리와 같은 테이블 정의와 관련된 작업에서 발생할 수 있습니다.

**ERROR CODE: -224, 'A database has not been restarted.'**

- 이 메시지는 CUBRID 데이터베이스에서 데이터베이스 연결이 설정되지 않았거나, 데이터베이스가 재시작되지 않은 상태에서 데이터베이스 작업을 시도할 때 발생하는 오류입니다, 주로 데이터베이스 서버와의 연결이 끊어졌거나, 데이터베이스가 정상적으로 시작되지 않은 상태에서 SQL 문을 실행하거나 데이터베이스 작업을 수행하려 할 때 발생합니다.

**ERROR CODE: -231, 'Invalid API operation attempted on a temporary object.'**

- 이 메시지는 CUBRID 데이터베이스에서 임시 객체(temporary object)에 대해 허용되지 않는 API 작업을 시도할 때 발생하는 오류입니다, CUBRID에서 임시 객체는 주로 스키마 템플릿이나 트리거와 관련된 임시적인 객체로, 일반적인 데이터베이스 객체와는 다른 생명주기와 제약사항을 가집니다, 주로 임시 객체에 대해 일반 객체에서만 허용되는 작업(예: 데이터 접근, 수정, 삭제 등)을 시도할 때 발생합니다.

**ERROR CODE: -232, 'Not allowed to use object templates over multiple transactions.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체 템플릿(object template)이 여러 트랜잭션에 걸쳐 사용하려고 할 때 발생하는 오류입니다, 객체 템플릿은 일반적으로 단일 트랜잭션 내에서 생성 및 사용되어야 하는 임시적인 스키마 정의 객체입니다. 여러 트랜잭션에 걸쳐 사용될 경우, 데이터 일관성 문제나 예상치 못한 동작을 유발할 수 있어 시스템에서 이를 금지합니다, 주로 객체 템플릿의 생명주기 관리와 관련된 로직에서 발생하며, 템플릿이 생성된 트랜잭션 범위를 벗어나 사용될 때 나타납니다.

**ERROR CODE: -247, 'Invalid arguments to schema manager internal function.'**

- 이 메시지는 CUBRID 데이터베이스의 스키마 관리자 내부 함수에 잘못된 인자(NULL, 빈 문자열, 또는 유효하지 않은 값)가 전달되었을 때 발생하는 내부 오류입니다, 주로 스키마 생성, 수정, 조회 등의 작업 중에 발생하며, 개발자나 시스템 레벨에서의 문제를 나타냅니다.

**ERROR CODE: -263, 'Schema manager internal corruption detected.'**

- 이 메시지는 CUBRID 데이터베이스의 스키마 관리자(Schema Manager) 내부에서 데이터 손상이 감지되었음을 나타내는 시스템 오류입니다, 주로 객체 표현(object representation), 테이블 정의,

또는 메타데이터 처리 과정에서 내부 일관성 검사가 실패했을 때 발생합니다, 이는 데이터베이스 파일 손상, 메모리 오류, 또는 시스템 비정상 종료 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -314, 'Object buffer overflow while writing.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체(테이블 레코드 등)의 데이터를 저장(쓰기)할 때, 내부적으로 할당된 버퍼의 크기를 초과하는 데이터가 기록되려고 할 때 발생합니다, 주로 대용량 데이터 처리, 비정상적으로 큰 객체, 버퍼 크기 설정 오류, 내부 버그 등으로 인해 나타날 수 있으며, 데이터 손실, 트랜잭션 실패, 시스템 오류로 이어질 수 있습니다.

**ERROR CODE: -321, 'Possible corruption in the workspace detected.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace에서 처리시 원하지 않는 정보가 존재할 경우 발생하는 오류입니다, workspace 관리 시스템에서 데이터 무결성 검증 과정에서 발생합니다, 일반적으로 workspace는 메모리 내의 객체 관리 구조가 일관성을 유지해야 하는데, 이러한 일관성이 깨졌을 때 이 오류가 보고됩니다.

**ERROR CODE: -322, 'MOP not found in the workspace.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace 내의 해시 테이블에서 특정 MOP를 찾을 수 없을 때 나타나는 오류입니다, CUBRID 내부의 MOP 관리 로직에 문제가 있거나, 동시성 제어 메커니즘에 예기치 않은 문제가 발생했을 때 나타날 수 있습니다

**ERROR CODE: -324, 'Object found in workspace without class.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace 내의 객체(MOP)가 테이블 정보 없이 존재할 때 나타나는 오류입니다, CUBRID 내부의 객체-테이블 관리 로직에 문제가 있거나, 동시성 제어 메커니즘에 예기치 않은 문제가 발생했을 때 나타날 수 있습니다.

**ERROR CODE: -325, 'Attempt to garbage collect a MOP in the dirty list.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace의 가비지 컬렉션(Garbage Collection) 과정 중, 'dirty list'에 있는 MOP를 정리하려고 시도했을 때 나타나는 오류입니다, workspace 관리 시스템에서 MOP의 상태 일관성 검증 과정에서 발생합니다, 일반적으로 MOP는 'dirty list'에 있으면 가비지 컬렉션 대상이 아니어야 하는데, 이러한 불일치가 발생했을 때 이 오류가 보고됩니다.

**ERROR CODE: -326, 'Changing class pointer for object in workspace.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace에 instance 캐싱시 class MOP되어 발생하는 오류입니다, workspace 관리 시스템에서 객체의 테이블 할당 과정에서 발생합니다, 주로 이미 다른 테이블에 속한 객체를 다른 테이블로 재할당하려고 할 때 발생합니다.

**ERROR CODE: -327, 'Cannot create MOP with NULL OID.'**

- 이 메시지는 CUBRID 데이터베이스에서 Managed Object Pointer (MOP)를 생성할 때 Object ID (OID)가 NULL인 경우 나타나는 오류입니다, workspace 관리 시스템에서 MOP 생성 시 OID 유효성 검증 과정에서 발생합니다, 주로 OID 할당 실패하거나 OID가 포함된 메모리 구조가 손상되었을 때 발생할 수 있습니다.

**ERROR CODE: -328, 'Instance encountered in workspace without class.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace에 테이블 정의가 없는 레코드가 발견되었을 때 나타나는 오류입니다, 스키마 관리 시스템에서 레코드와 테이블 간의 관계 검증 과정에서 발생합니다, 주로 테이블을 가져오지 못하거나 테이블을 찾을 수 없을 때 발생합니다.

**ERROR CODE: -335, 'Ignoring attempt to free object not allocated within the workspace.'**

- 이 메시지는 CUBRID 데이터베이스에서 workspace(작업 공간) 내에 할당되지 않은 메모리 오브젝트를 해제하려고 시도했을 때 나타나는 오류입니다, 메모리 관리 시스템에서 포인터 유효성 검증 과정에서 발생하는데 주로 메모리 포인터가 유효한 area 블록 내에 있지 않을 때 발생합니다.

**ERROR CODE: -344, '%1\$s.'**

- 이 메시지는 CUBRID 데이터베이스 프로세스(cub\_server)가 shutdown 진행 중이라는 경고성 메시지는입니다.

**ERROR CODE: -345, 'Error occurred during reading a shutdown message from master.'**

- 이 메시지는 CUBRID 데이터베이스 프로세스(cub\_server)가 마스터 프로세스(cub\_master)로부터 shutdown 요청 메시지를 확인시 발생합니다.

**ERROR CODE: -347, 'Could not establish a pipe to the master server.'**

- 이 메시지는 CUBRID 데이터베이스 서버 프로세스(cub\_server)에서 마스터 프로세스(cub\_master)와 통신 수행하려는데, invalid socket인 경우 발생하는 오류입니다.

**ERROR CODE: -350, 'Cannot find hostname ("%1\$s") in "%2\$s". Please check permissions of the file or if there is the hostname.'**

- 이 메시지는 CUBRID 시스템에서 hostname으로 ip address 가져오는 발생하는 오류입니다, /etc/hosts의 권한 오류 또는 hostname 정보가 없는 경우, use\_user\_host = yes인 경우, \$CUBRID/conf/cubrid\_hosts.conf 에 hostname 정보가 없는 경우에서 발생합니다.

**ERROR CODE: -363, 'Recvmsg from master socket failed.'**

- 이 메시지는 CUBRID 데이터베이스 프로세스(cub\_server)와 마스터 프로세스(cub\_master) 연결 후 메시지가 수신할 경우 발생하는 오류입니다.

**ERROR CODE: -364, 'Sending a new request to server.'**

- 이 메시지는 CUBRID 데이터베이스 프로세스(cub\_server)와 마스터 프로세스(cub\_master) 연결이 종료된 경우 발생합니다.

**ERROR CODE: -366, '366 Error from server: %1\$s'**

- 이 메시지는 CUBRID HA 환경에서 change mode시 발생하거나 CAS 프로세스가 standby server에 연결되었을 경우 데이터베이스 서버 프로세스(cub\_server)가 연결 종료시킬때 발생하는 오류입니다.

**ERROR CODE: -367, 'Server %1\$s already exists.'**

- 이 메시지는 CUBRID 시스템에서 데이터베이스 생성 시 동일한 데이터베이스가 이미 존재하는 경우 발생합니다.

**ERROR CODE: -368, 'Communication error during connect for %1\$s.'**

- 이 메시지는 CUBRID 시스템에서 데이터베이스 프로세스(cub\_server)구동 시 마스터 프로세스(cub\_master)와 연결 중 발생합니다.

**ERROR CODE: -423, 'Invalid session.'**

- 이 메시지는 CUBRID 데이터베이스에서 유효하지 않은 세션이 사용되었을 때 나타나는 오류입니다, 세션 검증 과정에서 세션이 NULL이거나 파서(parser)가 초기화되지 않은 상태일 때입니다.

**ERROR CODE: -430, 'Unknown variable "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 정의되지 않은 변수가 참조되었을 때 나타나는 오류입니다, 변수 검증 과정에서 호스트 변수(host variable) 처리 과정에서 변수를 찾을 수 없을 때입니다.

**ERROR CODE: -456, 'Data type references are incompatible.'**

- 이 메시지는 CUBRID 데이터베이스에서 서로 다른 데이터 타입 간의 참조나 연산이 호환되지 않을 때 나타나는 오류입니다, 쿼리에서 데이터 타입 호환성 검증 과정에서 발생합니다, 즉, 서로 다른 데이터 타입 간의 비교, 연산, 할당 등이 불가능한 경우 발생합니다.

**ERROR CODE: -469, 'Invalid data type used.'**

- 이 메시지는 CUBRID 데이터베이스에서 지원하지 않는 또는 잘못된 데이터 타입이 사용되었을 때 나타나는 오류입니다.

**ERROR CODE: -547, 'Server release %1\$s is different from client release %2\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 서버와 클라이언트의 릴리즈 버전이 다를 때 나타나는 오류입니다, 주로 네트워크 연결 과정에서 서버와 클라이언트 간의 버전 호환성 검사에서 발생합니다, 서버와 클라이언트의 릴리즈 버전이 호환되지 않을 때 연결이 거부됩니다.

**ERROR CODE: -554, 'The loader has entered an invalid state. No further operations are possible.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 잘못된 상태에 진입했을 때 발생하는 오류입니다, 이는 주로 loaddb 명령어 세션이 유효하지 않거나, 로더가 예상치 못한 상태에 있을 때 발생합니다, 로더(loaddb)가 이 상태에 있으면 더 이상의 데이터 로드 작업을 수행할 수 없습니다.

**ERROR CODE: -555, 'Memory allocation error during database loading.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 필요한 메모리를 할당하지 못했을 때 발생합니다, 로딩 작업 중 대량의 데이터를 처리하거나, 시스템의 가용 메모리가 부족한 경우, 또는 메모리 파편화, 운영체제의 제한 등으로 인해 할당 요청이 실패할 수 있습니다.



**ERROR CODE: -556, 'Too many values supplied. Expecting %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 제공된 값의 개수가 예상된 개수보다 많을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 속성 값을 처리할 때 발생합니다, 테이블에 정의된 속성의 개수보다 많은 값이 제공되었을 때 발생합니다.

**ERROR CODE: -559, 'For attribute %1\$s of %2\$s, expected type %3\$s, got %4\$s.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 속성의 도메인 타입이 예상된 타입과 일치하지 않을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 도메인 타입을 검증할 때 발생합니다, 특정 속성에 대해 예상된 타입과 실제로 받은 타입이 다를 때 발생합니다.

**ERROR CODE: -560, 'For attribute %1\$s of %2\$s, domain is not specific enough to support unqualified object references.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 객체 참조의 도메인이 모호할 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 객체 도메인을 검증할 때 발생합니다, 특정 속성의 도메인이 unqualified 객체 참조를 지원하기에 충분히 구체적이지 않을 때 발생합니다.

**ERROR CODE: -561, 'Nested sets are not allowed.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 중첩된 집합(collection)을 처리하려고 할 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 컬렉션 요소를 처리할 때 발생합니다, CUBRID는 중첩된 집합을 허용하지 않으므로, 집합 내에 또 다른 집합이 포함되려고 하면 이 오류가 발생합니다.

**ERROR CODE: -562, 'System class %1\$s cannot be populated using the database loader.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 시스템 테이블을 로드하려고 할 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 컨텍스트를 초기화할 때 발생합니다, 시스템 테이블은 CUBRID 내부에서 사용되는 특별한 테이블로, 일반적인 데이터 loaddb를 통해 로드할 수 없습니다.

**ERROR CODE: -563, 'Reference to internal class %1\$s converted to NULL.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 내부 테이블에 대한 참조를 처리할 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 내부 테이블에 대한 참조가 발견되면, 이를 NULL로 변환하여 처리합니다.

**ERROR CODE: -565, 'Constructor method %1\$s is not defined on class %2\$s.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 지정된 생성자 메소드가 테이블에 정의되어 있지 않을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 생성자 메소드가 테이블에 존재하지 않거나 정의되지 않았을 때 발생합니다.



**ERROR CODE: -566, 'No class specified or no attribute in class.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드할 때, 테이블이 지정되지 않았거나 테이블에 속성이 없을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 테이블이 존재하지 않거나, 테이블은 존재하지만 속성이 정의되지 않았을 때 발생합니다.

**ERROR CODE: -567, 'Too many constructor method arguments supplied. Expected %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 예상했던 인자의 개수보다 많은 수의 인자가 제공되었을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 생성자 메소드에 필요한 인자의 개수보다 많은 수의 인자가 제공되었을 때 발생합니다.

**ERROR CODE: -568, 'Missing constructor method parameters. Expected %1\$d, found %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 예상했던 인자의 개수와 실제로 제공된 인자의 개수가 일치하지 않을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 생성자 메소드에 필요한 인자의 개수보다 적은 수의 인자가 제공되었을 때 발생합니다.

**ERROR CODE: -569, 'Missing attribute values. Expected %1\$d, found %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 데이터를 로드하는 과정에서 예상했던 속성 값의 개수와 실제로 발견된 속성 값의 개수가 일치하지 않을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다, 테이블에 정의된 속성의 개수보다 적은 수의 속성 값이 제공되었을 때 발생합니다.

**ERROR CODE: -570, 'Could not access Glo data file "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 GLO(Generalized Large Object) 데이터 파일에 액세스할 수 없을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 BLOB(Binary Large Object) 또는 CLOB(Character Large Object) 데이터를 처리할 때 발생합니다, LOB 데이터 파일의 경로가 잘못되었거나, 파일이 존재하지 않거나, 권한이 없을 때 발생합니다.

**ERROR CODE: -571, 'Instance was previously created through a forward reference.nThis is not allowed for instances created using constructor methods.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 forward reference를 통해 이미 생성된 레코드를 constructor 메소드를 사용하여 다시 생성하려고 할 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블 레코드를 생성할 때 발생합니다.

**ERROR CODE: -576, 'Reference to class %3\$s is not compatible with the domain of %1\$s of %2\$s.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 테이블 객체를 참조할 때, 해당 참조가 특정 속성의 도메인과 호환되지 않을 때 발생하는 오류입니다, 이는 주로 데이터 로드 과정에서 테이블

블 레코드를 생성하거나 업데이트할 때 발생합니다, 테이블 참조가 속성의 도메인과 일치하지 않을 때 발생합니다.

**ERROR CODE: -579, 'Target domain must include type "object" to allow references to classes.'**

- 이 메시지는 CUBRID 데이터베이스 loaddb 명령어에서 테이블 객체를 참조할 때 도메인 타입이 올바르게 없을 때 발생하는 오류입니다, 테이블 객체를 참조하려면 대상 도메인이 "object" 타입을 포함해야 합니다, 이는 CUBRID의 타입 시스템에서 테이블 참조를 위한 특별한 요구사항입니다.

**ERROR CODE: -581, 'Attempted to update the database when updates are disabled.'**

- 이 메시지는 CUBRID 데이터베이스 서버가 standby 또는 to\_be\_active 상태로 데이터 수정이 비활성화된 상태에서 데이터베이스를 수정하려고 할 때 발생하는 오류입니다, 데이터베이스 수정이 비활성화된 상태는 읽기 전용 모드나 특별한 상황에서 발생할 수 있습니다.

**ERROR CODE: -582, 'Unknown storage purpose %1\$d. Valid range is from %2\$d to %3\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 볼륨의 사용 용도(purpose)가 유효하지 않을 때 발생하는 디스크 관리 오류입니다, 볼륨 사용 용도는 데이터가 영구적인지(permanent) 또는 임시적인지(temporary)를 나타내는 값입니다, 유효한 범위를 벗어난 값이 전달되어 볼륨을 생성하거나 관리할 수 없습니다.

**ERROR CODE: -588, 'INTERNAL ERROR: Assertion '%1\$s' failed.'**

- 이 메시지는 CUBRID 데이터베이스에서 내부적인 오류로 특정 조건이 실패했을 때 발생하는 내부 오류입니다, 이 오류가 발생하면 CUBRID의 내부 로직에 문제가 있음을 의미합니다.

**ERROR CODE: -589, 'Invalid user name "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 사용자 이름이 유효하지 않거나 존재하지 않음을 나타냅니다, 사용자 인증, 권한 관리, 저장 프로시저 생성 등에서 사용자 이름을 검증할 때 발생합니다, 이는 존재하지 않는 사용자를 참조할 때 발생할 수 있습니다.

**ERROR CODE: -595, 'The database name "%1\$s" is too long. Database name should be less than %2\$d characters.'**

- 이 메시지는 CUBRID에서 데이터베이스 이름이 최대 길이를 초과해서 데이터베이스 이름과 동일한 볼륨 파일명으로 생성할 때 발생하는 오류입니다.

**ERROR CODE: -611, '%1\$s cannot be executed in client/server mode.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 기능이 클라이언트/서버 모드에서 실행될 수 없을 때 나타납니다, 해당 기능은 standalone 모드에서만 실행 가능하다는 의미입니다.

**ERROR CODE: -612, 'Logging cannot be disabled at this moment since there are pending actions to recover.'**

- 이 메시지는 CUBRID 데이터베이스에서 loaddb -S 에서 -no-logging 사용 시 로깅을 비활성화하려고 할 때 발생합니다, 복구해야 할 pending action이 있는 상태에서는 로깅을 비활성화할 수 없다는 의미입니다.

**ERROR CODE: -619, 'Bad source codeset.'**

- 이 메시지는 CUBRID 데이터베이스에서 문자열의 codeset 변환 시 소스(source) 코드 세트가 유효하지 않거나 지원되지 않을 때 나타납니다, 문자열을 한 코드 세트에서 다른 코드 세트로 변환할 때 소스 코드 세트가 잘못 지정되었을 때 발생할 수 있습니다.

**ERROR CODE: -629, 'Pagesize %1\$d is not a power of 2 or pagesize is too small. Pagesize of %2\$d is used instead.'**

- 이 메시지는 CUBRID 데이터베이스에서 페이지 크기를 설정할 때 발생하는 경고입니다, 사용자가 지정한 페이지 크기가 2의 제곱수가 아니거나 최소 크기보다 작을 때 나타납니다, CUBRID 시스템이 자동으로 유효한 페이지 크기로 조정했음을 알립니다.

**ERROR CODE: -630, 'Conversion not supported by %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 데이터 타입 간의 변환을 시도했지만, 해당 변환이 시스템에서 지원되지 않을 때 발생하는 오류입니다, 내부 함수에서 타입 간의 변환을 시도할 때 발생합니다, 예를 들어, 지원되지 않는 데이터 타입 조합이나 복합 타입 간의 변환을 시도할 때 이 오류가 발생할 수 있습니다.

**ERROR CODE: -632, '%1\$s is not a backup volume.'**

- 이 메시지는 CUBRID 데이터베이스에서 지정된 파일이나 디렉토리가 유효한 백업 볼륨이 아닐 때 발생하는 오류입니다, 백업 파일의 헤더를 읽을 수 없거나, 백업 파일의 매직 넘버가 올바르지 않을 때 나타납니다, CUBRID 백업 파일이 아닌 다른 형식의 파일을 백업 파일로 인식하려고 할 때 발생합니다.

**ERROR CODE: -633, '1\$s is a backup of database %2\$s created on %3\$s instead of given database %4\$s created on %5\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 백업 파일을 복원할 때 발생하는 오류입니다, 백업 파일이 다른 데이터베이스의 백업이거나 다른 시점에 생성된 데이터베이스의 백업일 때 나타납니다, 사용자가 지정한 데이터베이스와 백업 파일에 포함된 데이터베이스 정보가 일치하지 않을 때 발생합니다.

**ERROR CODE: -634, 'A database volume/file was expected in backup.'**

- 이 메시지는 CUBRID 데이터베이스에서 백업 파일을 복원하거나 목록을 확인할 때 발생하는 오류입니다, 백업 파일 내에서 데이터베이스 볼륨이나 파일 정보를 찾을 수 없을 때 나타납니다, 백업 파일의 구조가 손상되었거나 예상된 형식과 다를 때 발생합니다.

**ERROR CODE: -648, 'Backup is incompatible with current "%1\$s" release "%2\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 백업 파일이 현재 CUBRID 버전과 호환되지 않을 때 발생하는 오류입니다, 백업 파일을 생성한 CUBRID 버전과 복구를 시도하는 CUBRID 버전 간의 호환성 문제를 나타냅니다, 이는 백업 파일이 현재 버전에서 지원되지 않을 때 발생하는 오류입니다.

**ERROR CODE: -669, 'Server refused client connection: max clients, (%1\$d), exceeded.'**

- 이 메시지는 CUBRID 데이터베이스 서버가 설정된 최대 클라이언트(max\_clients) 연결 수를 초과하여 새로운 클라이언트 연결을 거부했을 때 발생합니다, 서버의 리소스 보호와 성능 유지를 위해 동시 연결 수를 제한하는 메커니즘입니다, 현재 활성화된 클라이언트 연결 수가 서버에서 허용하는 최대값을 초과했음을 의미합니다.

**ERROR CODE: -675, 'Cannot read the database location file, '%1\$s''**

- 이 메시지는 CUBRID 데이터베이스 시스템이 데이터베이스 위치 파일(예: *databases.txt*)을 읽으려고 시도했지만 실패했을 때 발생하는 오류입니다, 이 파일은 CUBRID가 시스템에 등록된 데이터베이스의 위치와 정보를 파악하는 데 사용됩니다, 파일에 접근할 수 없거나 내용이 손상된 경우 이 오류가 발생합니다.

**ERROR CODE: -676, 'Database '%1\$s', cannot be found in the database location file, '%2\$s''**

- 이 메시지는 CUBRID 데이터베이스 시스템이 특정 데이터베이스를 시작하거나 연결하려고 할 때, 해당 데이터베이스의 정보가 데이터베이스 위치 파일(예: *databases.txt*)에서 발견되지 않아 발생하는 오류입니다, CUBRID는 *databases.txt*와 같은 설정 파일을 통해 시스템에 등록된 데이터베이스의 위치와 정보를 관리합니다, 이 파일에 데이터베이스 정보가 없으면 시스템은 해당 데이터베이스를 인식할 수 없습니다.

**ERROR CODE: -677, 'Failed to connect to database server, '%1\$s', on the following host(s): %2\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 클라이언트가 지정된 호스트(들)에서 데이터베이스 서버에 연결을 시도했지만 실패했을 때 발생하는 오류입니다, 네트워크 연결, 서버 상태, 인증 등 다양한 원인으로 인해 데이터베이스 서버에 연결할 수 없는 상황을 나타냅니다, 자세한 원인은 *csql.err* 파일 또는 *\$CUBRID/log/\*.err* 파일 등을 확인 필요합니다.

**ERROR CODE: -695, 'Wrong interface has been used to kill a client.'**

- 이 메시지는 CUBRID 데이터베이스에서 *killtran* 명령어로 클라이언트 세션을 강제 종료하려고 할 때 잘못된 인터페이스나 부적절한 매개변수가 사용되었을 때 발생합니다, 클라이언트 종료 작업을 수행하기 위해서는 적절한 트랜잭션 인덱스, 사용자명, 호스트명 등의 정보가 필요하지만, 이러한 정보가 부족하거나 잘못된 경우 이 오류가 발생합니다.

**ERROR CODE: -696, 'Unknown transaction (index %1\$d, %2\$s@%3\$s|%4\$d)'**

- 이 메시지는 CUBRID 데이터베이스에서 *killtran* 명령어로 특정 트랜잭션을 강제 종료하려고 시도했으나, 요청된 트랜잭션 인덱스에 해당하는 트랜잭션이 현재 시스템에 존재하지 않을 때 발생합니다.

니다, 주로 클라이언트가 잘못된 트랜잭션 인덱스를 사용하거나, 해당 트랜잭션이 이미 종료되어 시스템에서 제거되었을 때 발생할 수 있습니다.

**ERROR CODE: -697, 'Given transaction index (index %1\$d, %2\$s@%3\$s|%4\$d) does not match with current transaction index (index %5\$d, %6\$s@%7\$s|%8\$d). Likely, the given transaction has finished and its transaction index was re-assigned to another transaction.'**

- 이 메시지는 CUBRID 데이터베이스에서 killtran 명령어로 특정 트랜잭션을 강제 종료 하려고 시도했으나, 요청된 트랜잭션 인덱스와 현재 시스템에서 인식하는 트랜잭션 정보가 일치하지 않을 때 발생합니다, 주로 클라이언트가 오래된(stale) 트랜잭션 인덱스를 가지고 있거나, 서버에서 해당 트랜잭션이 이미 종료되어 인덱스가 다른 트랜잭션에 재할당되었을 때 발생할 수 있습니다.

**ERROR CODE: -743, 'Failed on handshake between client and server. (peer host %1\$s)'**

- 이 메시지는 CUBRID 데이터베이스에서 클라이언트와 서버 간의 초기 연결 설정 과정(handshake)에서 오류가 발생했을 때 나타나는 오류입니다, Handshake는 클라이언트와 서버가 통신을 시작하기 전에 서로의 버전, 기능, 호환성을 확인하는 과정입니다, 이 과정에서 클라이언트의 버전 정보를 받지 못하거나, 서버와 클라이언트 간의 기능 호환성 검사가 실패할 때 발생합니다, 네트워크 연결 문제나 서버 상태 문제로 인해 발생할 수 있습니다.

**ERROR CODE: -752, 'Error restoring backup unit\_num %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스 백업을 복원하는 과정에서, 백업 파일에서 데이터를 읽는 중에 발생하는 일반적인 I/O 오류입니다, 주로 백업 파일의 물리적 손상, 디스크 I/O 오류, 파일 접근 권한 문제, 또는 백업 파일 형식의 불일치 등 다양한 원인으로 발생할 수 있습니다.

**ERROR CODE: -753, 'Error restoring backup unit\_num %1\$d, pageid %2\$d exceeds total number of pages %3\$d for volid %4\$d.'**

- 이 메시지는 CUBRID 데이터베이스 백업을 복원하는 과정에서, 백업 파일 내의 페이지 ID가 복원 대상 볼륨의 유효한 페이지 범위를 벗어났을 때 발생합니다, CUBRID 백업 파일의 메타데이터가 손상되었거나, 백업이 생성될 때와 다른 환경(예: 더 작은 볼륨 크기)으로 복원하려 할 때 발생할 수 있습니다.

**ERROR CODE: -755, '%1\$s cannot be executed in stand-alone mode.'**

- 이 메시지는 CUBRID 데이터베이스에서 스탠드얼론 모드로 실행 중일 때, 클라이언트-서버 통신이 필요한 작업을 시도했을 때 발생합니다, CUBRID 데이터베이스에서 checkpoint와 같은 특정 작업이나 명령이 스탠드얼론(stand-alone) 모드에서는 실행할 수 없을 때 발생하는 이 오류를 나타냅니다.

**ERROR CODE: -757, 'Internal Error: cannot assign permanent OID during object encode.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체를 인코딩하는 과정 중 영구 OID를 할당하지 못했을 때 발생합니다. OID는 데이터베이스 내에서 각 객체를 고유하게 식별하는 중요한 값입니다. 영구 OID 할당 실패는 객체 생성 또는 영속화 과정에 문제가 있음을 의미합니다, 이는 주로 시스템 내부

의 데이터 일관성 문제, 저장 공간 부족, 트랜잭션 문제, 또는 내부 로직 오류 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -767, 'Cannot create a level %1\$d backup without first creating a level %2\$d backup.'**

- 이 메시지는 CUBRID 데이터베이스에서 증분 백업을 수행할 때 백업 레벨의 연속성이 보장되지 않을 때 발생합니다. 이는 백업 레벨을 지정할 때 2는 1이, 1은 0 레벨 백업이 없을 때 오류가 발생하고 CUBRID는 다음과 같은 백업 레벨을 지원합니다:

Level 0 (Full Backup): 전체 데이터베이스 백업 Level 1 (Big Incremental): Level 0 백업을 기반으로 한 증분 백업 Level 2 (Small Incremental): Level 1 백업을 기반으로 한 증분 백업

**ERROR CODE: -769, 'Attribute "%1\$s", of class "%2\$s" is not a valid class or shared attribute.'**

- 이 메시지는 CUBRID 데이터베이스에서 loaddb 명령어를 수행 시 테이블 속성을 처리할 때, 지정된 속성이 유효한 테이블 속성 또는 공유 속성이 아닌 상황임을 나타냅니다. CUBRID의 데이터 로더 메커니즘에서 발생하며, 주로 속성의 네임스페이스 검증 실패로 인해 발생할 수 있습니다.

**ERROR CODE: -771, 'Invalid parameter for function.'**

- 이 메시지는 CUBRID 데이터베이스에서 함수 호출 시 전달된 파라미터가 유효하지 않거나 예상된 형식을 따르지 않을 때 발생합니다. 이는 CUBRID 내부의 특정 함수가 작업을 수행하기 위해 필요한 파라미터의 유효성을 검사하는 과정에서 실패할 경우 오류가 발생합니다.

**ERROR CODE: -778, 'Cannot create a symbolic link "%1\$s" to the file "%2\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 심볼릭 링크를 생성하려고 할 때, 심볼릭 링크 생성에 실패한 상황임을 나타냅니다. 이는 CUBRID의 파일 시스템 관리 메커니즘에서 발생하며, 주로 createdb, copydb, renamedb 명령어 수행 시 발생할 수 있습니다.

**ERROR CODE: -820, 'Cannot initialize a connection.'**

- 이 메시지는 CUBRID 시스템이 클라이언트와 서버 간의 연결(connection)을 초기화하는 데 실패했을 때 발생하는 내부 오류입니다. 연결 초기화는 클라이언트가 서버에 접속할 때 연결 객체를 생성하고, 필요한 리소스를 할당하며, 연결 상태를 설정하는 과정입니다. 이 과정에서 실패했다는 것은 메모리 할당 실패, 클라이언트 ID 할당 실패, 뮤텍스 초기화 실패 등이 발생했음을 의미합니다. 연결 초기화 실패는 클라이언트가 서버에 접속할 수 없게 만들며, 데이터베이스 서비스 이용이 불가능해집니다.

**ERROR CODE: -821, 'Shutdown a connection.'**

- 이 메시지는 CUBRID 시스템이 클라이언트와 서버 간의 연결(connection)을 강제로 종료할 때 발생하는 내부 오류입니다. 이는 오류라기보다는 시스템이 특정 연결을 종료하고 있다는 정보성 메시지입니다. 주로 트랜잭션 중단(slam transaction) 과정에서 발생하며, 해당 연결과 관련된 모든 리소스를 정리하는 과정입니다.

**ERROR CODE: -827, 'Connection list traverse function returned invalid value.'**

- 이 메시지는 CUBRID 시스템이 연결 리스트를 순회하는 과정에서 콜백 함수가 잘못된 반환값을 반환했을 때 발생하는 내부 오류입니다, 잘못된 반환값이 반환되었다는 것은 콜백 함수에 버그가 있거나, 예상치 못한 상황이 발생했음을 의미합니다.

**ERROR CODE: -829, 'This mutex lock has been unlocked already.'**

- 이 메시지는 CUBRID 데이터베이스의 스레드 관리 모듈에서 크리티컬 섹션을 해제할 때 발생합니다, 즉, 이미 해제된 크리티컬 섹션을 다시 해제하려 할 때 발생하는 오류 메시지입니다, 이는 프로그래밍 오류나 동시성 제어 로직의 문제를 나타내며, 뮤텍스 락의 중복 해제를 방지하기 위한 안전 장치입니다.

**ERROR CODE: -831, 'All transaction descriptors(%1\$d) are in use. Raise up max\_clients parameter.'**

- 이 메시지는 CUBRID 데이터베이스 설정에서 max\_clients 설정값을 초과해 연결 요청을 받을 때 발생합니다, 이는 max\_clients 파라미터 설정값 보다 데이터베이스에 동시에 너무 많은 클라이언트가 접속하거나, 기존 클라이언트들이 트랜잭션을 종료하지 않고 계속 유지하고 있을 때 발생할 수 있습니다.

**ERROR CODE: -832, 'Backup is already running. Backup cannot be run by two processes.'**

- 이 메시지는 CUBRID 데이터베이스의 로그 페이지 버퍼 관리 모듈에서 백업(backupdb) 작업의 중복 실행을 방지하기 위해 발생합니다, 데이터베이스 관리자가 백업 작업이 완료되기 전에 또 다른 백업을 시도하거나, 여러 클라이언트에서 동시에 백업을 요청했을 때 발생합니다.

**ERROR CODE: -835, 'Undefined statistics item.'**

- 이 메시지는 CUBRID 데이터베이스의 통계 관리 모듈에서 정의되지 않은 통계 아이템을 참조하려 할 때 발생합니다, CUBRID는 제한된 수의 통계 아이템만을 지원하며, 이 범위를 벗어나는 요청에 대해 이 오류를 반환합니다.

**ERROR CODE: -837, 'Cannot drop the user who owns database objects.'**

- 이 메시지는 CUBRID 데이터베이스의 사용자 관리 모듈에서 사용자를 삭제하려 할 때 발생합니다, 데이터베이스의 무결성과 보안을 보장하기 위한 제약 사항으로, 사용자가 소유한 객체들이 무분별하게 삭제되는 것을 방지합니다. 사용자를 삭제하기 전에 먼저 해당 사용자가 소유한 모든 데이터베이스 객체를 삭제하거나 다른 사용자에게 소유권을 이전해야 합니다.

**ERROR CODE: -841, 'Requested operation can be executed by only sole client(transaction).'**

- 이 메시지는 CUBRID 데이터베이스의 시스템 파라미터 관리 모듈에서 특정 작업을 수행하려 할 때, 여러 클라이언트(트랜잭션)가 동시에 연결되어 있는 상황에서 발생합니다, 특정 시스템 파라미터를 변경하거나 특정 관리 작업을 수행할 때, 데이터 일관성과 안정성을 보장하기 위해 단일 클라이언트 환경에서만 수행해야 하는 작업에 대한 제약 사항을 나타냅니다.



**ERROR CODE: -877, 'No data to be unloaded.'**

- 이 메시지는 CUBRID 데이터베이스에서 스키마나 데이터를 언로드(unload)하는 과정에서 발생합니다. 이런 상황은 일반적으로 데이터베이스에 스키마 정보가 없거나, 언로드 대상 테이블이나 객체가 존재하지 않을 때 나타납니다.

**ERROR CODE: -878, 'Invalid database location file, '%1\$s''**

- 이 메시지는 CUBRID 데이터베이스에서 *databases.txt* 파일을 읽고 파싱하는 과정에서 발생합니다.

이런 상황은 일반적으로 데이터베이스 시작 시 *databases.txt* 파일의 형식이 올바르지 않거나, 필수 정보(데이터베이스 이름, 경로, 호스트, 로그 경로 등)가 누락되었을 때 나타납니다.

**ERROR CODE: -883, 'Failed to get '%1\$s' file status.'**

- 이 메시지는 CUBRID 데이터베이스에서 *renamedb* 명령어 수행 시 특정 파일이나 디렉터리의 상태 정보를 가져오려고 할 때 발생합니다. 이는 파일 시스템 접근 문제, 파일이 존재하지 않는 경우 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -884, 'Volext-pathname '%1\$s' is not permitted to write.'**

- 이 메시지는 CUBRID 데이터베이스에서 *renamedb* 명령어 수행 시 특정 경로에 쓰기 권한이 없을 때 발생합니다. 이는 데이터베이스의 파일 시스템 접근 권한 문제로 인해 데이터베이스 운영에 필수적인 파일 작업이 실패했음을 의미합니다.

**ERROR CODE: -885, 'Failure to move from '%1\$s' to '%2\$s'. Use the same device(disk).'**

- 이 메시지는 CUBRID 데이터베이스에서 *renamedb -E* 옵션과 함께 명령어 수행 시 확장볼륨 파일들을 다른 위치로 이동하려고 할 때 발생합니다. 이는 데이터베이스의 무결성과 성능을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -896, 'Compression failed. Zip Method: %1\$d (%2\$s), Zip Level: %3\$d (%4\$s).'**

- 이 메시지는 CUBRID 데이터베이스에서 LZ4 압축 알고리즘을 사용하여 varchar 데이터 또는 log 데이터를 압축할 때 실패하여 발생합니다.

**ERROR CODE: -897, 'Decompression failed.'**

- 이 메시지는 CUBRID가 LZ4 알고리즘으로 압축된 데이터를 해제하는 과정에서 발생합니다. 객체(varchar) 데이터나 로그 데이터의 압축을 해제할 때 발생할 수 있으며, 데이터 손상이나 메모리 문제로 인한 실패를 나타냅니다.

**ERROR CODE: -910, 'Not allowed access to partition.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 파티션에 접근하려는 시도가 권한 부족, 파티션의 상태 문제, 또는 잘못된 접근 방식으로 인해 거부되었을 때 발생합니다. 즉, 데이터베이스 시스템이



사용자가 요청한 작업(예: 데이터 삽입, 조회, 수정, 삭제 또는 스키마 변경)을 수행하기 위해 특정 파티션에 접근해야 하지만, 해당 파티션에 대한 접근 권한이 없거나 파티션 자체가 접근 불가능한 상태(예: 오프라인, 손상)일 때 발생하는 오류입니다.

이런 상황은 일반적으로 파티션 테이블에 대한 DML/DDL 작업을 수행할 때, 사용자 계정에 필요한 권한이 없거나, 파티션이 손상되어 접근할 수 없거나, 시스템 내부적으로 파티션 관리 로직에 문제가 발생했을 때 나타납니다. 이는 데이터베이스의 보안, 데이터 무결성 및 시스템 안정성을 보호하기 위한 보호성 오류입니다.

#### **ERROR CODE: -941, 'The operation is not supported.'**

- 이 메시지는 CUBRID 데이터베이스의 클라이언트 인터페이스에서 다양한 상황에서 발생할 수 있는 범용 오류 코드입니다, 즉, 요청된 작업이 현재 CUBRID 버전이나 구현에서 지원되지 않는 경우에 사용됩니다.

#### **ERROR CODE: -962, 'Out of memory'**

- 이 메시지는 CUBRID 데이터베이스의 DBlink gateway에서 메모리 할당을 하지 못한 경우 발생합니다.

#### **ERROR CODE: -969, 'Server bit platform (%1\$d) is different from client bit platform (%2\$d).'**

- 이 메시지는 CUBRID 데이터베이스 클라이언트가 서버에 연결을 시도할 때, 클라이언트와 서버가 서로 다른 비트 플랫폼(예: 32비트 클라이언트와 64비트 서버, 또는 그 반대)에서 실행되고 있음을 감지했을 때 발생합니다. CUBRID는 클라이언트와 서버 간의 안정적인 통신과 데이터 일관성을 위해 동일한 비트 플랫폼 환경에서 실행되는 것을 권장하며, 이처럼 비트 플랫폼이 다를 경우 연결을 거부하고 이 오류를 발생시킵니다. 이는 데이터 손상이나 예상치 못한 동작을 방지하기 위한 보호성 오류입니다.

#### **ERROR CODE: -971, 'Program '%1\$s' (pid %2\$d) connected to database server '%3\$s' on the host '%4\$s' (port %5\$d).'**

- 이 메시지는 클라이언트가 CUBRID 데이터베이스 서버에 성공적으로 연결되었을 때 발생하는 알림입니다. 실제 오류가 아닌 정상적인 연결 상태를 나타내는 알림 메시지로, 클라이언트의 연결 정보와 서버 접속 상태를 상세히 보여줍니다.

#### **ERROR CODE: -972, 'Program '%1\$s' (pid %2\$d) was connected from the host '%3\$s'. (transaction index %4\$d).'**

- 이 메시지는 CUBRID 데이터베이스에 클라이언트 프로그램이 성공적으로 연결되었음을 알리는 정보성 로그 메시지입니다. 이는 실제 오류가 아니라, 데이터베이스 서버가 클라이언트의 연결 요청을 수락하고 세션을 설정했음을 기록하는 것입니다. 이 메시지는 시스템 관리자가 데이터베이스에 어떤 클라이언트가 언제, 어디서, 어떤 프로그램으로 연결했는지 추적하고 모니터링하는 데 중요한 정보를 제공합니다. 특히, 비정상적인 연결 시도나 보안 감사 시 유용하게 활용될 수 있습니다.

**ERROR CODE: -973, 'Server status is %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 서버의 현재 상태를 알려주는 정보성 메시지입니다, 이는 실제로는 오류가 아니라 서버의 부팅 과정이나 HA(High Availability) 상태 변경 시 서버의 현재 상태를 로그에 >기록하는 알림 메시지입니다, 서버는 부팅 과정에서 DOWN 상태에서 시작하여 UP 상태로 전환되거나, HA 환경에서 MAINTENANCE 모드로 전환될 때 이 메시지를 출력합니다, 이는 시스템 관리자가 서버의 현재 상태를

를 모니터링하고 문제 발생 시 디버깅할 수 있도록 도와주는 역할을 합니다.

**ERROR CODE: -974, 'Archive log "%1\$s" is created to archive pages from %2\$lld to %3\$lld.'**

- 이 메시지는 실제 오류가 아닌 정보성 메시지로, 데이터베이스 시스템이 특정 범위의 페이지들을 아카이브 로그 파일로 성공적으로 생성했음을 나타냅니다. 이는 데이터베이스의 백업과 복구를 위한 정상적인 로

그 아카이빙 프로세스의 일부입니다.

**ERROR CODE: -977, 'Checkpoint started (previous checkpoint page id: %1\$lld, previous redo page id: %2\$lld).'**

- 이 메시지는 CUBRID 데이터베이스에서 체크포인트 작업이 시작되었음을 알리는 시스템 알림입니다. 데이터베이스의 일관성과 복구 가능성을 보장하기 위한 중요한 작업의 시작을 나타냅니다.

**ERROR CODE: -978, 'Checkpoint finished (checkpoint page id: %1\$lld, checkpoint redo page id: %2\$lld, flushed page count: %3\$d).'**

- 이 메시지는 CUBRID 데이터베이스에서 체크포인트 작업이 완료되었음을 알리는 시스템 알림입니다. 체크포인트 과정에서 처리된 페이지 정보와 함께 작업의 성공적인 종료를 나타냅니다.

**ERROR CODE: -982, 'Set interrupt to the transaction %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 트랜잭션에 대해 인터럽트(중단 신호)가 성공적으로 설정되었음을 나타냅니다, 이는 일반적으로 외부 요인(예: 사용자 요청, 시스템 관리자의 강제 종료 명령, 타임아웃 등)에 의해 해당 트랜잭션의 실행을 중단시키려는 시도가 있었고, 그 시도가 시스템 내부적으로 처리되었음을 의미합니다, 이 메시지는 오류라기보다는 트랜잭션 중단 요청이 처리되었음을 알리는 정보성 메시지에 가깝습니다. 인터럽트가 설정되면 해당 트랜잭션은 현재 수행 중인 작업을 중단하고 롤백될 수 있습니다.

**ERROR CODE: -984, 'Cannot set %3\$s to "%6\$s" in the "[%5\$s]" section when it is set to "%4\$s" in the "[%2\$s]" section of the configuration file "%1\$s".'**

- 이 메시지는 CUBRID 설정 파일(\$CUBRID/conf/cubrid.conf)의 ha\_mode 파라미터 설정값이 다른 값으로 설정되어 있을 때 발생합니다, 이는 설정 파일의 일관성을 보장하고 파라미터 값의 충돌을 방지하기 위한 보호성 오류입니다. 특히 HA(High Availability) 모드와 같은 중요한 파라미터에서 발생할 수 있습니다.

**ERROR CODE: -985, 'The hostname on the database connection string should be specified when multihost is set in "databases.txt". ex) csql demodb@localhost'**

- 이 메시지는 CUBRID 데이터베이스 클라이언트가 *databases.txt* 파일에 *multihost* 옵션이 활성화된 데이터베이스에 연결을 시도할 때 발생합니다. *multihost* 옵션은 데이터베이스가 여러 호스트에 걸쳐 서비스될 수 있음을 나타내며, 이 경우 클라이언트는 연결하려는 특정 서버의 호스트 이름을 명시적으로 지정해야 합니다, 클라이언트가 ``csql demodb``와 같이 데이터베이스 이름만으로 연결을 시도하면, CUBRID는 어떤 호스트에 연결해야 할지 알 수 없으므로 이 오류를 발생시킵니다. 이는 클라이언트가 올바른 서버 레코드에 연결되도록 유도하고, 모호한 연결 시도를 방지하기 위한 보호성 오류입니다.

**ERROR CODE: -991, 'Flush victim candidates of page buffer started.'**

- 이 메시지는 CUBRID 데이터베이스의 페이지 버퍼에서 희생자(victim) 페이지 플러시 작업이 시작되었음을 알리는 시스템 메시지입니다. 메모리 관리를 위해 더티 페이지를 디스크에 기록하는 중요한 작업의 시작을 나타냅니다.

**ERROR CODE: -992, 'Flush victim candidates of page buffer finished (count: %1\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 페이지 버퍼에서 희생자(victim) 페이지 플러시 작업이 완료되었음을 알리는 시스템 메시지입니다. 플러시된 페이지 수와 함께 작업의 성공적인 종료를 나타냅니다.

**ERROR CODE: -994, 'This statement cannot be a prepared statement.'**

- 이 메시지는 CUBRID 데이터베이스에서 `prepare from` 구문에서 `prepared statement` 구문(`prepare`, `execute`, `deallocated/drop`)을 사용하려는 경우 발생합니다.

**ERROR CODE: -995, 'A prepared statement with the name %1\$s does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 `EXECUTE` 문을 사용하여 특정 이름의 `prepared statement`를 실행하려고 할 때 발생합니다, `Prepared statement`는 `PREPARE` 문을 통해 미리 준비된 SQL 문으로, `EXECUTE` 문을 통해 실행됩니다. 하지만 요청한 이름의 `prepared statement`가 현재 세션에 존재하지 않는 경우 이 오류가 발생합니다.

이는 다음과 같은 상황에서 발생할 수 있습니다, `prepared statement`가 아직 생성되지 않았거나, 세션이 종료되어 `prepared statement`가 삭제되었거나, `prepared statement` 이름이 잘못 입력되었거나, 다른 세션에서 생성된 `prepared statement`를 현재 세션에서 실행하려고 할 때입니다, `Prepared statement`는 세션별로 관리되므로, 세션이 변경되면 이전에 준비된 `statement`에 접근할 수 없습니다.

**ERROR CODE: -1001, 'An unexpected condition or state has been reached. "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 내부적으로 예상치 못한 상태나 조건을 만났을 때 발생합니다, 개발자가 예측하지 못했거나, 특정 오류 코드를 할당하기 어려운 일반적인 내부 오류 상황에 사용되는 포괄적인 오류 코드입니다, 보통 구체적인 오류 상황에 대한 설명을 제공하여 문제

의 원인을 추적하는 데 도움을 줍니다. 이 메시지는 데이터베이스의 논리적 결함, 데이터 손상, 비정상적인 시스템 상태 또는 환경 문제로 인해 발생할 수 있습니다.

**ERROR CODE: -1009, 'COMPACTDB already started.'**

이 오류 코드는 이미 실행 중인 COMPACTDB 프로세스가 있을 때 새로운 COMPACTDB 작업을 시도할 경우 발생합니다.

**ERROR CODE: -1066, 'Session expired.'**

- 이 메시지는 CUBRID 데이터베이스에서 클라이언트 세션이 유효 기간을 초과하여 만료되었을 때 발생합니다. 세션은 클라이언트와 서버 간의 연결 상태를 유지하고 사용자별 정보를 저장하는 논리적인 단위입니다. 세션이 만료되면 해당 세션을 통해 진행 중이던 모든 작업이 중단되고, 클라이언트는 데이터베이스에 대한 접근 권한을 잃게 됩니다. 이는 주로 세션 타임아웃 설정, 클라이언트의 비활성 상태 유지, 또는 네트워크 연결 불안정 등으로 인해 발생할 수 있습니다. 데이터베이스의 보안과 리소스 관리를 위한 보호성 오류입니다.

**ERROR CODE: -1069, 'Too many prepared statements.'**

- 이 메시지는 CUBRID 데이터베이스에서 단일 세션이 허용된 최대 준비된 문장(prepared statements) 개수를 초과하여 문장을 준비하거나 사용하려고 할 때 발생합니다. 이는 PREPARE STMT 구문에서 데이터베이스 시스템은 과도한 메모리 사용이나 리소스 고갈을 방지하기 위해 세션당 준비된 문장의 개수를 제한합니다. 이 메시지는 주로 애플리케이션이나 쿼리가 예상보다 많은 준비된 문장을 사용하거나, 시스템의 `MAX\_PREPARED\_STMT\_COUNT`와 같은 관련 파라미터 설정이 현재 워크로드에 비해 너무 낮게 설정되어 있을 때 나타납니다. 이는 데이터베이스의 안정성과 리소스 관리를 위한 보호성 오류입니다.

**ERROR CODE: -1070, 'Session variable '@%1\$s' not defined.'**

- 이 메시지는 CUBRID 데이터베이스에서 현재 세션이 존재하지 않거나 정의되지 않은 세션 변수를 사용하려고 시도할 때 발생합니다. 세션 변수는 특정 사용자 세션 내에서만 유효한 임시 변수이며, 데이터베이스 시스템이 요청된 변수를 찾을 수 없을 때 이 오류를 반환합니다. 이는 주로 애플리케이션이나 쿼리에서 변수 이름의 오타, 변수 미정의, 또는 변수의 유효 범위(스코프)를 벗어난 접근 시도 등으로 인해 발생합니다. 이 메시지는 존재하지 않는 데이터에 대한 작업을 방지하고 데이터 무결성을 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -1071, 'Too many session variables.'**

- 이 메시지는 CUBRID 데이터베이스에서 단일 세션이 허용된 최대 세션 변수 개수를 초과하여 변수를 생성하거나 사용하려고 할 때 발생합니다. 세션 변수는 특정 사용자 세션 내에서만 유효한 임시 변수로, 데이터베이스 시스템은 과도한 메모리 사용이나 리소스 고갈을 방지하기 위해 세션 변수의 개수를 제한합니다.

**ERROR CODE: -1072, 'IP address(%1\$s) is not authorized.'**

- 이 메시지는 CUBRID 데이터베이스 서버가 클라이언트의 IP 주소가 접근 제어 목록(Access Control List, ACL)에 허용되지 않는다고 판단했을 때 발생합니다. 즉, 데이터베이스에 연결을 시도

하는 IP 주소가 서버의 보안 정책에 의해 차단되었음을 의미합니다. 이는 데이터베이스의 무단 접근을 방지하고 보안을 강화하기 위한 보호성 오류입니다.

#### **ERROR CODE: -1073, 'Invalid format in file(%1\$s).'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 특정 파일(특히 IP 접근 제어 파일)을 읽고 파싱하는 과정에서, 해당 파일의 내용이 예상된 형식이나 문법을 따르지 않을 때 발생합니다. 이는 주로 파일 내용의 오타, 잘못된 구문, 또는 비정상적인 수정으로 인해 발생할 수 있습니다. 이 오류가 발생하면 CUBRID는 해당 파일의 설정을 올바르게 적용할 수 없으므로, 서비스 시작이 실패하거나 예상치 못한 접근 제어 문제가 발생할 수 있습니다. 이는 시스템의 보안 및 안정적인 운영에 필수적인 설정 파일의 무결성 문제로, 즉각적인 확인과 수정이 필요합니다.

#### **ERROR CODE: -1074, 'Cannot read access list file(%1\$s).'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 특정 접근 목록 파일(Access List File)을 열거나 읽는데 실패했을 때 발생합니다. 접근 목록 파일은 CUBRID의 보안 및 접근 제어 정책을 정의하는 데 사용되는 중요한 설정 파일입니다. 이 메시지는 주로 파일이 존재하지 않거나, 파일 경로가 잘못되었거나, CUBRID 프로세스를 실행하는 사용자에게 해당 파일에 대한 읽기 권한이 없거나, 파일 자체가 손상되었을 때 나타납니다. 이 오류가 발생하면 CUBRID는 올바른 접근 제어 정책을 적용할 수 없으므로, 서비스 시작이 실패하거나 예상치 못한 접근 문제가 발생할 수 있습니다.

#### **ERROR CODE: -1076, 'Could not load system parameter.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 시작 또는 초기화 과정에서 필수적인 시스템 파라미터(예: *cubrid.conf* 파일에 정의된 설정)를 로드하는 데 실패했을 때 발생합니다. 시스템 파라미터는 데이터베이스의 동작 방식, 리소스 할당, 운영 정책 등을 구성하는 데 매우 중요합니다. 이러한 파라미터를 로드하지 못하면 시스템이 제대로 자신을 구성할 수 없어, 시작 실패나 비정상적인 동작으로 이어집니다. 이는 데이터베이스가 올바르게 기능하는 것을 방해하는 치명적인 오류입니다.

#### **ERROR CODE: -1077, 'The '%1\$s' parameter at line %2\$d in file '%3\$s' : Unknown parameter'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 *cubrid.conf*와 같은 설정 파일을 읽고 파싱하는 과정에서, 시스템이 인식하거나 지원하지 않는 파라미터 이름을 발견했을 때 발생합니다. 이는 일반적으로 파라미터 이름의 오타, 현재 CUBRID 버전에서 더 이상 지원되지 않는 파라미터의 사용, 또는 설정 파일의 비정상적인 수정으로 인해 발생할 수 있습니다. 이 오류가 발생하면 CUBRID는 해당 파라미터를 처리할 수 없으므로, 서버나 관련 서비스가 정상적으로 시작되지 않거나 예상치 못한 방식으로 동작할 수 있습니다. 이는 시스템의 안정적인 운영에 필수적인 설정 파일의 무결성 문제로, 즉각적인 확인과 수정이 필요합니다.

#### **ERROR CODE: -1078, 'Character at offset %1\$d is invalid with current codeset.'**

- 이 메시지는 CUBRID 데이터베이스에서 *loaddb* 명령어 실행 시 현재 설정된 문자 집합(codeset)으로 처리할 수 없는 유효하지 않은 문자를 발견했을 때 발생합니다. 이는 내부 문자열 처리 과정에서 데이터의 실제 인코딩과 시스템의 *codeset* 설정이 일치하지 않아 발생하며, 데이터 무결성 및 문자열 처리의 정확성을 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -1087, 'Backup active log '%1\$s' started.'**

- 이 메시지는 CUBRID 데이터베이스에서 활성 로그 파일의 백업 작업이 시작되었음을 알리는 정보 알림 메시지입니다. 이는 시스템이 로그 백업 프로세스를 시작하여 데이터의 안정성과 복구 가능성을 확보하기 위한 작업을 진행하고 있음을 의미합니다. 일반적으로 데이터베이스 관리 시스템에서 중요한 작업의 시작을 기록하는 정보성 메시지입니다.

**ERROR CODE: -1088, 'Backup active log '%1\$s' finished.'**

- 이 메시지는 CUBRID 데이터베이스에서 활성 로그 파일의 백업 작업이 성공적으로 완료되었음을 알리는 성공 알림 메시지입니다. 이는 시스템이 정상적으로 로그 백업을 수행하여 데이터의 안정성과 복구 가능성을 확보했음을 의미합니다. 일반적으로 데이터베이스 관리 시스템에서 중요한 작업이 성공적으로 마무리되었을 때 기록되는 정보성 메시지입니다.

**ERROR CODE: -1090, 'Locale initialization: %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스가 로케일(Locale) 관련 정보를 초기화하는 과정에서 문제가 발생했을 때 나타납니다. 로케일은 언어, 국가, 문자 인코딩 규칙 등을 정의하여 데이터의 정렬, 문자열 비교, 문자 집합 변환 등을 올바르게 수행하도록 돕는 중요한 설정입니다. 이 과정에서 오류가 발생하면 시스템의 로케일 설정, CUBRID의 구성, 또는 운영체제의 로케일 지원 기능에 문제가 있음을 의미합니다. 이는 문자 집합 처리, 국제화 기능, 전반적인 데이터 무결성에 영향을 미칠 수 있는 중요한 오류입니다.

**ERROR CODE: -1091, 'Locale generation: %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스가 로케일(Locale) 관련 정보를 생성하거나 처리하는 과정에서 문제가 발생했을 때 나타납니다. 로케일은 언어, 국가, 문자 인코딩 규칙 등을 정의하여 데이터의 정렬, 문자열 비교, 문자 집합 변환 등을 올바르게 수행하도록 돕는 중요한 설정입니다. 이 과정에서 오류가 발생하면 시스템의 로케일 설정, CUBRID의 구성, 또는 운영체제의 로케일 지원 기능에 문제가 있음을 의미합니다. 이는 문자 집합 처리, 국제화 기능, 전반적인 데이터 무결성에 영향을 미칠 수 있는 중요한 오류입니다.

**ERROR CODE: -1093, 'The arg '%1\$s' for inet\_aton is not a valid string formatted ipv4 address.'**

- 이 메시지는 CUBRID 데이터베이스에서 *inet\_aton* 함수를 사용하여 문자열 형식의 IP 주소를 숫자 형식으로 변환하려고 할 때, 입력된 인수가 유효한 IPv4 주소 문자열 형식이 아닐 경우 발생합니다. *inet\_aton* 함수는 "a.b.c.d" 형식의 문자열 IPv4 주소를 네트워크 바이트 순서의 32비트 정수형으로 변환하는 표준 라이브러리 함수입니다. 이 메시지는 주로 잘못된 데이터 입력, 데이터 타입 불일치, 또는 비정상적인 값으로 인해 발생하며, IP 주소 변환이 필요한 작업(예: 네트워크 관련 함수, 로그 기록, 특정 데이터 처리)이 실패했음을 의미합니다. 이는 데이터 유효성 검사 실패로 인한 보호성 오류입니다.

**ERROR CODE: -1094, 'The arg '%1\$lld' for inet\_ntoa is not a valid number formatted ipv4 address.'**

- 이 메시지는 CUBRID 데이터베이스에서 *inet\_ntoa* 함수를 사용하여 숫자 형식의 IP 주소를 문자열 형식으로 변환하려고 할 때, 입력된 인수가 유효한 IPv4 주소 형식이 아닐 경우 발생합니다. *inet\_ntoa* 함수는 네트워크 바이트 순서의 32비트 정수형 IPv4 주소를 "a.b.c.d" 형식의 문자열로



변환하는 표준 라이브러리 함수입니다. 이 메시지는 주로 잘못된 데이터 입력, 데이터 타입 불일치, 또는 비정상적인 값으로 인해 발생하며, IP 주소 변환이 필요한 작업(예: 네트워크 관련 함수, 로그 기록, 특정 데이터 처리)이 실패했음을 의미합니다.

#### **ERROR CODE: -1095, 'User name is too long.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 사용자 이름(User Name)을 생성하거나 변경할 때, 입력된 사용자 이름의 길이가 시스템에서 허용하는 최대 길이를 초과했을 때 발생합니다, 즉, CUBRID 서버가 사용자 관리 작업을 처리하는 과정에서, *CREATE USER* 또는 *ALTER USER*와 같은 명령을 통해 제공된 사용자 이름이 내부적으로 정의된 최대 문자열 길이를 초과하는 경우 이 오류를 반환합니다. 이는 데이터베이스의 스키마 및 시스템 메타데이터의 일관성을 유지하고, 내부 버퍼 오버플로우와 같은 잠재적인 문제를 방지하기 위한 보호성 제약입니다.

#### **ERROR CODE: -1099, 'DDL statement is not allowed by configuration (block\_ddl\_statement=yes).n SQL Text: %1\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 DDL(Data Definition Language) 문을 실행하려고 할 때 발생하는 오류입니다, 즉, 시스템 파라미터 *block\_ddl\_statement*가 *yes*로 설정되어 있을 때, 테이블 생성, 수정, 삭제 등의 스키마 변경 작업을 차단하는 보호 기능입니다, 이는 데이터베이스 스키마의 안정성과 보안을 보장하기 위한 보호성 오류로, 의도하지 않은 스키마 변경을 방지합니다, 주로 *CREATE*, *ALTER*, *DROP* 등의 DDL 문이 실행될 때 *pt\_is\_ddl\_statement* 함수로 DDL 문인지 확인하고, 시스템 파라미터 설정에 따라 실행을 허용하거나 차단합니다.

#### **ERROR CODE: -1100, 'Statement without WHERE clause is not allowed by configuration (block\_nowhere\_statement=yes).'**

- 이 메시지는 CUBRID 데이터베이스에서 WHERE 절이 없는 DELETE 문을 실행하려고 할 때 발생하는 오류입니다, 즉, 시스템 파라미터 *block\_nowhere\_statement*가 *yes*로 설정되어 있을 때, 조건 없이 모든 레코드를 삭제하는 위험한 SQL 문의 실행을 차단하는 보호 기능입니다, 이는 데이터 무결성과 시스템 안정성을 보장하기 위한 보호성 오류로, 실수로 전체 테이블의 데이터가 삭제되는 것을 방지합니다, 주로 *DELETE* 문에서 *search\_cond* (검색 조건)이 NULL인 경우를 감지하여 발생하며, 시스템 파라미터 설정에 따라 실행을 허용하거나 차단합니다.

#### **ERROR CODE: -1102, 'Unknown system parameter or bad value.'**

- 이 메시지는 CUBRID 데이터베이스에서 시스템 파라미터를 설정하거나 변경하려고 할 때 발생하는 변경값이 없는 경우 오류입니다, 즉, 존재하지 않는 시스템 파라미터를 참조하거나, 파라미터의 값이 허용된 범위나 형식을 벗어났을 때 발생합니다, 이는 시스템 안정성과 데이터 무결성을 보장하기 위한 보호성 오류입니다.

#### **ERROR CODE: -1103, 'Cannot change system parameter.'**

- 이 메시지는 CUBRID 데이터베이스에서 시스템 파라미터를 변경하려고 할 때 발생하는 변경값이 없는 경우 오류입니다, 즉, 데이터베이스 실행 중에 동적으로 변경할 수 없는 파라미터를 변경하려고 시도했거나, 변경 권한이 없는 파라미터를 수정하려고 했을 때 발생합니다, 이는 시스템 안정성과 데이터 무결성을 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -1104, 'Slow query (%1\$d msec)n%2\$s'**

- 이 메시지는 실제로는 오류가 아니라 CUBRID 데이터베이스에서 성능 모니터링을 위한 정보성 메시지입니다, 즉, 설정된 임계값보다 오래 걸린 쿼리가 실행되었을 때 시스템이 이를 감지하여 성능 분석을 위한 상세 정보를 제공하는 것입니다, 이는 데이터베이스 성능 최적화와 문제 진단을 위한 중요한 모니터링 기능입니다, 시스템은 쿼리 실행 시간, SQL 문, 실행 계획, 버퍼 통계, 대기 시간 등의 상세 정보를 수집하여 성능 병목 지점을 파악할 수 있도록 도와줍니다.

**ERROR CODE: -1115, 'Started to update statistics (class "%1\$s", oid : %2\$d|%3\$d|%4\$d).'**

- 이 메시지는 실제로는 오류가 아니라 CUBRID 데이터베이스의 통계정보 갱신 *update statistics* 작업이 시작되었음을 알리는 정보성 메시지입니다, 즉, 테이블의 통계정보(인덱스 통계, 컬럼 분포도, 카디널리티 등)를 업데이트하는 작업이 시작되었음을 의미합니다, 이는 쿼리 최적화를 위해 필요한 통계정보를 수집하고 갱신하는 과정이 시작되었음을 나타내는 긍정적인 메시지입니다, 통계정보 갱신은 테이블의 데이터 분포, 인덱스 구조, 컬럼 값의 분포 등을 분석하여 쿼리 실행 계획을 최적화하기 위한 중요한 작업입니다.

**ERROR CODE: -1116, 'Finished to update statistics (class "%1\$s", oid : %2\$d|%3\$d|%4\$d, error code : %5\$d).'**

- 이 메시지는 실제로는 오류가 아니라 CUBRID 데이터베이스의 통계정보 갱신 *update statistics* 작업이 완료되었음을 알리는 정보성 메시지입니다, 즉, 테이블의 통계정보(인덱스 통계, 컬럼 분포도, 카디널리티 등)를 업데이트하는 작업이 정상적으로 종료되었음을 의미합니다, 이는 쿼리 최적화를 위해 필요한 통계정보를 수집하고 갱신하는 과정이 완료되었음을 나타내는 긍정적인 메시지입니다,

**ERROR CODE: -1117, 'Cannot change attribute "%1\$s". CUBRID cannot change an attribute as a SHARED and vice versa.'**

- 이 메시지는 CUBRID에서 이미 정의된 속성의 SHARED 특성을 변경하려고 할 때 발생합니다. SHARED 속성은 테이블 정의 시점에 결정되며, 이후에는 변경이 불가능한 특성입니다.

**ERROR CODE: -1123, 'pthread\_cond\_timedwait() timed out.'**

- 이 메시지는 CUBRID 데이터베이스 내부에서 사용하는 쓰레드(thread)가 critical section (예 : buffer latch 등)을 기다리는 동안 지정된 시간을 초과했을 때 발생합니다.

**ERROR CODE: -1130, 'Transaction is aborted because transaction lock counts exceeds %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션이 보유한 잠금의 개수가 시스템에서 설정된 임계값을 초과했을 때 발생합니다, 즉, \$CUBRID/conf/cubrid.conf 환경설정 파라미터에서 *rollback\_on\_lock\_escalation* 설정값이 true이고,, lock escalation 설정된 임계값을 초과했을 때 발생합니다, 이런 상황은 일반적으로 대량의 데이터를 처리하는 쿼리에서 많은 수의 행 잠금이 필요할 때 발생할 수 있습니다.



**ERROR CODE: -1132, 'Encryption library failure: %1\$s'**

- 이 메시지는 CUBRID dblink server에서 암호 관련 암복화 시 발생하는 오류입니다.

**ERROR CODE: -1135, 'A previous link of B+tree(%1\$s) is corrupted.: vpid = (%2\$d, %3\$d)'**

- 이 메시지는 CUBRID 데이터베이스 checkdb -repair-prev-link 명령어 수행 시 b-tree 손상을 감지했다는 경고성 메세지입니다.

**ERROR CODE: -1136, 'A previous link of B+tree(%1\$s) is repaired.: vpid = (%2\$d, %3\$d)'**

- 이 메시지는 CUBRID 데이터베이스 checkdb -repair-prev-link 명령어 수행 시 감지된 b-tree 손상을 수정했다는 경고성 메세지입니다.

**ERROR CODE: -1138, 'Handshake error (peer host %1\$s): incompatible interruptibility. (client: %2\$s, server: %3\$s)'**

- 이 메시지는 CUBRID 데이터베이스 클라이언트와 서버 간의 네트워크 연결을 설정하는 핸드셰이크 과정에서, 양측의 '인터럽트 가능성(interruptibility)' 설정이 서로 호환되지 않을 때 발생합니다, 즉, 클라이언트와 서버가 트랜잭션 또는 쿼리 실행 중 인터럽트(예: Ctrl+C 또는 KILL TRANSACTION)를 처리하는 방식에 대한 약속이 일치하지 않아 통신 연결을 수립할 수 없음을 의미합니다, 이는 네트워크 통신의 일관성과 안정성을 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -1145, 'To change the owner of a system class is not allowed.'**

- 이 메시지는 CUBRID 데이터베이스에서 시스템 내부적으로 중요한 역할을 하는 '시스템 테이블'의 소유자를 변경하려고 시도할 때 발생합니다, 즉, 데이터베이스의 핵심 구조와 보안을 유지하기 위해, 일반 사용어나 특정 권한을 가진 사용자라도 시스템 테이블의 소유권을 임의로 변경하는 것을 허용하지 않는 보호성 오류입니다, 이런 상황은 일반적으로 `ALTER CLASS ... CHANGE OWNER TO ...`와 같은 DDL(Data Definition Language) 문을 사용하여 시스템 테이블에 대해 소유자 변경을 시도할 때 발생합니다, 이는 데이터베이스의 안정성과 무결성을 보호하기 위한 중요한 보안 정책입니다.

**ERROR CODE: -1152, 'The owner or a member of DBA group is only allowed to kill the transaction (%1\$d).'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 트랜잭션을 강제로 종료(kill)하려고 할 때, 해당 작업을 수행할 권한이 없는 사용자가 시도했을 때 발생합니다, 즉, 트랜잭션 종료는 데이터베이스의 안정성과 보안을 위해 제한된 권한을 가진 사용자만 수행할 수 있도록 설계되어 있습니다, 이런 상황은 일반적으로 일반 사용자가 다른 사용자의 트랜잭션을 종료하려고 시도하거나, 시스템 트랜잭션을 종료하려고 할 때 발생합니다, 이는 데이터베이스의 보안 정책과 무결성을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -1158, 'Serializable conflict due to concurrent updates'**

- 이 메시지는 CUBRID 데이터베이스에서 SERIALIZABLE 격리 수준으로 실행되는 트랜잭션이 다른 동시 트랜잭션의 업데이트 작업과 충돌하여 직렬화 가능성(serializability)이 깨졌을 때 발생합니다

다. *SERIALIZABLE* 격리 수준은 동시 실행되는 트랜잭션들이 마치 순차적으로 실행된 것처럼 동일한 결과를 보장하는 가장 엄격한 수준입니다, 이 메시지는 현재 트랜잭션이 접근하려는 데이터가 다른 트랜잭션에 의해 변경되었거나, 팬텀 읽기(phantom read)와 같은 직렬화 불가능한 현상이 감지되었을 때 발생하며, 데이터 일관성을 유지하기 위해 충돌하는 트랜잭션 중 하나를 롤백시킵니다. 이는 *SERIALIZABLE* 격리 수준의 엄격한 특성으로 인해 발생하는 정상적인 동작입니다.

#### **ERROR CODE: -1159, 'Timezone compile error: %s'**

- 이 메시지는 CUBRID 데이터베이스가 시간대(timezone) 데이터를 내부적으로 처리하거나 “컴파일”하는 과정에서 문제가 발생했음을 나타냅니다, 시간대 규칙 및 데이터를 파싱하고 내부적으로 사용할 수 있는 형태로 변환하는 로직을 포함합니다. 이 메시지는 주로 시스템에 설치된 시간대 데이터 파일이 손상되었거나, 형식이 잘못되었거나, CUBRID가 해당 데이터를 해석하는 데 실패했을 때 발생합니다. 이는 데이터베이스가 시간대 관련 연산을 정확하게 수행할 수 없게 만들며, 날짜/시간 데이터의 유효성 및 일관성에 영향을 미칠 수 있는 중요한 문제입니다.

#### **ERROR CODE: -1160, 'Timezone library loader error: %s'**

- 이 메시지는 CUBRID 데이터베이스가 시간대(timezone) 관련 연산을 수행하기 위해 필요한 시간대 라이브러리 또는 데이터를 로드하는 데 실패했음을 나타냅니다. CUBRID는 날짜/시간 데이터의 정확한 처리, 특히 *DATETIME WITH TIME ZONE* 타입이나 *TIMESTAMP* 값의 시간대 변환을 위해 시스템의 시간대 정보에 의존합니다, 이 메시지는 주로 시간대 데이터 파일이 없거나, 손상되었거나, 접근 권한이 없거나, 또는 시간대 관련 환경 설정이 잘못되었을 때 발생합니다. 시간대 라이브러리 로드 실패는 데이터베이스가 시간대 관련 기능을 정상적으로 수행할 수 없게 만들며, 이는 데이터베이스 시작 실패나 날짜/시간 관련 쿼리 및 연산의 오류로 이어질 수 있는 문제입니다.

#### **ERROR CODE: -1161, 'Timezone conversion error.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜/시간 값을 한 시간대에서 다른 시간대로 변환하는 과정에서 내부적으로 예상치 못한 문제가 발생했음을 나타냅니다. CUBRID는 내부적으로 UTC(협정 세계시)를 기준으로 시간을 관리하며, 사용자 세션의 시간대 설정에 따라 표시되는 시간을 변환합니다. 이 메시지는 주로 다음과 같은 상황에서 발생할 수 있습니다:

#### **ERROR CODE: -1162, 'Invalid or missing timezone.'**

- 이 메시지는 CUBRID 데이터베이스에서 시간대(timezone) 정보가 유효하지 않거나, 날짜/시간 관련 연산을 수행하는 데 필요한 시간대 정보가 누락되었을 때 발생합니다. CUBRID는 날짜/시간 데이터의 정확한 처리와 변환을 위해 유효한 시간대 정보가 필수적입니다.

#### **ERROR CODE: -1163, 'Invalid or missing daylight saving time.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜/시간 값을 처리할 때, 일광 절약 시간(Daylight Saving Time, DST) 정보가 유효하지 않거나 필요한 DST 정보가 누락되었을 때 발생합니다. CUBRID는 시간대 변환 및 날짜/시간 계산의 정확성을 위해 DST 규칙을 엄격하게 적용합니다. 이 메시지는 주로 다음과 같은 상황에서 발생할 수 있습니다:

**ERROR CODE: -1165, 'Invalid combination of date, time, timezone and daylight specifier.'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜, 시간, 시간대(timezone) 및 일광 절약 시간(Daylight Saving Time, DST) 지정자들 간의 조합이 논리적으로 유효하지 않을 때 발생합니다. CUBRID는 시간대 관련 연산에서 입력된 날짜/시간과 시간대 정보, 그리고 DST 규칙이 서로 일치하는지 검증합니다. 예를 들어, 특정 날짜와 시간이 주어진 시간대에서 DST 규칙과 충돌하거나, 존재하지 않는 시간 조합(예: DST 전환 구간에서 존재하지 않는 시간)을 지정했을 때 이 오류가 발생합니다. 이는 데이터의 정확성과 일관성을 보장하기 위한 검증 메커니즘으로, 잘못된 시간대 변환이나 계산을 방지합니다.

**ERROR CODE: -1166, 'The specified combination of date and time do not exist (during daylight saving interval).'**

- 이 메시지는 CUBRID 데이터베이스에서 사용자가 지정한 날짜와 시간 조합이 특정 시간대(timezone)의 일광 절약 시간(Daylight Saving Time, DST) 전환 구간으로 인해 실제로는 존재하지 않는 시간일 때 발생합니다. 예를 들어, DST가 시작되어 시계가 새벽 2시에서 새벽 3시로 한 시간 앞으로 점프하는 경우, 새벽 2시 0분부터 2시 59분까지의 시간은 해당 날짜에 존재하지 않습니다. 사용자가 이러한 존재하지 않는 시간을 입력하거나 계산하려고 시도할 때, CUBRID는 데이터의 유효성을 보장하기 위해 이 오류를 발생시킵니다. 이는 주로 날짜/시간 데이터 삽입, 업데이트, 또는 날짜/시간 함수 사용 시 발생할 수 있습니다.

**ERROR CODE: -1167, 'Comment string cannot have more than 1024 bytes.'**

- 이 메시지는 CUBRID 데이터베이스에서 사용자나 객체에 대한 코멘트(주석) 문자열의 길이가 허용된 최대 크기인 1024 바이트를 초과했을 때 발생하는 오류를 나타냅니다. 코멘트는 데이터베이스 사용자 객체(테이블, 컬럼, 사용자 등)에 대한 설명이나 메모를 저장하는 데 사용되며, 시스템의 안정성과 성능을 위해 길이 제한이 있습니다, 즉, 이 메시지는 CUBRID 데이터베이스의 사용자 인증 시스템에서 사용자 코멘트를 설정할 때 발생합니다.

**ERROR CODE: -1168, 'Unable to update statistics on table (%1\$s) because a lock is not immediately granted.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 테이블의 통계(statistics)를 업데이트하려고 시도했지만, 필요한 잠금(lock)을 즉시 획득하지 못하여 작업이 실패했음을 나타냅니다. 통계 업데이트는 쿼리 최적화기가 최적의 실행 계획을 수립하는 데 필수적인 정보를 제공합니다. 이 메시지는 주로 통계 업데이트 대상 테이블에 다른 트랜잭션이 배타적인 잠금을 보유하고 있거나, 잠금 경합이 심하여 통계 업데이트 작업이 필요한 잠금을 제때 얻지 못할 때 발생합니다. 이는 통계 업데이트 작업이 지연되거나 실패하여, 이후 실행되는 쿼리의 성능에 부정적인 영향을 미칠 수 있음을 의미합니다.

**ERROR CODE: -1177, 'Incompatible timezone data: %1\$s has different checksum from %2\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 클라이언트와 서버 간의 시간대(timezone) 데이터가 서로 호환되지 않음을 나타내는 오류입니다. CUBRID는 시간대 관련 연산의 정확성을 보장하기 위해 클라이언트와 서버가 동일한 시간대 데이터를 사용해야 합니다. 이 메시지는 클라이언트와 서버의 시간대 데이터 체크섬이 다를 때 발생하며, 이는 서로 다른 버전의 시간대 데이터를 사용하고 있거나

나, 시간대 데이터가 손상되었음을 의미합니다. 시간대 데이터의 불일치는 날짜/시간 변환 함수의 부정확한 결과를 초래할 수 있으므로, 데이터베이스는 이러한 상황에서 오류를 발생시켜 데이터 무결성을 보호합니다.

**ERROR CODE: -1179, 'Found not vacuumed entries in heap.'**

- 이 메시지는 CUBRID checkdb 명령어 수행 시 발생할 수 있는 오류로 힙 파일(테이블의 실제 데이터가 저장되는 공간) 내에 진공(vacuum) 처리되지 않은 엔트리(레코드)가 존재함을 나타내는 정보성 또는 경고성 메시지입니다. 데이터베이스에서 레코드가 삭제되거나 업데이트될 때, 해당 공간은 즉시 재사용되지 않고 “죽은(dead)” 공간으로 표시됩니다. 진공 작업은 이러한 죽은 공간을 회수하고 재사용 가능하게 정리하는 역할을 합니다, 이 메시지는 진공(vacuum) 작업이 필요하거나, 특정 검사 과정에서 아직 정리되지 않은 공간이 발견되었음을 알려줍니다. 이는 데이터베이스의 일관성 문제라기보다는 공간 효율성 및 성능 유지보수와 관련된 정보입니다.

**ERROR CODE: -1180, 'Found not vacuumed OIDs. INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d | %4\$d | %5\$d).'**

- 이 메시지는 CUBRID checkdb 명령어 수행 시 발생할 수 있는 오류로 인덱스에 진공(vacuum) 처리되지 않은(즉, 삭제되었지만 아직 공간이 회수되지 않은) 객체 ID(OID)가 남아 있음을 나타내는 정보성 또는 경고성 메시지입니다. 데이터베이스에서 레코드가 삭제되면 해당 레코드의 OID는 즉시 재사용되지 않고, 진공 작업을 통해 비활성화되거나 재사용 가능한 상태로 정리됩니다, 이 메시지는 특정 인덱스에 이러한 “정리되지 않은” OID가 존재함을 알려주며, 이는 인덱스 공간의 비효율적인 사용이나 잠재적인 성능 저하를 의미할 수 있습니다. 일반적으로는 데이터베이스의 일관성 문제라기보다는 유지보수와 관련된 정보입니다.

**ERROR CODE: -1186, 'checksumdb: %1\$s (error code: %2\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 *checksumdb* 유틸리티를 실행하는 동안 내부적으로 문제가 발생했음을 나타냅니다. *checksumdb*는 데이터베이스 파일의 체크섬을 검사하여 데이터 손상 여부를 확인하는 도구입니다, 이 메시지는 체크섬 검사 과정에서 파일 접근 문제, 데이터 불일치, 내부 로직 오류 등 다양한 원인으로 인해 유틸리티가 정상적으로 작업을 완료하지 못했음을 의미합니다.

**ERROR CODE: -1187, 'Invalid access on page %1\$d,%2\$d which was part of a deleted heap.'**

- 이 메시지는 CUBRID 데이터베이스에서 이미 삭제(해제)된 힙 파일(테이블의 실제 데이터 저장소)에 속했던 페이지에 접근하려고 할 때 발생합니다. , 즉, 논리적으로나 물리적으로 삭제된 테이블(혹은 오브젝트)의 데이터 페이지를 참조하거나 읽으려 할 때 시스템이 이를 감지하여 차단하는 보호성 오류입니다, 이런 상황은 일반적으로 테이블 DROP, TRUNCATE, 또는 내부적으로 힙 파일이 삭제된 후, 해당 페이지에 대한 잘못된 참조가 남아있을 때 발생합니다.

**ERROR CODE: -1188, 'Dropping an active user '%1\$s' is not allowed.'**

- 이 메시지는 CUBRID 데이터베이스에서 *DROP USER* 수행시 트랜잭션 수행 중인 USER를 DROP 할 수 없을 때 발생합니다.

**ERROR CODE: -1189, 'The specified combination of date and time do not exist (during transition of timezone offset rules).'**

- 이 메시지는 CUBRID 데이터베이스에서 날짜 및 시간 값을 처리할 때, 특정 시간대(timezone)의 오프셋 규칙이 변경되는 전환 기간(예: 일광 절약 시간제(DST) 시작 또는 종료 시점)에 존재하지 않는 날짜와 시간 조합이 입력되었을 때 발생합니다. 예를 들어, DST가 시작되어 시계가 새벽 2시에서 새벽 3시로 건너뛰는 경우, 새벽 2시 30분과 같은 시간은 해당 날짜에 물리적으로 존재하지 않게 됩니다. 이 메시지는 이러한 비존재 시간을 감지하여 데이터의 유효성을 보호하기 위해 발생합니다.

**ERROR CODE: -1190, 'Stand-alone vacuum execution is started.'**

- 이 메시지는 CUBRID 데이터베이스에서 독립 실행형(stand-alone) 진공(vacuum) 작업이 시작되었음을 알리는 정보성 메시지입니다. 이는 오류가 아니라, 데이터베이스의 공간을 회수하고 성능을 최적화하는 진공 작업이 정상적으로 시작되었음을 사용자에게 알려주는 로그 메시지입니다.

**ERROR CODE: -1191, 'Stand-alone vacuum execution is ended.'**

- 이 메시지는 CUBRID 데이터베이스에서 독립 실행형(stand-alone) 진공(vacuum) 작업이 성공적으로 완료되었음을 알리는 정보성 메시지입니다. 이는 오류가 아니라, 데이터베이스의 공간을 회수하고 성능을 최적화하는 진공 작업이 정상적으로 종료되었음을 사용자에게 알려주는 로그 메시지입니다.

**ERROR CODE: -1194, 'Unable to check file %1\$d|%2\$d because lock on class %3\$d|%4\$d|%5\$d is not immediately granted.'**

- 이 메시지는 CUBRID checkdb 명령어에서 object (heap, btree 등) 검사 중 해당 object의 lock 획득 오류로 특정 파일(일반적으로 데이터 페이지 파일)을 검사하거나 접근하려 할 때, 해당 파일이 속한 테이블(테이블)에 대한 잠금(lock)을 즉시 획득하지 못하여 발생하는 문제입니다. 이는 주로 동시성 제어와 관련된 상황에서 발생하며, 다른 트랜잭션이 해당 테이블에 대한 배타적 잠금을 보유하고 있거나, 잠금 요청이 너무 많아 즉시 처리되지 못할 때 나타날 수 있습니다. 시스템이 파일의 무결성을 확인하거나 안전하게 접근하기 위해 잠금이 필요하지만, 이를 얻지 못해 작업을 진행할 수 없음을 의미합니다.

**ERROR CODE: -1241, 'XASL tree needs recompile.'**

- 이 메시지는 CUBRID 데이터베이스 시스템의 쿼리 처리 과정에서 XASL(eXtensible Abstract Syntax Language) 트리가 재컴파일되어야 함을 나타내는 내부적인 요청 또는 오류입니다. XASL은 CUBRID의 내부 쿼리 표현 형식으로, SQL 쿼리가 파싱되고 최적화된 후 실행 계획으로 변환되는 중간 단계입니다. 이 실행 계획은 성능 향상을 위해 캐시될 수 있습니다. 이 메시지는 캐시된 XASL 트리가 더 이상 유효하지 않거나, 최적화된 실행 계획을 다시 생성해야 하는 상황이 발생했음을 의미합니다. 이는 주로 테이블 스키마 변경, 통계 정보 업데이트, 또는 시스템 내부 상태 변화와 같은 이유로 발생합니다. 일반적으로 사용자에게 직접적으로 노출되는 심각한 오류라기보다는, CUBRID 내부에서 쿼리 처리의 일관성과 최적화를 유지하기 위한 메커니즘의 일부로 발생합니다.

**ERROR CODE: -1247, 'The time(%1\$s) specified must be after the time(%2\$s) of the specified backup.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 복구 작업을 수행하는 동안, 사용자가 지정한 복구 시점이 사용하려는 백업의 생성 시점 보다 이전이거나 같을 때 발생하는 오류입니다. 데이터베이스 복구는 특정 백업 시점부터 시작하여 이후의 트랜잭션 로그를 적용하여 데이터베이스를 특정 시점(Point-In-Time Recovery)으로 되돌리거나 최신 상태로 복원하는 과정입니다. 복구 시점은 항상 백업 시점보다 미래여야 합니다. 백업 시점 이전으로 복구하는 것은 논리적으로 불가능하며, 백업 시점과 동일한 시점으로 복구하는 것은 백업 자체를 사용하는 것과 같으므로, 일반적으로 백업 시점 이후의 특정 시점으로 복구할 때 이 오류가 발생할 수 있습니다.

**ERROR CODE: -1248, 'Invalid Key file : %1\$s'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption)에 사용되는 키 파일을 읽거나 처리하는 과정에서 해당 키 파일이 유효하지 않거나 손상되었음을 감지했을 때 발생하는 오류입니다.

**ERROR CODE: -1249, 'Cannot find the key (index: %1\$d).'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 과정에서 특정 인덱스에 해당하는 마스터 키를 찾으려고 시도했지만, 해당 키를 찾을 수 없어 실패했음을 나타내는 오류입니다.

**ERROR CODE: -1250, 'The master key set on the database and what is given don't match. The key index set on the database: %1\$d'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 과정에서, 데이터베이스에 설정된 마스터 키 정보(특히 키 인덱스)와 현재 시스템에 제공된 마스터 키 정보가 일치하지 않을 때 발생하는 오류입니다. TDE는 데이터베이스의 데이터를 암호화하여 보안을 강화하는 기능이며, 마스터 키는 데이터 암호화 키(Data Encryption Key, DEK)를 암호화하는 데 사용되는 최상위 키입니다. 이 마스터 키가 일치하지 않으면 암호화된 데이터에 접근할 수 없습니다. 이 메시지는 주로 TDE 키 파일의 내용이 변경되었거나, 잘못된 키 파일이 사용되었거나, 데이터베이스의 메타데이터에 기록된 키 인덱스와 현재 로드된 키의 인덱스가 다를 때 발생합니다.

**ERROR CODE: -1251, 'Error while TDE-encrypting'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하여 데이터를 암호화하는 과정에서 예상치 못한 오류가 발생하여 암호화 작업에 실패했음을 나타내는 오류입니다. TDE는 데이터베이스의 데이터를 암호화하여 보안을 강화하는 기능으로, 데이터가 디스크에 기록되기 전에 암호화되고 읽힐 때 복호화됩니다. 이 메시지는 이러한 암호화 과정에서 문제가 발생했음을 의미합니다. 암호화 실패는 데이터가 제대로 저장되지 않거나, 데이터 무결성이 손상되거나, 데이터베이스의 정상적인 작동이 방해될 수 있음을 시사합니다.

**ERROR CODE: -1252, 'Error while TDE-decrypting'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하여 암호화된 데이터를 복호화하는 과정에서 예상치 못한 오류가 발생하여 복호화 작업에 실패했음을 나타내는 오



류입니다, TDE는 데이터베이스의 데이터를 암호화하여 보안을 강화하는 기능으로, 데이터가 디스크에 기록되기 전에 암호화되고 읽힐 때 복호화됩니다. 이 메시지는 이러한 복호화 과정에서 문제가 발생했음을 의미합니다, 복호화 실패는 암호화된 데이터에 접근할 수 없거나, 데이터 무결성이 손상되거나, 데이터베이스의 정상적인 작동이 방해될 수 있음을 시사합니다.

**ERROR CODE: -1253, 'TDE Module is not loaded.'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하려고 할 때, TDE 모듈이 제대로 로드되지 않았거나 초기화되지 않았음을 나타내는 오류입니다, 즉, TDE KEY 파일 없이 데이터베이스가 구동되었거나 잘못된 TDE KEY 파일을 가지고 구동되어 TDE 모듈이 제대로 동작되지 않는 상태에서, TDE 관련 작업을 수행하는 경우 발생합니다.

**ERROR CODE: -1254, 'Cannot create a key.'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하여 암호화 키를 생성하는 과정에서 예상치 못한 오류가 발생하여 키 생성 작업에 실패했음을 나타내는 오류입니다, TDE는 데이터베이스의 데이터를 암호화하여 보안을 강화하는 기능으로, 이 과정에서 마스터 키나 데이터 암호화 키(DEK) 등의 키가 필요합니다. 이 메시지는 이러한 키 생성 과정에서 문제가 발생했음을 의미합니다, 키 생성 실패는 TDE 기능이 정상적으로 작동하지 않거나, 데이터 보안에 문제가 발생할 수 있음을 시사합니다.

**ERROR CODE: -1255, 'Cannot load TDE module. You can't use TDE feature.n'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하려고 할 때, 데이터베이스 시작 시 TDE KEY 파일이 없어서 TDE 기능을 사용할 수 없다는 정보를 알려주는 메시징입니다.

**ERROR CODE: -1256, 'Cannot copy key file (\_keys). You can't use TDE feature on the copied database.'**

- 이 메시지는 CUBRID 데이터베이스를 copydb 명령어로 복사하는 과정에서 TDE(Transparent Data Encryption)에 사용되는 키 파일(\_keys)을 복사하는데 실패했음을 나타내는 오류입니다,

**ERROR CODE: -1258, 'Cannot find the key set on the database from %1\$s. The key file from backup volume is going to be used.'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 데이터베이스를 시작하거나 접근할 때, 데이터베이스에 설정된 마스터 키 파일(\_keys 파일)을 지정된 경로에서 찾을 수 없음을 나타냅니다, 하지만, 시스템은 다행히 백업 볼륨 내에서 해당 키 파일을 발견했으며, 이 백업 키 파일을 사용하여 데이터베이스를 계속 운영할 것임을 알리는 경고성 메시지입니다.

**ERROR CODE: -1259, 'Rename the key file from "%1\$s" to "%2\$s".'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 데이터베이스의 키 파일을 복구하거나 복원하는 과정에서, 이전 키 파일로 변경한다는 경고성 메시지입니다.

**ERROR CODE: -1260, 'Copy the key file "%1\$s" to "%2\$s"'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 데이터베이스를 복원하는 과정에서, 백업에 있던 키 파일로 복원했다는 경고성 메세지입니다.

**ERROR CODE: -1261, 'The first key (key index: %1\$d, created time: %2\$s) on the restored key file has been set.'**

- 이 메시지는 CUBRID 데이터베이스 복원 과정에서 TDE(Transparent Data Encryption) 키 파일이 성공적으로 복원되었고, 해당 키 파일 내의 첫 번째 마스터 키(지정된 인덱스와 생성 시간을 가짐)가 데이터베이스에 설정되었음을 나타내는 정보성 또는 경고성 메시지입니다, TDE는 데이터베이스의 데이터를 암호화하여 보안을 강화하는 기능이며, 마스터 키는 데이터 암호화 키(Data Encryption Key, DEK)를 암호화하는 데 사용되는 최상위 키입니다. 이 메시지는 복원된 데이터베이스가 TDE 기능을 사용하여 암호화된 데이터에 접근할 준비가 되었음을 의미합니다, 일반적으로 긍정적인 결과이지만, 사용자가 의도한 키가 맞는지 확인하는 것이 중요합니다.

**ERROR CODE: -1262, 'The key file is full.'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하는 과정에서, TDE KEY를 새롭게 추가할때 최대 키 저장 개수를 넘을때 발생합니다.

**ERROR CODE: -1263, 'It fails to TDE-encrypt the log page (pageid: %1\$lld). It won't be tried to encrypt this page any longer.'**

- 이 메시지는 CUBRID 시스템이 TDE(Transparent Data Encryption) 기능을 사용하여 데이터베이스의 로그 페이지를 암호화하는 과정에서 예상치 못한 오류가 발생하여 암호화 작업에 실패했음을 나타내는 오류입니다, 로그 페이지는 데이터베이스의 모든 변경 사항을 기록하는 매우 중요한 부분이며, TDE가 활성화된 환경에서는 이 로그 페이지도 보안을 위해 암호화되어야 합니다.

**ERROR CODE: -1286, 'Cannot find the log lsa at the time to extract. (input time : %1\$s).'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 특정 시간의 로그 LSA(Log Sequence Address)를 찾으려고 할 때, 해당 시간에 해당하는 로그 LSA를 찾을 수 없음을 나타내는 오류입니다, CDC는 데이터베이스의 변경 사항을 실시간으로 캡처하고 추적하는 기능이며, LSA는 로그 시퀀스 주소로 데이터베이스의 특정 시점을 식별하는 데 사용됩니다.

**ERROR CODE: -1287, 'Timed out attempting to extract log info. (elapsed time : %1\$d sec, extraction timeout : %2\$d sec)'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 추출하는 과정에서 설정된 시간 초과 값을 초과하여 작업이 중단되었음을 나타내는 오류입니다.

**ERROR CODE: -1289, 'Skip producing log info for invalid transaction. (trid : %1\$d)'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 처리하는 과정에서, 유효하지 않은 트랜잭션을 발견하여 해당 트랜잭션에 대한 로그 정보 생성을 건너뛰는 의미입니다.



**ERROR CODE: -1290, 'Invalid log lsa (%1\$Ild|%2\$d) to extract log info.'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 추출하려고 할 때, 유효하지 않은 로그 LSA(Log Sequence Address)를 만났음을 나타내는 오류입니다, CDC는 데이터베이스의 변경 사항을 실시간으로 캡처하고 추적하는 기능이며, LSA는 로그 시퀀스 주소로 데이터베이스의 특정 시점을 식별하는 데 사용됩니다, 이 메시지는 주로 지정된 LSA가 데이터베이스의 로그 범위를 벗어났거나, 로그 파일이 손상되었거나, LSA 형식이 잘못되었을 때 발생합니다.

**ERROR CODE: -1291, 'Cannot connect to server for extracting log info. Try again with setting 'supplemental\_log' parameter.'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 추출하려고 할 때, 서버에 연결할 수 없음을 나타내는 오류입니다, CDC는 데이터베이스의 변경 사항을 실시간으로 캡처하고 추적하는 기능이며, 이 기능을 사용하려면 서버와의 연결이 필요합니다.

**ERROR CODE: -1292, 'Cannot find the log lsa at the time (%1\$s). Log lsa is adjusted to the time (%2\$s).'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 특정 시간의 로그 LSA(Log Sequence Address)를 찾으려고 할 때, 해당 시간에 정확히 일치하는 LSA를 찾을 수 없어서 가장 가까운 유효한 시간의 LSA로 조정했음을 나타내는 경고성 메시지입니다, CDC는 데이터베이스의 변경 사항을 실시간으로 캡처하고 추적하는 기능이며, LSA는 로그 시퀀스 주소로 데이터베이스의 특정 시점을 식별하는 데 사용됩니다, 이 메시지는 시스템이 요청된 시간에 정확히 일치하는 로그를 찾지 못했지만, 가장 가까운 유효한 시간의 로그를 찾아서 작업을 계속할 수 있도록 조정했음을 의미합니다.

**ERROR CODE: -1293, 'Cannot get log page while producing log infos. Log lsa to generate log info indicates NULL.'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 생성하는 과정에서, 로그 페이지를 가져오려고 할 때 로그 LSA(Log Sequence Address)가 NULL 값을 가지고 있어서 작업을 수행할 수 없음을 나타내는 오류입니다,

**ERROR CODE: -1294, 'Log info has been generated. (type : "%1\$s").'**

- 이 메시지는 CUBRID 시스템이 CDC(Change Data Capture) 기능을 사용하여 로그 정보를 성공적으로 생성했음을 나타내는 정보성 메시지입니다, CDC는 데이터베이스의 변경 사항을 실시간으로 캡처하고 추적하는 기능이며, 로그 정보는 이러한 변경 사항을 기록한 데이터입니다, 이 메시지는 일반적으로 성공적인 작업을 나타내며, 생성된 로그 정보의 유형을 포함하여 어떤 종류의 로그 정보가 생성되었는지를 알려줍니다.

**ERROR CODE: -1296, 'Log recovery: ANALYSIS Phase is started.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 첫 번째 단계인 '분석(ANALYSIS) 단계'가 시작되었음을 알리는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해

필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되지 않은 트랜잭션을 롤백하고, 커밋되었지만 디스크에 반영되지 않은 변경 사항을 재반영하는 역할을 합니다, 분석 단계에서는 로그 파일을 스캔하여 마지막 체크포인트 이후의 모든 트랜잭션 활동을 파악하고, 어떤 트랜잭션이 커밋되었고 어떤 트랜잭션이 롤백되어야 하는지 등의 정보를 수집합니다.

**ERROR CODE: -1297, 'Log recovery: REDO Phase is started. Log pages to redo: %1\$lld, Log records to redo: %2\$lld.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 두 번째 단계인 '재실행(REDO) 단계'가 시작되었음을 알리는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해 필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되었지만 디스크에 반영되지 않은 변경 사항들을 로그에 기록된 순서대로 데이터베이스에 재반영하는 역할을 합니다, 재실행 단계에서는 이전 '분석 단계'에서 파악된 정보를 바탕으로, 커밋된 트랜잭션의 변경 사항들을 데이터 파일에 적용합니다, 이 메시지는 재실행할 로그 페이지의 수와 로그 레코드의 수를 함께 제공하여 복구 작업의 규모를 알려줍니다.

**ERROR CODE: -1298, 'Log recovery: UNDO Phase is started. Log pages to undo: %1\$lld, transactions to undo: %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 세 번째 단계인 '취소(UNDO) 단계'가 시작되었음을 알리는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해 필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되지 않은 트랜잭션을 롤백하는 역할을 합니다, 취소 단계에서는 이전 '분석 단계'에서 파악된 정보를 바탕으로, 커밋되지 않은 트랜잭션의 변경 사항들을 되돌립니다, 메시지는 취소할 로그 페이지의 수와 취소할 트랜잭션의 수를 함께 제공하여 복구 작업의 규모를 알려줍니다.

**ERROR CODE: -1299, 'Log recovery: %1\$s Phase is being finished up.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 특정 단계(예: ANALYSIS, REDO, UNDO)가 성공적으로 마무리되고 있음을 알리는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해 필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되지 않은 트랜잭션을 롤백하고, 커밋되었지만 디스크에 반영되지 않은 변경 사항을 재반영하는 역할을 합니다, 이 메시지는 각 복구 단계가 성공적으로 진행되고 있음을 나타내며, 전체 복구 프로세스가 순조롭게 진행되고 있음을 시사합니다.

**ERROR CODE: -1300, 'Log recovery: %1\$s Phase is finished.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 특정 단계(예: ANALYSIS, REDO, UNDO)가 성공적으로 완료되었음을 알리는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해 필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되지 않은 트랜잭션을 롤백하고, 커밋되었지만 디스크에 반영되지 않은 변경 사항을 재반영하는 역할을 합니다, 이 메시지는 각 복구 단

계가 성공적으로 완료되었음을 나타내며, 전체 복구 프로세스가 순조롭게 진행되고 있음을 시사합니다.

**ERROR CODE: -1301, 'Log recovery: %1\$s progress: (%2\$lld/%3\$lld), %4\$.2f percent, elapsed time: %5\$.2f (s), estimated remaining time: %6\$.2f (s).'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 비정상적으로 종료된 후 재시작되거나, 특정 복구 작업이 시작될 때, 트랜잭션 로그 복구 과정의 특정 단계(ANALYSIS, REDO, UNDO)의 현재 진행 상황을 실시간으로 알려주는 정보성 메시지입니다, 로그 복구는 데이터베이스의 일관성과 무결성을 보장하기 위해 필수적인 과정으로, 시스템 충돌 등으로 인해 커밋되지 않은 트랜잭션을 롤백하고, 커밋되었지만 디스크에 반영되지 않은 변경 사항을 재반영하는 역할을 합니다, 이 메시지는 복구 작업의 현재 상태, 완료된 작업량, 전체 작업량, 진행률, 경과 시간 및 예상 남은 시간을 상세하게 제공하여 관리자가 복구 진행 상황을 모니터링할 수 있도록 돕습니다.

**ERROR CODE: -1302, 'dblink %1\$s'**

- 이 메시지는 CUBRID의 DBLink 기능에서 발생하는 일반적인(포괄적) 에러 메시지로, 다양한 하위 오류 상황에서 사용됩니다, 이는 parser 등에서 dblink 관련 오류 발생시 사용하는 오류코드로 실제 오류의 원인은 여러 메시지가 치환되어 전달되는 상세 메시지에 따라 달라지며, 네트워크 통신, 인증, 파라미터, 쿼리 실행 등 DBLink 관련 모든 계층에서 발생할 수 있습니다.

**ERROR CODE: -1303, 'dblink invalid number of columns specified.'**

- 이 메시지는 CUBRID에서 DBLink를 통해 외부 DB에 쿼리를 실행할 때, 지정한 컬럼 개수가 실제 쿼리 결과의 컬럼 개수와 일치하지 않거나, 시스템에서 요구하는 조건을 만족하지 않을 때 발생합니다, 예를 들어, SELECT 쿼리에서 반환되는 컬럼 개수와 DBLink 객체에 지정된 컬럼 개수가 다르거나, 컬럼 매핑이 잘못된 경우에 발생할 수 있습니다.

**ERROR CODE: -1304, 'dblink catalog \_db\_server class not found.'**

- 이 메시지는 CUBRID에서 DBLink 관련 작업(서버 객체 생성, 조회, 수정 등)을 수행할 때, 내부적으로 참조하는 시스템 카탈로그 테이블(\_db\_server 테이블)이 데이터베이스에 존재하지 않을 경우 발생합니다, 즉, DBLink 기능의 핵심 정보 저장소가 손상되었거나, 제품 버그로 비정상적인 경우입니다.

**ERROR CODE: -1305, 'dblink server "%1\$s" not found.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체를 참조(예: 연결, 수정, 삭제 등)하려고 할 때, 지정한 이름의 서버 객체가 데이터베이스에 존재하지 않을 경우 발생합니다, 즉, 서버명이 잘못 입력되었거나, 이미 삭제되었거나, 아직 생성되지 않은 경우입니다.

**ERROR CODE: -1306, 'dblink server "%1\$s" already exists.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체를 생성(CREATE SERVER)하려고 할 때, 지정한 이름의 서버 객체가 이미 존재할 경우 발생합니다, 즉, 동일한 서버명이 이미 등록되어 있어 중복 생성이 불가능한 상황입니다.

**ERROR CODE: -1307, 'dblink Cannot update server object.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체의 정보를 수정(UPDATE)하려고 할 때, 시스템 정책 또는 권한 제한 등으로 인해 해당 작업이 허용되지 않을 때 발생합니다, 예를 들어, onwer, dba 및 dba member가 아닌 경우 권한 부족, 또는 서버 객체의 일관성 보장 등의 이유로 UPDATE가 거부될 수 있습니다.

**ERROR CODE: -1308, 'dblink not supported type %1\$s.'**

- 이 메시지는 CUBRID에서 DBLink를 통해 외부 DB에 쿼리를 실행할 때, 사용된 데이터 타입이 CUBRID DBLink 기능에서 지원하지 않는 타입일 경우 발생합니다, 예를 들어, 외부 DB의 특정 데이터 타입(예: 대용량 바이너리, 특수 타입 등)이 CUBRID DBLink에서 매핑/변환/처리가 불가능할 때 발생합니다.

**ERROR CODE: -1309, 'dblink invalid bind param.'**

- 이 메시지는 CUBRID에서 DBLink를 통해 외부 DB에 쿼리를 실행할 때, 전달된 바인드 파라미터가 올바르지 않을 경우 발생합니다.

예를 들어, 파라미터 개수가 맞지 않거나, 타입이 다르거나, 값이 NULL/비정상 등 허용되지 않는 경우에 발생할 수 있습니다.

**ERROR CODE: -1310, 'dblink password length exceeds max size.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체를 생성하거나 수정할 때, 입력한 비밀번호의 길이가 시스템에서 허용하는 최대 크기를 초과할 경우 발생합니다, 이는 비밀번호가 너무 길게 입력되었거나, 인코딩 문제로 인해 실제 바이트 수가 초과된 경우에도 발생할 수 있습니다.

**ERROR CODE: -1311, 'dblink encrypted password length is incorrect.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체의 암호화된 비밀번호를 복호화하거나 검증하는 과정에서, 암호화된 문자열의 길이가 시스템이 요구하는 길이와 다를 때 발생합니다, 이는 암호화된 비밀번호가 손상되었거나, 잘못된 방식으로 생성/저장된 경우, 또는 DBLink 객체 생성 시 잘못된 암호문이 입력된 경우에 발생할 수 있습니다.

**ERROR CODE: -1312, 'dblink the checksum of the encrypted password does not match.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체의 암호화된 비밀번호를 복호화하거나 검증하는 과정에서, 암호화된 문자열의 체크섬 값이 기대값과 다를 때 발생합니다, 이는 암호화된 비밀번호가 손상되었거나, 암호화/복호화 환경(알고리즘, 키 등)이 일치하지 않거나, 데이터가 잘못 저장/전송된 경우에 발생할 수 있습니다.

**ERROR CODE: -1313, 'dblink invalid cipher string format.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체의 비밀번호를 복호화하거나 검증하는 과정에서, 암호화된 문자열(암호문)의 형식이 올바르지 않을 때 발생합니다, 암호화된 문자열이 손상되었거나, 포맷이 맞지 않거나, 예상과 다른 방식으로 저장된 경우(예: 길이 부족, 구분자 누락, 인코딩 오류 등)에 발생할 수 있습니다.

**ERROR CODE: -1314, 'dblink Failed to decryption password. error=%1\$d''**

- 이 메시지는 CUBRID에서 DBLink 서버 객체를 통해 외부 DB에 접속하려 할 때, 저장된 비밀번호를 복호화하는 과정에서 내부적으로 실패했을 때 발생합니다, 이는 암호화 라이브러리의 문제, 암호화/복호화 키 불일치, 암호화된 데이터 손상, 시스템 환경 문제, 내부 버그 등 다양한 원인으로 인해 복호화가 정상적으로 수행되지 않을 때 반환됩니다.

**ERROR CODE: -1315, 'dblink Failed to encryption password. error=%1\$d''**

- 이 메시지는 CUBRID에서 DBLink 서버 객체를 생성하거나 수정할 때, 외부 DB 접속용 비밀번호를 암호화하는 과정에서 내부적으로 실패했을 때 발생합니다, 이는 암호화 라이브러리의 문제, 입력 값 오류, 시스템 리소스 부족, 암호화 알고리즘 미지원, 내부 버그 등 다양한 원인으로 인해 암호화가 정상적으로 수행되지 않을 때 반환됩니다.

**ERROR CODE: -1317, 'dblink Not allowed "%1\$s" Server.'**

- 이 메시지는 CUBRID에서 DBLink 서버 객체에 대해 허용되지 않은 조작(ALTER 등)을 시도할 때 발생합니다, 이는 특정 시스템 정책, 권한, 또는 DBLink 서버의 속성에 따라 해당 서버에 대한 변경(ALTER, DROP 등)이 금지되어 있을 수 있습니다.

예를 들어, 시스템에서 보호되는 서버이거나, 현재 세션/사용자에게 해당 서버를 변경할 권한이 없는 경우에 발생합니다.

**ERROR CODE: -1318, '[%1\$5.5s][%2\$d] %3\$s'**

- 이 메시지는 CUBRID의 Gateway(DBLink 외부 DB 연동, ODBC 등) 기능에서 외부 데이터베이스(ODBC 드라이버 등)로부터 에러가 발생했을 때, 해당 에러의 상세 정보를 그대로 포매팅하여 전달하는 메시지입니다. 즉, CUBRID 내부가 아닌 외부 DBMS 또는 ODBC 드라이버에서 발생한 에러의 SQLSTATE, 에러 코드, 상세 메시지를 그대로 보여줍니다.
- 이 메시지는 CUBRID가 DBLink 외부 DB 연동 중 발생한 문제의 원인을 직접 해석하지 않고, 외부 시스템의 에러 정보를 그대로 전달하는 역할을 합니다.

**ERROR CODE: -1319, 'Parameter binding error.'**

- 이 메시지는 CUBRID의 Gateway(DBLink 외부 DB 연동, ODBC 등) 기능에서 SQL 쿼리 실행 시, 파라미터(입력 변수) 바인딩 과정에서 문제가 발생했을 때 출력됩니다. 주로 바인딩할 파라미터의 개수가 맞지 않거나, 타입이 일치하지 않거나, 내부적으로 바인딩 핸들(구조체 등)이 올바르게 생성되지 않은 경우에 발생합니다. 예를 들어, 쿼리에서 3개의 파라미터가 필요하지만 2개만 바인딩했거나, 바인딩 값의 타입이 DB 컬럼 타입과 맞지 않을 때 발생할 수 있습니다.

**ERROR CODE: -1320, 'Invalid ODBC handle.'**

- 이 메시지는 CUBRID의 Gateway(DBLink 외부 DB 연동, ODBC 등) 기능에서 ODBC 핸들(식별자, 포인터 등)이 유효하지 않을 때 발생합니다. 즉, 이미 닫힌(해제된) 핸들을 참조하거나, 올바르게 생성되지 않은 핸들을 사용할 때, 또는 내부적으로 핸들이 손상되었을 때 발생할 수 있습니다. 주로 연결 해제 후 재사용, 핸들 누락, 메모리 손상, 드라이버 버그 등에서 발생합니다.

**ERROR CODE: -1322, 'not supported type %1\$s(%2\$d).'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 외부 데이터베이스와의 연동 또는 쿼리 실행 시, CUBRID가 지원하지 않는 데이터 타입이 사용되었을 때 발생합니다. 예를 들어, 외부 DB의 테이블에 CUBRID에서 인식할 수 없는 타입(예: BLOB, CLOB, 사용자 정의 타입 등)이 포함되어 있거나, 타입 매핑이 정의되어 있지 않은 경우입니다. 이 메시지는 주로 데이터 타입 호환성 문제, 타입 변환 로직 미구현, 또는 드라이버/연동 모듈의 한계로 인해 발생합니다.

**ERROR CODE: -1323, 'Invalid Statement handle.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 SQL 문을 관리하는 Statement 핸들(식별자, 포인터 등)이 유효하지 않을 때 발생합니다. 즉, 이미 닫힌 Statement를 참조하거나, 올바르게 생성되지 않은 핸들을 사용할 때, 또는 내부적으로 핸들이 손상되었을 때 발생할 수 있습니다. 주로 Statement 해제 후 재사용, 핸들 누락, 메모리 손상, 드라이버 버그 등에서 발생합니다.

**ERROR CODE: -1324, 'Invalid db connection handle.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 데이터베이스 연결을 관리하는 핸들(식별자, 포인터 등)이 유효하지 않을 때 발생합니다. 즉, 이미 닫힌 연결을 참조하거나, 올바르게 생성되지 않은 핸들을 사용할 때, 또는 내부적으로 핸들이 손상되었을 때 발생할 수 있습니다. 주로 연결 해제 후 재사용, 핸들 누락, 메모리 손상, 드라이버 버그 등에서 발생합니다.

**ERROR CODE: -1325, 'Link server connection url do not exist.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 외부 데이터베이스에 연결을 시도할 때, 연결에 필요한 URL(접속 정보)이 정의되어 있지 않거나, 시스템에서 해당 정보를 찾을 수 없을 때 발생합니다. 즉, DBLink 또는 Gateway 서버 객체를 생성할 때 연결 URL을 누락했거나, 내부적으로 해당 정보가 정상적으로 등록되지 않은 경우입니다.

**ERROR CODE: -1326, 'Invalid numeric value.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 질의에 잘못된 numeric 타입 값 처리 시 발생합니다.

**ERROR CODE: -1327, 'Invalid precision value of "%1\$s" column.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 외부 DB의 테이블 또는 쿼리 결과를 처리할 때, 특정 컬럼의 정밀도(precision) 값이 허용 범위를 벗어나거나, 잘못된 값(음수, 0, 너무 큰 값 등)으로 지정된 경우에 발생합니다. 주로 외부 DB의 스키마 정의가 잘못되었거나, ODBC 드라이버가 컬럼의 정밀도 정보를 잘못 반환할 때 발생할 수 있습니다.

**ERROR CODE: -1329, 'Not supported dbms.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 연결을 시도한 외부 DBMS가 CUBRID에서 공식적으로 지원하지 않는 DBMS일 때 발생합니다. 즉, CUBRID Gateway가 인식할 수 없는 DBMS 유형이거나, 지원 목록에 포함되지 않은 DBMS에 연결을 시도할 때 발생합니다.



다. 예를 들어, CUBRID Gateway가 Oracle, MySQL, MSSQL 등 일부 DBMS만 지원하는데, 지원하지 않는 DBMS(예: SQLite, MariaDB, 특수 목적 DB 등)에 연결을 시도하면 이 오류가 반환됩니다.

**ERROR CODE: -1330, 'Unable to allocate an environment handle.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 외부 데이터베이스와의 연결을 시작할 때, 환경 핸들(ODBC 환경 핸들 등)을 생성(할당)하는 과정에서 실패했을 때 발생합니다. 환경 핸들은 외부 DB와의 세션, 연결, 트랜잭션 등 모든 작업의 기반이 되는 리소스이므로, 할당에 실패하면 이후의 모든 DB 연동 작업이 불가능해집니다. 주로 시스템 메모리 부족, ODBC 드라이버/라이브러리 문제, 내부 리소스 고갈 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -1331, 'A null column binder was referenced.'**

- 이 메시지는 CUBRID의 Gateway(DBlink 외부 DB 연동, ODBC 등) 기능에서 SQL 쿼리 실행 결과를 애플리케이션 변수에 바인딩하는 과정에서, 컬럼 바인더(바인딩 핸들)가 NULL(초기화되지 않음) 상태로 참조될 때 발생합니다. 즉, 컬럼 바인더가 올바르게 생성/할당되지 않았거나, 이미 해제된(Free) 상태에서 접근하려고 할 때 발생하는 프로그래밍 오류입니다. 주로 DBlike 외부 DB 연동, ODBC 드라이버, 또는 내부 바인딩 로직의 문제로 인해 발생할 수 있습니다.

**ERROR CODE: -1332, 'The active log volume (%1\$s) is too sane to recreate. If you remove the active log, it could cause an unexpected and irreversible problem.'**

- 이 메시지는 CUBRID에서 *cubrid emergencylog -r* 명령어를 사용 시 active log가 정상인 경우 재생성할 수 없다고 발생합니다. 이는 활성 로그(active log) 볼륨을 재생성(recreate)하려고 할 때, 해당 로그 파일이 손상되지 않고 정상(sane) 상태이기 때문에 시스템이 재생성을 허용하지 않을 때 발생합니다.

**ERROR CODE: -1333, 'Invalid time (%1\$s) to start flashback. Time is required to be set between (%2\$s) and (%3\$s).'**

- 이 메시지는 CUBRID에서 플래시백(Flashback) 기능을 실행할 때, 사용자가 지정한 복구 시점(시간)이 시스템에서 허용하는 범위를 벗어난 경우 발생합니다. 플래시백은 로그 보관, 데이터베이스 상태, 백업 시점 등에 따라 복구 가능한 시간 범위가 제한되며, 이 범위를 벗어난 시각으로 요청하면 데이터 일관성 및 무결성 보장을 위해 작업이 거부됩니다.

**ERROR CODE: -1334, 'Can not find the class to flashback (%1\$s).'**

- 이 메시지는 CUBRID에서 플래시백(Flashback) 기능을 실행할 때, 복구 대상이 되는 테이블(테이블)이 데이터베이스에 존재하지 않거나, 시스템에서 해당 테이블을 찾을 수 없는 경우 발생합니다. 주로 테이블이 삭제되었거나, 이름이 잘못 지정되었거나, 시스템 카탈로그에 문제가 있을 때 발생할 수 있습니다.

**ERROR CODE: -1335, 'Too many transactions to flashback. Less than %1\$d transactions can be flashback.'**

- 이 메시지는 CUBRID에서 플래시백(Flashback) 기능을 실행할 때, 복구 대상이 되는 트랜잭션의 개수가 시스템에서 허용하는 최대값을 초과한 경우 발생합니다. 플래시백 작업은 데이터베이스의

일관성과 성능을 위해 한 번에 처리할 수 있는 트랜잭션 수에 제한이 있습니다. 이 제한을 초과하면 시스템 과부하, 성능 저하, 복구 실패 등의 문제가 발생할 수 있으므로, 정책적으로 제한을 두고 있습니다.

**ERROR CODE: -1336, 'Cannot support flashback for this class (name : %1\$s , OID : %2\$d|%3\$d|%4\$d). Class schema has been changed.'**

- 이 메시지는 CUBRID에서 플래시백(Flashback) 기능을 실행할 때, 복구 대상이 되는 테이블(테이블)의 스키마(구조)가 플래시백 대상 시점 이후에 변경된 경우 발생합니다. 예를 들어, 컬럼 추가/삭제, 타입 변경, 제약조건 변경 등 스키마가 변경되면, 플래시백 시점의 데이터와 현재 구조가 일치하지 않아 데이터 복구가 불가능해집니다. 데이터 무결성과 일관성을 보장하기 위해, 스키마가 변경된 테이블에는 플래시백을 지원하지 않습니다.

**ERROR CODE: -1337, 'Log record does not exist (lsa : %1\$lld|%2\$d). The archive log volume has been deleted.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 로그 레코드(트랜잭션 복구, 플래시백 등에서 필요)가 존재하지 않거나, 해당 로그가 저장된 아카이브 로그 볼륨이 이미 삭제되어 더 이상 접근할 수 없을 때 발생합니다. 주로 오래된 로그가 자동/수동으로 삭제되었거나, 백업/복구/아카이브 정책에 따라 로그 파일이 정리된 경우, 또는 로그 파일 손상, 잘못된 LSA 접근 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -1338, 'Flashback is already running. Flashback cannot be run by more than one user at the same time.'**

- 이 메시지는 CUBRID에서 플래시백(Flashback) 기능을 실행하려고 할 때, 이미 다른 세션 또는 사용자가 플래시백 작업을 진행 중인 경우 발생합니다. 플래시백은 데이터베이스의 일관성과 무결성을 보장하기 위해 단일 사용자만이 동시에 실행할 수 있도록 제한되어 있습니다. 따라서, 한 번에 하나의 플래시백 작업만 허용되며, 중복 요청이 들어오면 해당 오류가 반환됩니다.

**ERROR CODE: -1341, 'Invalid arguments to authorization internal function.'**

- 이 메시지는 CUBRID 데이터베이스에서 class 또는 method의 owner 변경 시 원하는 정보가 없는 경우 발생합니다.

**ERROR CODE: -1342, 'AUTO\_INCREMENT does not allow change of owner.'**

- 이 메시지는 CUBRID 데이터베이스에서 auto increment의 serial 객체의 owner를 변경 시도할 때 발생합니다.

**ERROR CODE: -1345, 'DBA, members of DBA group, and owner can perform CREATE TRIGGER.'**

- 이 메시지는 CUBRID에서 트리거(CREATE TRIGGER)를 생성하려고 할 때, 해당 명령을 실행하는 사용자가 DBA, DBA 그룹의 멤버, 또는 트리거 대상 객체의 소유자가 아닐 경우 발생합니다. 즉, 트리거 생성 권한이 없는 사용자가 CREATE TRIGGER를 시도할 때 발생하는 권한(Privilege) 관련 오류입니다.



**ERROR CODE: -1350, 'Invalid synonym value.'**

- 이 메시지는 CUBRID에서 umloadddb 명령어 수행 시 잘 못된 정보를 가지고 있는 시노님 (SYNONYM) 객체에서 발생합니다.

**ERROR CODE: -1356, 'There is no hint in the DBLink query, or the statement type is incorrect. %1\$s'**

- 이 메시지는 CUBRID에서 DBLink 질의는 내부적으로 statement type을 hint로 추가해서 dblink gateway로 전달해서 어떤 종류의 질의를 판단하는데, 알수 없는 statement type이 전달되어서 발생합니다.

**ERROR CODE: -1357, 'Converting SQL string to wide string failed.'**

- 이 메시지는 CUBRID가 DBLink Gateway에서 외부 데이터베이스와 통신할 때, SQL 쿼리 문자열을 UTF8에서 유니코드로 변경할 수 없을 때, 주로 문자 인코딩이 맞지 않거나, 입력 문자열에 유효하지 않은(깨진) 문자가 포함되어 있을 때 발생합니다.

**ERROR CODE: -1358, 'Number of rows affected is unknown.'**

- 이 메시지는 CUBRID가 DML(INSERT, UPDATE, DELETE)을 실행 시 실제로 영향을 받은 행의 개수를 정상적으로 파악하지 못했을 때 발생합니다. DBLink 질의에서 데이터베이스 간의 연동 작업하면서 외부 시스템이 영향을 받은 행의 수를 반환하지 않거나, 반환값이 명확하지 않을 때 발생할 수 있습니다.

**ERROR CODE: -1362, 'Locale '%1\$s' is unavailable.'**

- 이 메시지는 CUBRID가 문자열을 날짜/시간로 변경시 서버에 로딩되지 않은 로케일 정보를 사용해 발생합니다,

**데이터베이스 파일 오류****ERROR CODE: -17, 'Internal error: fetching deallocated pageid %1\$d of volume "%2\$s"'**

- 이 메시지는 CUBRID 시스템이 이미 할당 해제된(deallocated) 페이지 ID를 가져오려고 시도했을 때 발생하는 내부 오류입니다, 이 메시지는 페이지 할당/해제 관리, 메모리 관리, 또는 페이지 접근 로직에 문제가 발생했음을 의미하며, 데이터베이스의 일관성 손상 및 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -19, 'Internal error: pageptr = %1\$p of page %2\$d of volume "%3\$s" is not fixed.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지에 대한 페이지 포인터가 메모리에서 "고정(fixed)"되지 않은 상태에서 접근하려고 시도했을 때 발생하는 내부 오류입니다, 이 메시지는 페이지 버퍼 관리, 메모리 보호, 또는 페이지 접근 로직에 문제가 발생했음을 의미하며, 데이터베이스의 일관성 손상 및 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -35, 'Internal error: Unknown volume identifier %1\$d.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨 ID(Volume Identifier)를 가진 볼륨을 참조하려고 시도했지만, 해당 볼륨 ID에 해당하는 볼륨을 찾을 수 없거나 유효하지 않은 볼륨으로 인식했을 때 발생하는 내부 오류입니다, 주로 임시 볼륨(temp temp volume)을 삭제 시 발생할 수 있습니다.

**ERROR CODE: -38, 'Internal error: Unknown file VFID %1\$d|%2\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 loaddb 명령어 수행 시 오브젝트파일(object file)을 찾을 수 없거나 파일 권한 문제등으로 파일을 열지 못하는 경우 발생합니다.

**ERROR CODE: -43, 'Internal error: The page %1\$d of volume "%2\$s" may be corrupted. %3\$d records was found when %4\$d was expected.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 슬롯 페이지를 검사하는 동안, 페이지 내에서 발견된 실제 레코드의 수가 페이지 헤더에 기록된 예상 레코드의 수와 일치하지 않음을 감지했을 때 발생하는 내부 오류입니다, 이 불일치는 슬롯 페이지의 내부 구조가 손상되었음을 강력히 시사하며, 데이터베이스의 일관성이 깨졌음을 나타냅니다, 이 메시지는 주로 페이지의 슬롯(slot) 정리 작업 수행 시 발생합니다.

**ERROR CODE: -44, 'No space available in slotted page %1\$d of volume "%2\$s".'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 슬롯 페이지에 새로운 데이터를 삽입하거나 기존 데이터를 확장하려고 시도했지만, 해당 페이지에 더 이상 사용 가능한 공간이 없음을 나타내는 오류입니다, 주로 페이지가 가득 찼을 때 발생하며, 새로운 레코드 삽입, 기존 레코드 업데이트(크기 증가), 또는 인덱스 엔트리 추가와 같은 작업에 실패할 수 있습니다.

**ERROR CODE: -45, 'Slot %1\$d on page %2\$d of volume "%3\$s" is allocated to an anchored record. A new record cannot be inserted here.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지에 있는 슬롯에 새로운 레코드를 삽입하려고 시도했지만, 해당 슬롯이 이미 '고정된 레코드(anchored record)'에 할당되어 있어 삽입할 수 없음을 나타내는 내부 오류입니다, 데이터베이스의 내부 구조(특히 슬롯 페이지 구조)에 문제가 있거나, 레코드 삽입 로직에 오류가 있음을 나타내며, 데이터 삽입 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -46, 'Internal error: slot %1\$d on page %2\$d of volume "%3\$s" is not allocated.'**

- 이 메시지는 CUBRID 시스템이 특정 볼륨의 특정 페이지에 있는 슬롯에 접근하려고 시도했지만, 해당 슬롯이 할당되지 않은 상태임을 감지했을 때 발생하는 내부 오류입니다, 데이터베이스의 내부 구조(특히 슬롯 페이지 구조)에 문제가 있음을 나타내며, 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다, 이는 주로 데이터베이스 페이지에서 슬롯 ID로 접근시 해당 슬롯이 없는 경우 발생합니다.

**ERROR CODE: -47, 'Unable to create a heap file in volume "%1\$s"'**

- 이 메시지는 CUBRID 시스템이 지정된 볼륨에 새로운 힙 파일(heap file)을 생성하려고 시도했지만 실패했음을 나타내는 오류입니다, 주로 디스크 공간 부족, 파일 시스템 권한 문제, 볼륨 손상, 또는 CUBRID 내부의 파일 시스템 관리 문제로 인해 발생할 수 있습니다, 이는 주로 새로운 테이블 생성, 인덱스 생성, 또는 기존 테이블에 데이터 추가와 같은 작업에 직접적인 영향을 미칩니다.

**ERROR CODE: -48, 'Accessing deleted object %1\$d|%2\$d|%3\$d.'**

- 이 메시지는 CUBRID 시스템이 이미 삭제되었거나 유효하지 않은 객체에 접근하려고 시도했을 때 발생하는 내부 오류입니다, 객체에 대한 참조가 객체 자체의 삭제 이후에도 남아있어, 시스템이 유효하지 않은 객체에 접근하려 할 때 발생할 수 있는 내부적인 불일치를 나타냅니다, 결과적으로 데이터 접근 실패, 쿼리 오류, 또는 시스템 불안정으로 이어질 수 있습니다.

**ERROR CODE: -49, 'Internal error: class of object %1\$d|%2\$d|%3\$d is unknown.'**

- 이 메시지는 CUBRID 시스템이 특정 객체(OID로 식별되는)의 테이블 정보를 찾을 수 없을 때 발생하는 내부 오류입니다, 객체와 테이블 간의 연결이 손상되었거나, 테이블 메타데이터가 유효하지 않거나, 데이터베이스의 내부 일관성이 깨졌음을 나타냅니다, 결과적으로 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -52, 'Internal error: object overflow address %1\$d|%2\$d|%3\$d may be corrupted.'**

- 이 메시지는 CUBRID 시스템이 객체의 오버플로우 주소가 손상되었을 가능성을 감지했을 때 발생하는 내부 오류입니다, 데이터베이스의 내부 구조 관리에 문제가 있음을 나타내며, 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -53, 'Fetching object %1\$d|%2\$d|%3\$d when only its OID has been assigned.'**

- 이 메시지는 CUBRID 시스템이 특정 객체의 데이터를 가져오려고 시도했지만, 해당 객체에는 OID(Object Identifier)만 할당되어 있고 실제 데이터가 저장되지 않은 상태임을 나타내는 오류입니다, 객체 생성과 데이터 저장 사이의 타이밍 문제나, 트랜잭션 처리 과정에서 발생하는 일관성 문제를 나타냅니다.

**ERROR CODE: -55, 'Internal error: A page cycle reference was detected on page %1\$d|%2\$d of heap file %3\$d|%4\$d|%5\$d.'**

- 이 메시지는 CUBRID 시스템이 힙 파일(heap file)에서 페이지 간의 순환 참조를 감지했을 때 발생하는 내부 오류입니다, 데이터베이스의 내부 구조 관리에 문제가 있음을 나타내며, 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -56, 'Internal error: unknown extendible hashing file (Volid: %1\$d Fileid: %2\$d) page (Volid: %3\$d Pageid: %4\$d) was specified.'**

- 이 메시지는 CUBRID 시스템이 확장 가능 해싱(extendible hashing) 구조에서 알 수 없거나 유효하지 않은 파일과 페이지를 참조하려고 할 때 발생하는 내부 오류입니다, 데이터베이스의 내부 구조

관리에 문제가 있음을 나타내며, 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 수 있습니다.

**ERROR CODE: -57, 'Key does not exist in the extendible hashing structure.'**

- 이 메시지는 CUBRID 시스템이 확장 가능 해싱(extendible hashing) 구조에서 특정 키를 찾으려고 시도했지만, 해당 키가 구조 내에 존재하지 않음을 나타내는 오류입니다, 주로 존재하지 않는 키로 데이터를 조회하거나, 키가 잘못되었거나, 해싱 구조 자체에 문제가 발생했을 때 발생할 수 있습니다.

**ERROR CODE: -59, 'Internal error: the specified key type %1\$d is not valid for the extendible hashing structure.'**

- 이 메시지는 CUBRID 시스템이 확장 가능 해싱(extendible hashing) 구조에서 사용하려는 키 타입이 해당 구조에서 지원되지 않는 유효하지 않은 타입임을 감지했을 때 발생하는 내부 오류입니다, 이는 데이터베이스의 내부 구조 관리에 문제가 있음을 나타내며, 쿼리 실행 실패나 데이터 접근 오류로 이어질 수 있습니다.

**ERROR CODE: -60, 'Internal error: the extendible hashing structure has been corrupted.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스의 확장 가능 해싱(extendible hashing) 구조에 손상이 발생했음을 감지했을 때 발생하는 내부 오류입니다, 이 구조의 손상은 데이터베이스의 무결성에 치명적인 영향을 미치며, 데이터 접근 실패, 쿼리 오류, 또는 데이터 손실로 이어질 가능성도 있습니다.

**ERROR CODE: -61, 'Internal error: the directory root page of the extendible hashing structure (Valid: %1\$d Fileid: %2\$d Pageid: %3\$d) has been corrupted.'**

- 이 메시지는 CUBRID 시스템에서 확장 가능 해싱(Extendible Hashing)의 최상위 디렉토리 페이지 무결성 검사에서 손상이 감지됨을 의미합니다, 인덱스/해시 디렉토리의 시작점이 깨졌으므로 해당 구조 전체 접근이 불가능하거나 결과가 신뢰될 수 없습니다.

**ERROR CODE: -63, 'Internal error: a temporary page was corrupted during sorting.'**

- 이 메시지는 CUBRID 시스템이 데이터 정렬 작업을 수행하는 동안 사용되는 임시 페이지(temporary)가 손상되었음을 감지했을 때 발생하는 내부 오류입니다, CUBRID는 대량의 데이터를 정렬할 때 메모리 부족 시 디스크에 임시 파일을 생성하여 사용하는데, 이 임시 파일의 페이지가 읽기/쓰기 과정에서 손상된 것입니다, 이는 데이터베이스의 일관성과 정렬 작업의 정확성에 영향을 미칠 수 있으며, 쿼리 실패로 이어질 수 있습니다.

**ERROR CODE: -66, 'Internal error: unknown force operation %1\$d for object %2\$d|%3\$d|%4\$d.'**

- 이 메시지는 CUBRID 시스템이 특정 객체에 대해 알 수 없거나 유효하지 않은 “강제 작업”을 수행하려고 시도했을 때 발생하는 내부 오류입니다, CUBRID 엔진의 내부 로직에 문제가 있거나, 데이터베이스의 내부 일관성이 심각하게 손상되었을 때 발생할 수 있습니다.

**ERROR CODE: -67, 'Internal error: a heap file has not been allocated to store object %1\$d|%2\$d|%3\$d.'**

- 이 메시지는 CUBRID 시스템이 특정 객체(OID)를 저장하기 위한 힙 파일이 할당되지 않았을 때 발생하는 내부 오류입니다, 이는 데이터베이스의 내부 구조 관리에 문제가 있음을 의미하며, 객체 저장 및 접근에 치명적인 영향을 미칠 수 있습니다.

**ERROR CODE: -68, 'Internal error: different classnames for class with oid = %1\$d|%2\$d|%3\$d were found. Found classnames are "%4\$s", "%5\$s" using classname hash table and heap, respectively.'**

- 이 메시지는 CUBRID 시스템이 특정 객체(OID)를 가진 테이블에 대해 두 가지 다른 테이블 이름을 발견했을 때 발생하는 내부 오류입니다, 이 불일치는 데이터베이스의 내부 메타데이터 구조가 심각하게 손상되었음을 의미하며, 데이터베이스의 일관성과 무결성에 치명적인 문제가 발생했음을 나타냅니다.

**ERROR CODE: -69, 'Internal error: different class object identifiers were found for class with name "%1\$s". Found OIDs are %2\$d|%3\$d|%4\$d, %5\$d|%6\$d|%7\$d using classname hash table and heap, respectively.'**

- 이 메시지는 CUBRID 시스템이 특정 테이블에 대해 두 가지 다른 객체 식별자(OID)를 발견했을 때 발생하는 내부 오류입니다, 이 불일치는 데이터베이스의 내부 메타데이터 구조가 심각하게 손상되었음을 의미하며, 데이터베이스의 일관성과 무결성에 치명적인 문제가 발생했음을 나타냅니다.

**ERROR CODE: -70, 'Internal error: Class with name "%1\$s" and oid = %2\$d|%3\$d|%4\$d does not exist in classname hash table.'**

- 이 메시지는 CUBRID 시스템이 특정 테이블(OID)를 테이블명 해시 테이블에서 찾으려고 시도했지만 실패했을 때 발생하는 내부 오류입니다, 테이블명 해시 테이블은 CUBRID가 테이블 이름을 빠르게 조회하기 위해 사용하는 내부 데이터 구조입니다. 이 메시지는 데이터베이스의 메타데이터 관리 시스템에 불일치 또는 손상이 발생했음을 나타냅니다, 이는 데이터베이스의 내부 구조에 문제가 있음을 의미합니다.

**ERROR CODE: -71, 'Internal error: Class with name "%1\$s" and oid = %2\$d|%3\$d|%4\$d does not exist in its heap.'**

- 이 메시지는 CUBRID 시스템이 특정 테이블(OID정보)를 데이터베이스의 힙(heap) 영역에서 찾으려고 시도했지만 실패했을 때 발생하는 내부 오류입니다, 힙은 실제 데이터 객체들이 저장되는 공간을 의미합니다. 이 메시지는 데이터베이스의 메타데이터(테이블 정의)와 실제 저장된 데이터 간에 불일치 또는 손상이 발생했음을 나타냅니다, 이는 데이터베이스의 무결성에 문제가 있음을 의미합니다.

**ERROR CODE: -80, 'Insufficient space in operating system device when writing logical log page %1\$lld (physical page %2\$lld) of "%3\$s". Could not write more than %4\$d bytes.'**

- 이 메시지는 CUBRID 시스템이 논리적 로그 페이지를 물리적 페이지에 쓰려고 할 때, 운영체제 장치에 충분한 공간이 없어서 쓰기 작업이 실패했음을 나타내는 오류입니다, 주로 디스크 공간 부족, 파일 시스템 제한, 또는 I/O 오류로 인해 발생합니다.

**ERROR CODE: -81, 'Internal error: logical log page %1\$lld may be corrupted.'**

- 이 메시지는 CUBRID 시스템이 데이터베이스의 트랜잭션 로그 파일 내에 있는 특정 논리적 로그 페이지가 손상되었을 가능성을 감지했을 때 발생하는 내부 오류입니다, 데이터베이스의 안정성과 데이터 손실 위험을 나타내는 매우 중요한 경고입니다.

**ERROR CODE: -96, 'Media recovery may be needed on volume "%1\$s".'**

- 이 메시지는 CUBRID 시스템이 데이터베이스 볼륨의 일관성 또는 무결성 문제를 감지하여 미디어 복구(Media Recovery) 작업이 필요할 수 있음을 알리는 오류입니다, 주로 비정상적인 종료, 디스크 손상, 또는 로그 파일의 불일치 등으로 인해 데이터베이스의 특정 볼륨이 손상되었을 가능성이 있을 때 발생합니다.

**ERROR CODE: -97, 'Internal error: unable to find log page %1\$lld in log archives.'**

- 이 메시지는 CUBRID 시스템이 특정 로그 페이지를 로그 아카이브에서 찾으려고 시도했지만 실패했을 때 발생하는 내부 오류입니다, 이는 데이터베이스의 트랜잭션 로그를 처리하는 과정에서 필요한 로그 페이지가 아카이브 파일에 존재하지 않거나, 아카이브 파일이 손상되었거나, 아카이브 파일에 접근할 수 없을 때 발생합니다.

**ERROR CODE: -98, 'Unable to create archive log "%1\$s" to archive pages from %2\$lld to %3\$lld.'**

- 이 메시지는 CUBRID 시스템이 로그 아카이브 작업을 수행할 때, 아카이브 로그 파일을 생성하는데 실패했을 때 발생하는 오류입니다, 데이터베이스의 일관성 및 복구 가능성에 영향을 미칠 수 있는 중요한 오류입니다.

**ERROR CODE: -296, 'Invalid property list encountered.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체의 프로퍼티(속성) 리스트가 유효하지 않거나 손상되었을 때 나타나는 오류입니다, 객체 표현(object representation) 또는 스키마 관리 시스템에서 프로퍼티 리스트의 유효성을 검증하는 과정에서 발생합니다, CUBRID 데이터베이스 내부의 메타데이터나 객체 정의가 잘못되었거나 손상되었을 가능성을 시사합니다.

**ERROR CODE: -315, 'Illegal metaclass definition encountered.'**

- 이 메시지는 CUBRID 데이터베이스에서 메타테이블 정의가 잘못되었을 때 나타나는 오류입니다, 메타테이블은 CUBRID의 시스템 카탈로그 테이블들을 정의하는 구조체를 의미합니다, 이 메시지는 메타테이블의 속성 정의가 올바르지 않을 때 발생합니다.

**ERROR CODE: -316, 'Transformer size calculation mismatch, expected %1\$d calculated %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체 변환 시 예상 크기와 실제 계산된 크기가 일치하지 않을 때 나타나는 오류입니다, 객체를 디스크에 저장하기 전에 크기를 계산하고, 실제 저장 후 크기를 검증합니다, 크기 불일치는 내부 로직 오류나 데이터 손상을 나타낼 수 있습니다.

**ERROR CODE: -317, 'Invalid disk representation encountered for class "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블의 디스크 표현(representation)이 유효하지 않을 때 나타나는 오류입니다, 디스크 표현은 객체가 디스크에 저장되는 방식을 정의하는 구조체로 테이블에서 특정 representation ID에 해당하는 표현을 찾을 수 없을 때 발생합니다.

**ERROR CODE: -318, 'Out of sync during object loading. Database is likely corrupted or out of date.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체를 로딩하는 과정에서 동기화(sync) 문제가 발생했을 때 나타나는 오류입니다, 디스크에서 읽은 데이터의 크기와 예상 크기가 다를 때 발생합니다, 일반적으로 데이터베이스 파일 손상, 버전 호환성 문제, 또는 메모리 손상으로 인해 발생할 수 있습니다.

**ERROR CODE: -406, 'Invalid B+tree index identifier: (vfid = (%1\$d, %2\$d), rt\_pgid: %3\$d).'**

- 이 메시지는 CUBRID 데이터베이스에서 B-tree 인덱스 식별자(BTID)가 유효하지 않을 때 나타나는 오류입니다, B-tree 저장소 관리에서 인덱스 식별자의 유효성 검사 과정에서 주로 B-tree 인덱스 식별자의 vfid(볼륨 파일 ID)나 pageid가 잘못되었을 때 발생합니다.

**ERROR CODE: -407, 'Unknown key %1\$s referenced in B+tree index {vfid: (%2\$d, %3\$d), rt\_pgid: %4\$d, key\_type: %5\$s}.'**

- 이 메시지는 B+tree 인덱스 구조에서 존재하지 않거나 손상된 키를 참조하려 할 때 발생하는 시스템 오류입니다, 데이터베이스의 인덱스 구조 무결성에 문제가 있음을 나타내는 중요한 오류 메시지로, 인덱스의 일관성이 깨졌거나 데이터 손상이 발생했을 수 있습니다.

**ERROR CODE: -412, 'Invalid range search specification.'**

- 이 메시지는 CUBRID 데이터베이스에서 B-tree 인덱스를 사용한 범위 검색(range search)을 수행할 때, 검색 범위 명세가 유효하지 않을 때 나타나는 오류입니다, B-tree 저장소 관리에서 범위 검색 유효성 검사 과정에 지원되지 않는 범위 검색 타입이나 잘못된 범위 검색 조건이 사용되었을 때입니다.

**ERROR CODE: -415, 'Invalid class identifier: %1\$d|%2\$d|%3\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 내부적으로 사용되는 테이블 식별자(OID, Object Identifier)가 유효하지 않거나, 존재하지 않는 테이블을 참조할 때 발생합니다, 주로 시스템 카탈로그, 쿼리 실행, 객체 접근 등에서 잘못된 OID가 전달되었거나, 테이블이 삭제/변경되어 더 이상 유효하지 않은 경우, 또는 내부 데이터 손상 등으로 인해 발생할 수 있습니다.



**ERROR CODE: -416, 'Unknown representation identifier: %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체(테이블, 레코드 등)의 내부 표현식(Representation)을 식별하는 ID(REPRID)가 유효하지 않거나, 시스템에 등록되어 있지 않은 값을 참조할 때 발생합니다. 주로 시스템 카탈로그, 객체 접근, 쿼리 실행 등에서 잘못된 식별자가 전달되었거나, 테이블 구조 변경, 데이터 손상, 복구/이관 과정에서 식별자가 유실/불일치한 경우에 발생할 수 있습니다.

**ERROR CODE: -417, 'Invalid representation identifier: %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 객체(테이블, 레코드 등)의 내부 표현식(Representation)을 식별하는 ID(REPRID)가 유효하지 않거나, 허용되지 않는 값이 전달될 때 발생합니다. 주로 시스템 카탈로그, 객체 접근, 쿼리 실행 등에서 잘못된 식별자가 사용되었거나, 테이블 구조 변경, 데이터 손상, 복구/이관 과정에서 식별자가 불일치하거나 손상된 경우에 발생할 수 있습니다.

**ERROR CODE: -421, 'Representations Directory Missing in Catalog for Class: %1\$d|%2\$d|%3\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 테이블의 representation 디렉터리가 시스템 카탈로그에서 누락되었을 때 나타나는 오류입니다. 시스템 카탈로그 손상으로 필요한 정보를 찾을 수 없을 때 이 오류가 발생할 가능성이 매우 높습니다.

**ERROR CODE: -422, 'Representation Information Record Missing in Catalog for Class Repr\_Id: %1\$d|%2\$d|%3\$d %4\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 테이블의 representation 정보 레코드가 시스템 카탈로그에서 누락되었을 때 나타나는 오류입니다. 시스템 카탈로그 손상으로 필요한 정보를 찾을 수 없을 때 이 오류가 발생할 가능성이 매우 높습니다.

**ERROR CODE: -475, 'No query specification with index %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 인덱스 번호에 해당하는 쿼리 명세(Query Specification)를 찾을 수 없을 때 나타나는 오류입니다. 쿼리 명세는 가상 테이블나 뷰에서 사용되는 SQL 쿼리 정의를 의미하며, 각 명세는 인덱스 번호로 식별됩니다. 이 메시지는 존재하지 않는 인덱스 번호로 쿼리 명세를 조회하거나 삭제하려고 할 때 발생합니다.

**ERROR CODE: -540, '%1\$s.'**

- 이 메시지는 CUBRID 데이터베이스에서 긴급 오류가 발생했을 때 나타나는 일반적인 오류 메시지입니다. 주로 내부 시스템 오류나 예상치 못한 상황이 발생했을 때 사용되는 범용 오류 메시지입니다. 이 오류의 구체적인 내용은 트리거 컴파일 오류, B+Tree 구조 오류, 인덱스 구조의 손상이나 불일치 발생될 수 있습니다.



**ERROR CODE: -544, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). B+tree key %6\$s entry for object OID: %7\$d|%8\$d|%9\$d was not found on B+tree: %10\$d|%11\$d|%12\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 B+tree 인덱스에서 특정 키 엔트리를 찾을 수 없을 때 나타나는 내부 오류입니다, 주로 인덱스 일관성 검사 과정에서 힙에 존재하는 객체의 키가 B+tree 인덱스에 존재하지 않을 때 발생합니다, 인덱스와 힙 간의 데이터 불일치를 나타내는 내부 오류입니다.

**ERROR CODE: -545, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). Key and OID: %6\$d|%7\$d|%8\$d entry on B+tree: %9\$d|%10\$d|%11\$d is incorrect. The object does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 B+tree 인덱스의 키와 OID 엔트리가 올바르지 않을 때 나타나는 내부 오류입니다, 주로 인덱스 스캔 과정에서 B+tree에 존재하는 OID가 실제로는 힙에 존재하지 않을 때 발생합니다, 인덱스와 힙 간의 데이터 불일치를 나타내는 내부 오류입니다.

**ERROR CODE: -546, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). Expecting %6\$d OID entry values, but %7\$d were found on B+tree: %8\$d|%9\$d|%10\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 B+tree 인덱스의 OID 엔트리 개수가 예상과 다를 때 나타나는 내부 오류입니다, 주로 인덱스 일관성 검사 과정에서 힙(heap)의 OID 개수와 B+tree의 OID 개수가 일치하지 않을 때 발생합니다, 인덱스와 힙 간의 데이터 불일치를 나타내는 내부 오류입니다.

**ERROR CODE: -551, 'Unable to locate volume information path file "%1\$s"... Continue reading from internal tables'**

- 이 메시지는 CUBRID 데이터베이스에서 볼륨 정보 파일(DB\_vinf)을 찾을 수 없을 때 나타나는 오류입니다, 주로 데이터베이스 부팅이나 로그 페이지 버퍼 스캔 과정에서 볼륨 정보 파일에 접근할 수 없을 때 발생합니다.

**ERROR CODE: -585, 'Unknown heap %1\$s|%2\$d|%3\$d'**

- 이 메시지는 CUBRID 데이터베이스에서 존재하지 않거나 알 수 없는 heap 파일에 접근하려고 할 때 발생하는 오류입니다, heap 파일은 CUBRID에서 데이터를 저장하는 기본 단위이며, 각 heap 파일은 고유한 식별자(HFID)를 가집니다.

**ERROR CODE: -587, 'Entries of permanent volumes are unsorted in your "%1\$s" volinfo file. Entry %2\$d: %3\$d %4\$s is out of sequence.'**

- 이 메시지는 CUBRID 데이터베이스의 DBname\_vinf 파일에서 영구 볼륨(permanent volume) 엔트리들이 올바른 순서로 정렬되어 있지 않음을 나타냅니다, DBname\_vinf 파일은 데이터베이스의 볼륨 정보를 저장하는 파일경로, 파일 정보등 볼륨 ID 순서대로 정렬되어 있어야 합니다.

**ERROR CODE: -596, 'The %1\$d pages of total temporary space allowed have been exceeded.'**

- 이 메시지는 CUBRID 데이터베이스에서 백엔드(Backend)가 사용할 수 있도록 허용된 총 임시 공간(temporary space)의 페이지 수가 초과되었음을 나타냅니다, CUBRID는 복잡한 쿼리 처리, 정렬, 조

인 등의 작업을 수행할 때 임시 데이터를 저장하기 위해 임시 공간을 사용합니다, 이 메시지는 cubrid.conf 파일에 `temp\_file\_max\_size\_in\_pages`와 같은 설정 파라미터에 의해 제한된 임시 공간을 초과했을 때 발생합니다.

**ERROR CODE: -597, 'Number of pages for heap file %1\$d|%2\$d|%3\$d is inconsistent. n%4\$d and %5\$d were found according to heap chain and file table, respectively.'**

- 이 메시지는 CUBRID 데이터베이스에서 checkdb 명령어 수행 시 힙 파일(heap file)에서 페이지(page) 개수의 불일치가 발견되었음을 나타내는 내부 오류입니다, 힙 파일은 실제 데이터 레코드가 저장되는 공간이며, 힙 체인(heap chain)은 힙 파일 내 페이지들의 논리적 연결을, 파일 테이블(file table)은 파일 시스템 수준에서 관리되는 페이지 정보를 의미합니다.

**ERROR CODE: -603, 'Internal Error: Sector/page table of file VFID %1\$d|%2\$d seems corrupted.'**

- 이 메시지는 CUBRID 데이터베이스에서 파일 시스템의 내부 구조(섹터/페이지 테이블)에 손상이 감지되었을 때 발생합니다, 데이터베이스 파일의 무결성이 깨져서 정상적인 데이터 접근 및 관리가 불가능할 수 있음을 나타내는 치명적인 내부 오류입니다.

**ERROR CODE: -610, 'Your database is likely to be corrupted since logging was turned off when your database crashed.'**

- 이 메시지는 CUBRID 데이터베이스에서 로그 복구 과정에서 발생합니다, 데이터베이스가 크래시된 시점에 로깅이 비활성화되어 있어서 데이터베이스가 손상되었을 가능성이 높다는 의미입니다.

**ERROR CODE: -614, 'Number of active log archives has been exceeded the max desired number of %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 활성 로그 아카이브의 개수가 설정된 최대 허용 개수를 초과했을 때 나타납니다, 로그 아카이브 관리 시스템에서 자동으로 오래된 로그 아카이브를 삭제하는 과정에서 발생하는 알림입니다.

**ERROR CODE: -625, 'Internal Error. Trying to update the wrong instance object %1\$d|%2\$d|%3\$d attribute information template with instance object %4\$d|%5\$d|%6\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 내부적으로 발생하는 오류입니다, 잘못된 레코드 객체의 속성 정보 OID 불일치로 인해 발생하는 내부 오류입니다, OID(Object Identifier)는 CUBRID에서 데이터베이스 내의 각 객체를 고유하게 식별하는 8바이트 크기의 식별자입니다.

**ERROR CODE: -626, 'Internal Error. %1\$d requested attributes were not found.'**

- 이 메시지는 CUBRID 데이터베이스에서 내부적으로 발생하는 오류입니다, 요청된 속성(attribute) 중 일부를 찾을 수 없을 때 나타납니다, 테이블의 속성 정보를 캐시하거나 검색하는 과정에서 발생합니다.

**ERROR CODE: -638, 'Warning: Flushing a non-updatable log archive pageid = %1\$d'**

- 이 메시지는 CUBRID 데이터베이스에서 업데이트할 수 없는 로그 아카이브 페이지를 플러시하려고 할 때 발생하는 경고입니다, 로그 페이지가 아카이브 범위를 벗어나거나 더 이상 업데이트할 수 없는 상태에서 플러시 작업이 시도될 때 발생합니다.

**ERROR CODE: -644, 'LOG FATAL ERROR: %1\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 로그 시스템에서 발생하는 치명적인 오류를 나타냅니다, 로그 처리 과정에서 복구할 수 없는 오류가 발생했을 때 사용됩니다, 이는 데이터베이스의 무결성과 안정성에 영향을 미칠 수 있는 오류입니다.

**ERROR CODE: -694, 'SYSTEM ERROR: Unable to load B+tree.'**

- 이 메시지는 CUBRID 데이터베이스에서 B+tree 인덱스를 로드(생성)하려고 할 때 발생하는 시스템 레벨 오류입니다, 인덱스 생성 과정에서 메모리 부족, 디스크 I/O 오류, 매개변수 검증 실패, 또는 내부 알고리즘 오류 등으로 인해 발생할 수 있습니다.

**ERROR CODE: -698, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). the number of OIDs in the unique hierarchy: %6\$d does not equal the number of OIDs: %7\$d found in the unique B+tree plus the number of NULLs: %8\$d found in the unique hierarchy for B+tree: %9\$d|%10\$d|%11\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 내부 일관성 오류를 나타냅니다, 특정 테이블의 인덱스에 대한 고유 제약조건(UNIQUE constraint)이 적용되는 계층구조(hierarchy) 내의 총 OID 개수가 B+tree에서 찾은 OID 개수와 고유 계층구조에서 찾은 NULL 개수의 합과 일치하지 않을 때 발생합니다, 이는 데이터베이스의 메타데이터(스키마 정보)와 실제 B+tree 인덱스 구조 간의 불일치를 의미하며, 데이터 무결성이 손상되었거나 인덱스 구조에 문제가 있을 가능성이 있습니다.

**ERROR CODE: -699, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). the number of OIDs in the unique hierarchy: %6\$d does not equal the number of OIDs: %7\$d found in the unique B+tree root statistics for B+tree: %8\$d|%9\$d|%10\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 내부 일관성 오류를 나타냅니다, 특정 테이블의 인덱스에 대한 고유 제약조건(UNIQUE constraint)이 적용되는 계층구조(hierarchy) 내의 OID(Object Identifier) 개수가 해당 인덱스의 B+tree 루트 통계(root statistics)에 기록된 OID 개수와 일치하지 않을 때 발생합니다, 이는 데이터베이스의 메타데이터(스키마 정보)와 실제 B+tree 인덱스 구조 간의 불일치를 의미하며, 데이터 무결성이 손상되었거나 인덱스 구조에 문제가 있을 가능성이 있습니다.

**ERROR CODE: -700, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). the number of OIDs: %6\$d does not equal the number of NULLs: %7\$d plus the number of keys: %8\$d in the unique B+tree: %9\$d|%10\$d|%11\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 내부 일관성 오류를 나타냅니다, 특정 테이블의 인덱스에 대한 고유 B+tree에서 총 OID 개수가 NULL 값 개수와 키 개수의 합과 일치하지 않을 때 발생합니다,

이는 B+tree 인덱스의 내부 통계가 일관되지 않음을 의미하며, 데이터 무결성이 손상되었거나 인덱스 구조에 문제가 있을 가능성이 있습니다.

**ERROR CODE: -702, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). the number of NULLs in the unique hierarchy: %6\$d does not equal the number of NULLs: %7\$d found in the unique B+tree root statistics for B+tree: %8\$d|%9\$d|%10\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 내부 일관성 오류를 나타냅니다, 특정 테이블의 인덱스에 대한 고유 제약조건(UNIQUE constraint)이 적용되는 계층구조(hierarchy) 내의 NULL 값 개수가 해당 인덱스의 B+tree 루트 통계(root statistics)에 기록된 NULL 값 개수와 일치하지 않을 때 발생합니다, 이는 데이터베이스의 메타데이터(스키마 정보)와 실제 B+tree 인덱스 구조 간의 불일치를 의미하며, 데이터 무결성이 손상되었거나 인덱스 통계가 잘못되었을 가능성이 있습니다.

**ERROR CODE: -703, 'Internal error: INDEX %1\$s ON CLASS %2\$s (CLASS\_OID: %3\$d|%4\$d|%5\$d). OID: %6\$d|%7\$d|%8\$d found in unique B+tree: %9\$d|%10\$d|%11\$d does not belong to one of the classes for the unique constraint.'**

- 이 메시지는 CUBRID 데이터베이스의 내부 일관성 오류를 나타냅니다, 특정 테이블의 인덱스에 대한 고유 B+tree에서 발견된 객체 OID가 해당 고유 제약조건이 적용되는 테이블 계층구조(UNIQUE hierarchy)에 속하지 않을 때 발생합니다, 이는 데이터베이스의 메타데이터(스키마)와 실제 데이터 저장 구조(B+tree) 간의 불일치를 의미하며, 데이터 무결성이 손상되었을 가능성이 있습니다.

**ERROR CODE: -714, 'Query failed due to insufficient temporary file space.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 실행 중 임시 파일(temporary file) 공간이 부족하여 쿼리 처리가 실패했을 때 나타나는 오류입니다, CUBRID는 복잡한 쿼리(예: 정렬, 조인, 그룹화 등)를 처리할 때 중간 결과를 저장하기 위해 임시 파일을 사용합니다.

**ERROR CODE: -725, 'Internal error: expected temporary oid and encountered permanent oid.'**

- 이 메시지는 CUBRID 데이터베이스에서 collection data type 처리 시 임시 OID를 기대했지만 영구 OID를 만났을 때 나타나는 내부 오류입니다, OID는 CUBRID에서 데이터베이스 객체를 고유하게 식별하는 식별자입니다, 이 메시지는 객체의 상태와 예상된 OID 타입이 일치하지 않을 때 발생합니다.

**ERROR CODE: -728, 'Query failed because temporary file vfid is invalid for transaction %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 실행 중 임시 파일(temporary file)의 VFID(Volume File ID)가 특정 트랜잭션에 대해 유효하지 않을 때 나타나는 오류입니다, CUBRID는 복잡한 쿼리나 대량의 데이터를 처리할 때 임시 파일을 사용하여 중간 결과를 저장합니다, VFID는 CUBRID에서 파일을 고유하게 식별하는 식별자로, 볼륨 ID와 파일 ID로 구성됩니다, 이 메시지는 임시 파일의 VFID가 손상되었거나, 해당 트랜잭션과 연결된 임시 파일이 예상치 못한 상태에 있을 때 발생합니다.

**ERROR CODE: -858, 'Volume "%1\$s" is unknown at server restart.'**

- 이 메시지는 데이터베이스 서버가 구동/종료시 불필요한 temp temp volume을 삭제하는데, 시스템 정보에 등록되지 않는 temp temp volume이 존재해서 삭제한다는 경고성 메세지입니다.

**ERROR CODE: -909, 'Missing or invalid catalog class/vclass is found.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 내부적으로 사용하는 카탈로그 테이블 또는 가상 테이블(뷰)을 찾을 수 없거나, 해당 카탈로그의 정의가 유효하지 않을 때 발생합니다, 즉, 데이터베이스의 핵심 메타데이터(테이블, 컬럼, 인덱스, 사용자 정보 등 시스템 운영에 필수적인 정보)를 관리하는 내부 객체에 문제가 발생하여 시스템이 정상적으로 동작할 수 없는 상황을 의미합니다, 이런 상황은 일반적으로 데이터베이스 손상, 비정상적인 시스템 종료, 데이터베이스 업그레이드 실패, 또는 CUBRID 내부의 메타데이터 관리 로직 오류로 인해 발생합니다, 이는 데이터베이스 시스템의 무결성과 안정성을 보호하기 위한 치명적인 보호성 오류입니다.

**ERROR CODE: -934, 'Null domain referenced.'**

- 이 메시지는 CUBRID 데이터베이스에서 데이터 타입의 도메인 정보가 NULL이거나 유효하지 정보의 영향으로 일반적으로 잘못된 데이터 타입 정의로 인해 발생합니다.

**ERROR CODE: -983, 'Instances of a reusable OID class are non-referable. This operation is not permitted on non-referable instances.'**

- 이 메시지는 CUBRID 데이터베이스에서 REUSE\_OID 옵션이 설정된 테이블의 레코드를 참조하려고 할 때 발생합니다, `REUSE\_OID`는 CUBRID의 특별한 기능으로, 삭제된 레코드의 OID를 재사용하여 데이터베이스의 공간 효율성을 높이는 기능입니다. 하지만 이 기능이 활성화된 테이블의 레코드는 외부에서 참조할 수 없도록 제한됩니다, 이는 데이터 무결성을 보호하고 OID 재사용으로 인한 참조 오류를 방지하기 위한 보호성 오류입니다. 특히 외래 키 제약조건, 객체 참조, 또는 다른 테이블에서의 참조가 시도될 때 발생할 수 있습니다.

**ERROR CODE: -1016, '%1\$s external storage error: %2\$s'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 외부 저장소(External Storage)를 사용하여 LOB(Large Object) 데이터나 외부 파일을 처리하는 과정에서 발생하는 일반적인 오류입니다, 즉, CUBRID 서버가 외부 저장소 시스템에 접근하려고 할 때, 파일 생성, 읽기, 쓰기, 삭제 등의 작업이 실패했음을 의미합니다, 이는 외부 저장소의 초기화 실패, 파일 시스템 권한 문제, 디스크 공간 부족, 네트워크 연결 문제, 또는 외부 저장소 시스템 자체의 오류로 인해 발생할 수 있습니다, 이 메시지는 CUBRID의 LOB 데이터 관리 시스템에서 외부 저장소와의 연동 과정에서 발생하는 보호성 오류입니다.

**ERROR CODE: -1017, 'Path for external storage '%1\$s' is invalid.'**

- 이 메시지는 CUBRID 데이터베이스 시스템이 외부 저장소(External Storage)의 경로를 검증하는 과정에서 해당 경로가 유효하지 않다고 판단했을 때 발생합니다, 즉, CUBRID 서버가 LOB 데이터를 저장하기 위한 외부 저장소 경로를 초기화하거나 사용하려고 할 때, 경로 형식이 올바르지 않거나, 지원되지 않는 외부 저장소 타입을 지정했거나, 경로 구성 요소가 누락되었음을 의미합니다,

이는 CUBRID의 외부 저장소 시스템에서 경로의 유효성을 검증하는 과정에서 발생하는 보호성 오류입니다.

**ERROR CODE: -1019, 'External storage is not initialized because the path is not specified in "databases.txt"'**

- 이 메시지는 CUBRID 데이터베이스의 외부 스토리지(LOB) 경로가 databases.txt 파일에 정의되지 않아 초기화에 실패했을 때 발생합니다. LOB(Large Object) 데이터를 저장하기 위한 외부 스토리지 경로 설정이 누락되었음을 나타냅니다.

**ERROR CODE: -1020, 'External file "%1\$s" was not found.'**

- 이 메시지는 CUBRID가 외부 스토리지(LOB 저장소 또는 다른 외부 파일 시스템)에서 특정 파일을 찾으려고 시도했으나 해당 파일이 존재하지 않을 때 발생합니다. 파일 경로가 잘못되었거나, 파일이 실제로 삭제되었거나, 접근 권한 문제 등으로 인해 파일을 찾을 수 없는 상황을 나타냅니다.

**ERROR CODE: -1075, 'Descending index scan aborted because of lower priority on B+tree with index identifier: (vfid = (%1\$d, %2\$d), rt\_pgid: %3\$d).'**

- 이 메시지는 CUBRID 데이터베이스에서 B+트리 인덱스의 내림차순(Descending) 스캔 작업이, 시스템 내부에서 해당 작업의 우선순위가 낮다고 판단되어 중단(abort)되었을 때 발생합니다, 주로 내림차순 스캔시 원하는 btree 페이지를 얻지 못하는 경우 발생합니다.

**ERROR CODE: -1084, 'Skip invalid page in checkpoint. (page id: %1\$d, "%2\$s", oldest\_unflush\_lsa: %3\$lld|%4\$d, previous checkpoint redo lsa: %5\$lld|%6\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 체크포인트 (checkpoint) 시 내부의 시간 정보가 맞지 않아서 발생하는 오류 메시지로, 시스템 운영 및 데이터 정합성에 영향이 없는 오류입니다.

**ERROR CODE: -1125, 'Create the overflow key file. INDEX %1\$s%2\$s ON CLASS %3\$s%4\$s. key: %5\$s%6\$s.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 B-tree 인덱스를 생성하거나 업데이트할 때, 인덱스 키의 크기가 페이지에 직접 저장할 수 있는 최대 크기를 초과하여 오버플로우 키 파일을 생성했을 때 발생하는 정보성 알림입니다, 즉, CUBRID 서버가 B-tree 인덱스 작업을 처리하는 과정에서, 페이지 특정 크기 보다 큰 키를 발견하면 자동으로 오버플로우 키 파일을 생성하여 해당 키를 별도의 오버플로우 페이지에 저장합니다.

이는 B-tree의 깊이를 제한하고 인덱스 성능을 유지하기 위한 정상적인 동작으로, 오류가 아닌 시스템의 자동 최적화 기능입니다.

**ERROR CODE: -1126, 'Create a new overflow page. INDEX %1\$s%2\$s ON CLASS %3\$s%4\$s. key: %5\$s%6\$s.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 B-tree 인덱스를 생성하거나 업데이트할 때, 인덱스 리프 노드에 저장할 수 있는 OID(Object Identifier)의 개수가 페이지 한계를 초과하여 오버플로우 페이지를 생성했을 때 발생하는 정보성 알림입니다.



즉, CUBRID 서버가 B-tree 인덱스 작업을 처리하는 과정에서, 페이지 특정 크기에 맞는 최대(OID 개수)보다 많은 OID를 하나의 키에 대해 저장해야 할 때 자동으로 오버플로우 페이지를 생성하여 추가 OID를 별도 페이지에 저장합니다. 이는 B-tree의 깊이를 제한하고 인덱스 성능을 유지하기 위한 정상적인 동작으로, 오류가 아닌 시스템의 자동 최적화 기능입니다.

**ERROR CODE: -1127, 'Delete an empty overflow page. INDEX %1\$s%2\$s ON CLASS %3\$s%4\$s. key: %5\$s%6\$s.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 B-tree 인덱스를 정리하거나 최적화할 때, 더 이상 사용되지 않는 빈 오버플로우 페이지를 삭제했을 때 발생하는 정보성 알림입니다. 즉, CUBRID 서버가 B-tree 인덱스의 유지보수 작업을 수행하는 과정에서, 특정 키에 대한 OID(Object Identifier)가 삭제되거나 변경되어 해당 오버플로우 페이지가 비어있게 되었을 때, 시스템이 자동으로 해당 페이지를 정리하고 삭제합니다. 이는 저장 공간을 효율적으로 관리하고 인덱스 구조를 최적화하기 위한 정상적인 동작으로, 오류가 아닌 시스템의 자동 정리 기능입니다.

**ERROR CODE: -1128, 'Log recovery is started.'**

- 이 메시지는 실제로는 오류가 아니라 CUBRID 데이터베이스의 로그 복구(recovery) 과정이 시작되었음을 알리는 정보성 메시지입니다.

**ERROR CODE: -1129, 'Log recovery is finished.'**

- 이 메시지는 실제로는 오류가 아니라 CUBRID 데이터베이스의 로그 복구(recovery) 과정이 성공적으로 완료되었음을 알리는 정보성 메시지입니다.

**ERROR CODE: -1146, 'volume identifier %1\$d does not exist.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 볼륨 ID를 사용하여 볼륨에 접근하려고 할 때, 해당 볼륨이 시스템에 존재하지 않음을 감지했을 때 발생합니다. 즉, *show volume header* 구문 수행시 발생할 수 있는 오류로 데이터베이스가 참조하려는 물리적 저장 공간(볼륨)을 찾을 수 없을 때 발생하는 진단(diagnostic) 오류입니다. 이런 상황은 일반적으로 볼륨 파일이 삭제되었거나, 경로가 변경되었거나, 데이터베이스 구성 파일에 잘못된 볼륨 정보가 설정되었을 때 발생할 수 있습니다.

**ERROR CODE: -1149, 'Cannot find the page %1\$d of volume %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 볼륨의 특정 페이지에 접근하려고 할 때, 해당 페이지가 존재하지 않거나 이미 해제(deallocated)된 경우 발생합니다. 즉, *show slotted page header/slots* 구문 수행시 발생할 수 있는 오류로 시스템이 요청된 페이지를 찾을 수 없을 때 발생하는 진단(diagnostic) 오류입니다. 이런 상황은 일반적으로 페이지가 삭제되었거나, 볼륨이 손상되었거나, 잘못된 페이지 참조가 있을 때 발생합니다.

**ERROR CODE: -1151, 'The page %1\$d of volume %2\$d is not a slotted page.'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 볼륨의 특정 페이지에 접근하려고 할 때, 해당 페이지가 예상되는 '슬롯 페이지(Slotted Page)' 형식이 아님을 시스템이 감지했을 때 발생합니다. 즉, *show slotted page header/slots* 구문 수행시 발생할 수 있는 오류로 데이터베이스가 특정 페이지

를 슬롯 페이지로 기대하고 접근했으나, 실제 페이지의 헤더 정보나 내부 구조가 슬롯 페이지의 유효성 검사를 통과하지 못했을 때 발생하는 진단(diagnostic) 오류입니다.

#### **ERROR CODE: -1181, 'Manual vacuum is disabled for client-server mode.'**

- 이 메시지는 CUBRID 데이터베이스가 클라이언트-서버(Client-Server) 모드로 실행 중일 때, 사용자가 수동으로 진공(vacuum) 작업을 시도할 경우 발생합니다. CUBRID는 클라이언트-서버 환경에서 데이터베이스의 일관성과 안정성을 유지하기 위해, 수동 진공(vacuum) 작업을 제한하고 내부적으로 자동 진공(auto vacuum) 메커니즘을 통해 공간 회수 및 최적화를 관리합니다. 따라서, 이 메시지는 수동 진공 명령이 현재 운영 모드에서는 허용되지 않음을 알려주는 보호성 오류입니다.

### 트랜잭션 관련 오류

#### **ERROR CODE: -72, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) has been unilaterally aborted by the system.'**

- 이 메시지는 CUBRID 시스템에서 특정 트랜잭션이 시스템에 의해 일방적으로 중단되었음을 나타내는 오류입니다. 여기서 “일방적으로 중단”이라는 것은 트랜잭션을 실행한 사용자나 애플리케이션의 의도와 관계없이 시스템이 자동으로 트랜잭션을 롤백했다는 의미입니다. 이는 주로 데드락 해결, 시스템 리소스 부족, 서버 장애, 또는 기타 시스템 레벨의 문제로 인해 발생합니다.

#### **ERROR CODE: -73, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on object %6\$d|%7\$d|%8\$d. You are waiting for user(s) %9\$s to finish.'**

- 이 메시지는 CUBRID 시스템에서 특정 트랜잭션이 특정 객체(레코드)에 대한 잠금을 획득하기 위해 기다리는 중 시간 초과가 발생했음을 나타내는 락 타임아웃(lock timeout) 오류입니다. 현재 트랜잭션이 다른 사용자가 해당 객체(레코드)에 대한 잠금을 해제하기를 기다리고 있지만, 설정된 시간 내에 잠금이 해제되지 않아 타임아웃이 발생한 상황입니다. 이는 주로 객체(레코드) 레벨의 동시성 제어 문제, 데드락 상황, 또는 장시간 실행되는 트랜잭션으로 인해 발생합니다.

#### **ERROR CODE: -74, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on class %6\$s. You are waiting for user(s) %7\$s to finish.'**

- 이 메시지는 CUBRID 시스템에서 특정 트랜잭션이 테이블에서 잠금을 획득하기 위해 기다리는 중 시간 초과가 발생했음을 나타내는 락 타임아웃(lock timeout) 오류입니다. 현재 트랜잭션이 다른 사용자가 해당 테이블에 대한 잠금을 해제하기를 기다리고 있지만, 설정된 시간 내에 잠금이 해제되지 않아 타임아웃이 발생한 상황입니다. 이는 주로 테이블 레벨의 동시성 제어 문제, 데드락 상황, 또는 장시간 실행되는 트랜잭션으로 인해 발생합니다.

#### **ERROR CODE: -75, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on instance %6\$d|%7\$d|%8\$d of class %9\$s. You are waiting for user(s) %10\$s to finish.'**

- 이 메시지는 CUBRID 시스템에서 특정 트랜잭션이 테이블의 인스턴스에서 잠금을 기다리는 중 시간 초과가 발생했음을 나타내는 락 타임아웃(lock timeout) 오류입니다. 이는 주로 객체(레코드) 레벨 동시성 제어 문제, 데드락 상황, 또는 장시간 실행되는 트랜잭션으로 인해 발생합니다.



**ERROR CODE: -76, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s on page %6\$d|%7\$d. You are waiting for user(s) %8\$s to release the page lock.'**

- 이 메시지는 CUBRID 시스템에서 특정 트랜잭션이 페이지에서 작업을 수행하기 위해 페이지 잠금을 기다리는 중 시간 초과가 발생했음을 나타내는 래치 타임아웃(latch timeout) 오류입니다, 이는 주로 동시성 제어 문제, 데드락 상황, 또는 장시간 실행되는 트랜잭션으로 인해 발생합니다.

**ERROR CODE: -106, 'Cannot prepare to commit the current transaction with global transaction identifier %1\$d because that transaction identifier is in use by another transaction.'**

- 이 메시지는 CUBRID 시스템이 2PC(Two-Phase Commit) 분산 트랜잭션에서 현재 트랜잭션을 커밋 준비하려고 할 때, 지정된 글로벌 트랜잭션 식별자가 이미 다른 활성 트랜잭션에 의해 사용 중이어서 준비 작업을 진행할 수 없을 때 발생하는 오류입니다.

**ERROR CODE: -107, 'There is no global distributed transaction associated with transaction identifier %1\$d.'**

- 이 메시지는 CUBRID 시스템이 특정 트랜잭션 식별자에 연결된 글로벌 분산 트랜잭션을 찾을 수 없을 때 발생하는 오류입니다, 즉, 2PC(Two-Phase Commit) 분산 트랜잭션 환경에서 클라이언트가 특정 트랜잭션 ID를 사용하여 분산 트랜잭션에 대한 작업을 시도했지만 해당 ID에 해당하는 글로벌 분산 트랜잭션이 존재하지 않거나 이미 완료된 경우에 발생합니다.

**ERROR CODE: -108, 'Current transaction %1\$d must be committed or aborted before attempting to join to the distributed transaction with global identifier %2\$d.'**

- 이 메시지는 CUBRID 시스템이 2PC(Two-Phase Commit) 분산 트랜잭션에 참여하려고 할 때, 현재 활성화된 트랜잭션이 있어서 분산 트랜잭션에 참여할 수 없을 때 발생하는 오류입니다, 분산 트랜잭션에 참여하기 전에 현재 트랜잭션을 완료(커밋 또는 중단)해야 함을 의미합니다.

**ERROR CODE: -198, 'Receive buffer too small, data truncated.'**

- 이 메시지는 CUBRID 데이터베이스에서 데이터를 수신하는 과정에서 할당된 버퍼의 크기가 수신하려는 데이터의 크기보다 작아서 데이터의 일부가 잘렸음을 나타냅니다, 주로 네트워크 통신이나 파일 I/O 작업에서 대용량 데이터를 처리할 때 발생하며, 데이터 무결성에 영향을 줄 수 있는 오류입니다.

**ERROR CODE: -440, 'Invalid cursor position.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 결과를 처리하는 과정에서 커서의 위치가 유효하지 않을 때 나타나는 오류입니다, 쿼리 프로세서에서 커서 위치 검증 과정에서 커서가 유효하지 않은 위치에 있거나, 커서 상태가 올바르지 않을 때입니다.

**ERROR CODE: -441, 'Invalid cursor operation.'**

- 이 메시지는 CUBRID 데이터베이스에서 OID가 포함되어 있지 않는 결과(cursor)에서 레코드의 OID를 추출하려고 할 때 발생하는 오류입니다, 쿼리 프로세서에서 커서 작업 검증 과정에서 커서가 지

원하지 않는 작업을 수행하려고 하거나, 커서 상태가 해당 작업을 허용하지 않을 때 발생할 수 있습니다.

**ERROR CODE: -442, 'Unknown cursor position.'**

- 이 메시지는 CUBRID 데이터베이스에서 커서의 위치가 예상치 못한 상태이거나 정의되지 않은 상태일 때 나타나는 오류입니다, 쿼리 프로세서에서 커서 위치 검증 과정에서 커서 위치가 예상치 못한 값일 때 발생할 수 있습니다.

**ERROR CODE: -443, 'Tuple value index %1\$d is out of range.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 결과의 튜플에서 특정 인덱스로 값을 가져오려고 할 때, 해당 인덱스가 유효한 범위를 벗어났을 때 나타나는 오류입니다, 쿼리 프로세서에서 튜플 값 인덱스 검증 과정에서 튜플의 컬럼 수보다 큰 인덱스나 음수 인덱스를 사용했을 때입니다.

**ERROR CODE: -444, 'Invalid attribute name: "%1\$s".'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 결과의 컬럼 이름이나 애트리뷰트 이름이 유효하지 않을 때 나타나는 오류입니다, 쿼리 프로세서에서 컬럼 이름 검증 과정에서 존재하지 않는 컬럼 이름이나 애트리뷰트 이름을 참조했을 때입니다.

**ERROR CODE: -447, 'Attempted to operate a query result structure already closed.'**

- 이 메시지는 CUBRID 데이터베이스에서 이미 닫힌 쿼리 결과 구조(query result structure)에 대해 어떤 작업을 수행하려고 했을 때 나타나는 오류입니다, 쿼리 프로세서에서 쿼리 결과 구조의 상태를 검증하는 과정에서 발생합니다, 애플리케이션이 쿼리 결과를 명시적으로 닫았거나, 트랜잭션 종료 등으로 인해 내부적으로 닫힌 쿼리 결과에 다시 접근하려 할 때 발생할 수 있습니다.

**ERROR CODE: -449, 'Unknown query identifier: %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 존재하지 않는 쿼리 식별자가 참조되었을 때 나타나는 오류입니다, 쿼리에서 쿼리 식별자 검증 과정에서 발생합니다, 즉, 정의되지 않은 쿼리 ID나 잘못된 쿼리 식별자가 사용된 경우입니다.

**ERROR CODE: -452, 'Invalid XASL tree node content.'**

- 이 메시지는 CUBRID 데이터베이스에서 XASL(eXtensible Abstract Syntax Language) 트리 노드의 내용이 유효하지 않을 때 나타나는 오류입니다, 쿼리에서 XASL 트리 노드 검증 과정에서 발생합니다, 즉, XASL 트리 노드의 구조나 내용이 올바르지 않은 경우입니다.

**ERROR CODE: -460, 'Too few input host variables provided.'**

- 이 메시지는 CUBRID 데이터베이스에서 쿼리 실행에 필요한 호스트 변수 수보다 적은 수의 호스트 변수가 제공되었을 때 나타나는 오류입니다.

**ERROR CODE: -550, 'Unknown savepoint name %1\$s'**

- 이 메시지는 CUBRID 데이터베이스에서 존재하지 않는 savepoint 이름을 참조할 때 나타나는 오류입니다, 주로 트랜잭션의 부분 롤백(partial rollback)을 수행할 때 지정된 savepoint가 존재하지 않을 때 발생합니다.

**ERROR CODE: -609, 'Your transaction cannot be rolled back since logging was ignored. Your database may be corrupted.'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션 롤백을 시도할 때 발생합니다, 로깅이 무시된 상태에서는 트랜잭션을 롤백 시도하거나 트랜잭션 복구가 불가능하여 데이터베이스가 손상될 가능성이 있습니다.

**ERROR CODE: -640, 'System Error: a savepoint cannot be added when the transaction is not active (e.g., during commit or rollback).'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션이 활성화되지 않은 상태에서 세이프포인트를 추가하려고 할 때 발생하는 시스템 오류입니다, 트랜잭션이 커밋(commit) 또는 롤백(rollback) 중이거나, 아직 시작되지 않았거나, 이미 종료된 상태에서는 세이프포인트를 생성할 수 없습니다, 이는 트랜잭션의 생명주기 관리와 관련된 내부 시스템 오류를 나타냅니다.

**ERROR CODE: -641, 'A name must be given for a savepoint.'**

- 이 메시지는 CUBRID 데이터베이스에서 세이프포인트를 생성할 때 이름이 제공되지 않았을 때 발생하는 오류입니다, 세이프포인트는 트랜잭션 내에서 특정 지점을 저장하는 기능으로, 반드시 고유한 이름이 필요합니다, 이는 세이프포인트 생성 시 필수 파라미터인 이름이 누락되었을 때 발생하는 검증 오류입니다.

**ERROR CODE: -642, 'System Error: there is not an active top system operation for current transaction.'**

- 이 메시지는 CUBRID 데이터베이스에서 현재 트랜잭션에 대한 활성화된 최상위 시스템 작업이 없을 때 발생하는 시스템 오류입니다, 시스템 작업을 종료하려고 할 때 해당 작업이 활성화되어 있지 않을 때 발생합니다, 이는 시스템 작업의 상태 관리와 관련된 내부 시스템 오류를 나타냅니다.

**ERROR CODE: -643, 'May be a system error: Committing/Aborting transaction = %1\$d (index = %2\$d) which has permanent operationsn attached to it. Will attach those system operations to the transaction'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션을 커밋하거나 중단할 때 영구적인 시스템 작업이 연결되어 있을 때 발생하는 경고입니다, 트랜잭션에 시스템 작업이 연결된 상태에서 커밋이나 중단을 시도할 때 발생합니다, 이는 시스템 오류일 가능성이 있지만, CUBRID가 자동으로 해당 시스템 작업들을 트랜잭션에 연결하여 처리합니다.

**ERROR CODE: -674, 'There is not enough active threads to continue current server execution. Current number of active threads is %1\$d. You may need up to %2\$d threads for clean server'**

**execution in your environment. Transaction index %3\$d has been timed out to continue server execution.'**

- 이 메시지는 CUBRID 데이터베이스 서버에서 작업 스레드들이 전부 '대기(suspend) 상태로 묶여 있어 당장 일을 처리할'활성(Active) 스레드가 모자라 보인다는 의미의 경고 메시지입니다.

**ERROR CODE: -735, 'Unknown transaction index %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스에서 유효하지 않은 트랜잭션 인덱스가 사용되었을 때 나타나는 오류입니다, 트랜잭션 인덱스는 CUBRID에서 각 트랜잭션을 고유하게 식별하기 위한 내부 식별자입니다, 이 메시지는 시스템이 존재하지 않거나 잘못된 트랜잭션 인덱스를 참조하려고 할 때 발생합니다, 즉, 트랜잭션 관리 시스템의 내부 상태 불일치나 메모리 손상으로 인해 발생할 수 있습니다.

**ERROR CODE: -830, 'Cannot allocate query entry any more. Maximum allocatable entries are %1\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 쿼리 관리 모듈에서 새로운 쿼리를 실행하기 위한 쿼리 엔트리를 할당할 때 발생합니다, 즉, 트랜잭션당 최대 쿼리 엔트리 수를 초과하여 새로운 쿼리 엔트리를 할당할 수 없을 때 발생하는 오류 메시지입니다.

**ERROR CODE: -836, 'LATCH ON PAGE(%1\$d|%2\$d) TIMEDOUT'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 페이지에 대한 래치(동기화 잠금) 획득 시도가 지정된 시간 내에 완료되지 않아 타임아웃이 발생했을 때 나타납니다, 래치(LATCH)는 데이터베이스 내부에서 페이지의 동시 접근을 제어하기 위한 경량 잠금 메커니즘으로, 다수의 트랜잭션이 동시에 같은 페이지에 접근할 때 경합이 심하거나, 장기 점유로 인해 래치 획득이 지연될 경우 발생할 수 있습니다, 주로 대량의 동시 트랜잭션, 비효율적인 쿼리, 시스템 부하, 내부 데드락, 버그 등으로 인해 래치 경합이 심해질 때 나타되며, 이 영향은 I/O 병목 현상에 의해 원하는 페이지를 파일로 부터 가지고 오지 못하는 경우나, btree를 분할 또는 병합 수행 중 dead-latch가 발생하는 경우입니다.

**ERROR CODE: -842, 'Cannot cache lock of object %1\$d|%2\$d|%3\$d.'**

- 이 메시지는 클라이언트 모듈에 캐시되어 있는 락(LOCK)을 확인 시 NULL LOCK 이거나 LOCK 변경 오류로 여러 객체(레코드)들에 대해 락을 획득하는 과정에서, 특정 객체(레코드)에 대한 락 획득이 실패했거나 락 캐시 작업이 실패했을 때 발생하는 오류 메시지입니다.

**ERROR CODE: -843, 'Wrong arguments in %1\$s, a %2\$s was passed.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager) 함수들에서 잘못된 인자가 전달되었을 때 발생합니다, 주로 애플리케이션 코드에서 내부적으로 락 관리자 함수를 호출할 때 매개 변수가 NULL인 경우 발생합니다.

**ERROR CODE: -844, 'Unknown isolation(%1\$d) is found. The transaction index is %2\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 트랜잭션의 격리 수준을 확인할 때, 시스템이 인식할 수 없는 잘못된 격리 수준 값을 발견했을 때 발생합니다, CUBRID는

TRAN\_READ\_COMMITTED, TRAN\_REPEATABLE\_READ, TRAN\_SERIALIZABLE와 같은 정의된 격리 수준을 지원합니다. 이 범위를 벗어나는 값이 발견되면 이 오류가 발생합니다. 이는 일반적으로 메모리 손상, 동시성 제어 문제, 또는 트랜잭션 관리자의 내부 상태 불일치로 인해 발생하는 오류입니다.

**ERROR CODE: -845, 'Invalid object type(%1\$d) is found. The lockable object is %2\$d|%3\$d|%4\$d.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 트랜잭션 hold 리스트에서 락 엔트리를 삭제하는 과정에서 발생합니다. CUBRID에서 지원하는 범위를 벗어나는 값이 발견되면 이 오류가 발생합니다. 이는 일반적으로 메모리 손상, 동시성 제어 문제, 또는 락 관리자의 내부 상태 불일치로 인해 발생하는 오류입니다.

**ERROR CODE: -846, '%1\$s of tran(%2\$d) is not found in the holder list of an object(%3\$d|%4\$d|%5\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 테이블 락 강등(demotion) 과정에서 발생합니다. 즉, 특정 객체(레코드)의 holder 리스트에서 트랜잭션의 락 엔트리를 찾으려고 할 때, 해당 엔트리가 holder 리스트에 존재하지 않는 비정상적인 상태를 발견했을 때 발생하는 오류 메시지입니다. 이런 상황은 일반적으로 CUBRID 내부의 락 관리 로직에 문제가 있거나, 메모리 손상, 또는 동시성 제어 문제로 인해 락 엔트리의 상태 정보가 일관되지 않게 되었을 때 나타납니다. 이 오류가 동일 시점 반복적으로 발생될 경우 데이터베이스 서비스 안정성에 영향을 줄 수 있습니다.

**ERROR CODE: -847, '%1\$s on a %2\$s(%3\$d|%4\$d|%5\$d) is not found in the transaction(%6\$d) hold list(cnt=%7\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 트랜잭션의 hold 리스트 관리 과정에서 발생합니다. 즉, 특정 트랜잭션의 hold 리스트에서 락 엔트리를 삭제하려고 할 때, 해당 엔트리가 리스트에 존재하지 않는 비정상적인 상태를 발견했을 때 발생하는 오류 메시지입니다. 이런 상황은 일반적으로 CUBRID 내부의 락 관리 로직에 문제가 있거나, 메모리 손상, 또는 동시성 제어 문제로 인해 락 엔트리의 상태 정보가 일관되지 않게 되었을 때 나타납니다. 이 오류가 동일 시점 반복적으로 발생될 경우 데이터베이스 서비스 안정성에 영향을 줄 수 있습니다.

**ERROR CODE: -849, 'Trying to abort a transaction(%1\$d) twice.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 데드락 해결 과정에서 발생합니다. 즉, 데드락으로 인해 트랜잭션을 중단(abort)하려고 할 때, 해당 트랜잭션이 이미 중단 중인 상태에서 다시 중단을 시도하는 비정상적인 상황을 발견했을 때 발생하는 오류 메시지입니다. 이런 상황은 일반적으로 데드락 감지 및 해결 과정에서 동시성 제어 문제나 트랜잭션 상태 관리의 불일치로 인해 발생합니다.

**ERROR CODE: -850, 'Tran(%1\$d) is neither lock holder nor lock waiter of an object(%2\$d|%3\$d|%4\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 락을 해제할 때 발생합니다. 즉, 특정 객체(레코드)에 대한 락을 해제하려고 할 때, 해당 트랜잭션이 그 객체(Object)의 락을 소

유한 holder도 아니고 락을 기다리는 waiter도 아닌 비정상적인 상태를 발견했을 때 발생하는 오류 메시지입니다, 이런 상황은 일반적으로 CUBRID 내부의 락 관리 로직에 문제가 있거나, 메모리 손상, 또는 동시성 제어 문제로 인해 락 리소스의 상태 정보가 일관되지 않게 되었을 때 나타납니다.

**ERROR CODE: -851, 'Failed to allocate lock resource (%1\$s).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 락 엔트리(lock entry)를 할당할 때 발생합니다, 즉, 새로운 락을 획득하거나 락 상태를 관리하기 위해 필요한 메모리 자원을 할당하는 과정에서 실패했을 때 발생하는 오류 메시지입니다, 이런 상황은 일반적으로 시스템 메모리 부족, 락 관리자의 내부 자원 고갈, 또는 메모리 할당자(memory allocator)의 문제로 인해 발생합니다. 이는 데이터베이스의 락 관리 기능이 정상적으로 동작하지 못하게 하는 오류입니다.

**ERROR CODE: -852, 'Total holders mode(%1\$d) of an empty lock holder list is not NULL\_LOCK.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 데드락 감지 과정에서 발생합니다, 즉, 락 리소스를 순회하면서 데드락을 감지할 때, 락을 소유한 holder는 없지만 NULL\_LOCK이 아닌 비정상적인 상태를 발견했을 때 발생하는 오류 메시지입니다.

**ERROR CODE: -854, 'WARNING: No lock holder, but lock waiters exist. refer to %1\$s.'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 데드락 감지 과정에서 발생합니다, 즉, 락 리소스를 순회하면서 데드락을 감지할 때, 락을 소유한 holder는 없지만 락을 기다리는 waiter는 존재하는 비정상적인 상태를 발견했을 때 발생하는 경고 메시지입니다, 이런 상황은 일반적으로 CUBRID 내부의 락 관리 로직에 문제가 있거나, 메모리 손상, 또는 동시성 제어 문제로 인해 락 리소스의 상태 정보가 일관되지 않게 되었을 때 나타납니다.

**ERROR CODE: -855, 'WARNING: Two or more threads are lock-waiting for one transaction(%1\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 락을 요청하는 과정에서 발생합니다, 이런 상황은 일반적으로 하나의 트랜잭션이 여러 스레드에서 동시에 실행되거나, 트랜잭션 내에서 중첩된 락 요청이 발생할 때 나타납니다.

**ERROR CODE: -856, 'WARNING: Strange lockwait state(%1\$d|%2\$d), thread(%3\$d|%4\$lld) and transaction(%5\$d).'**

- 이 메시지는 CUBRID 데이터베이스의 락 관리자(lock manager)에서 락 대기(lockwait) 상태를 확인하는 도중, 예상치 못한 비정상적인 상태를 감지했을 때 발생하는 경고 메시지입니다, 이런 상황은 일반적으로 CUBRID 내부의 락 관리 로직에 문제가 있거나, 메모리 손상, 또는 동시성 제어 문제로 인해 락 대기 상태 정보가 일관되지 않게 되었을 때 나타납니다.

**ERROR CODE: -859, 'LATCH ON PAGE(%1\$d|%2\$d) ABORTED'**

- 이 메시지는 CUBRID 데이터베이스에서 특정 페이지에 대한 래치(동기화 잠금) 획득 시도가 시스템 내부 정책이나 예외 상황(예: 데드락, 강제 중단, 트랜잭션 롤백 등)으로 인해 중단(Abort)되었을 때 발생합니다, 주로 동시성 제어 과정에서 데드락이 감지되거나, 장기 점유로 인한 강제 중단, 시스템 내부 오류, 트랜잭션 강제 종료 등으로 인해 래치 획득이 정상적으로 완료되지 못할 때 나



타며 이 영향은 I/O 병목 현상에 의해 원하는 페이지를 file로 부터 가지고 오지 못하는 경우나, btree를 분할 또는 병합 수행 중 dead-latch가 발생하는 경우입니다.

**ERROR CODE: -860, 'Cannot make current transaction as a part of a global transaction because its state is '%1\$s''**

- 이 메시지는 CUBRID 데이터베이스에서 2단계 커밋(2PC, Two-Phase Commit) 프로토콜을 사용하여 글로벌 트랜잭션을 시작하려고 할 때 발생합니다, 즉, 현재 트랜잭션이 활성 상태가 아니거나 적절한 상태가 아닌데, 글로벌 트랜잭션의 일부로 만들려고 시도할 때 발생하는 오류입니다, 이는 2PC 프로토콜의 올바른 사용과 트랜잭션 상태 관리의 무결성을 보장하기 위한 보호성 오류입니다.

**ERROR CODE: -861, 'Cannot prepare to commit the current transaction with transaction identifier %1\$d because that transaction is not made as a part of a global transaction.'**

- 이 메시지는 CUBRID 데이터베이스에서 2단계 커밋(2PC, Two-Phase Commit) 프로토콜을 사용하여 트랜잭션을 prepare하려고 할 때 발생합니다, 즉, 현재 트랜잭션이 글로벌 트랜잭션의 일부로 시작되지 않았는데, 2PC prepare를 시도할 때 발생하는 오류입니다, 이는 2PC 프로토콜의 올바른 사용 순서를 보장하고, 글로벌 트랜잭션의 무결성을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -862, 'Cannot set the user information of the global transaction with the identifier %1\$d because its state is '%2\$s''**

- 이 메시지는 CUBRID 데이터베이스에서 2단계 커밋(2PC, Two-Phase Commit) 프로토콜을 사용하는 글로벌 트랜잭션의 사용자 정보를 설정하려고 할 때 발생합니다, 즉, 글로벌 트랜잭션이 이미 2PC 프로토콜의 중간 단계에 있어서 사용자 정보를 변경할 수 없는 상태일 때 발생하는 오류입니다, 이런 상황은 일반적으로 분산 트랜잭션 환경에서 여러 데이터베이스 간의 트랜잭션 일관성을 보장하기 위해 2PC 프로토콜을 사용할 때 발생합니다, 이는 트랜잭션의 상태 관리와 데이터 일관성을 보호하기 위한 보호성 오류입니다.

**ERROR CODE: -966, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on object %6\$d|%7\$d|%8\$d because of deadlock. You are waiting for user(s) %9\$s to finish.'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션이 특정 객체(레코드)에 대한 락을 획득하려고 대기하는 동안 데드락 상황이 발생하여 타임아웃이 발생했음을 나타냅니다, 데드락은 두 개 이상의 트랜잭션이 서로가 보유한 리소스의 락을 기다리면서 무한 대기 상태에 빠지는 상황을 의미하며, CUBRID는 이를 감지하고 해결하기 위해 일부 트랜잭션을 중단시킵니다.

**ERROR CODE: -967, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on class %6\$s because of deadlock. You are waiting for user(s) %7\$s to finish.'**

- 이 메시지는 CUBRID 데이터베이스에서 트랜잭션이 특정 테이블에 대한 락을 획득하려고 대기하는 동안 데드락 상황이 발생하여 타임아웃이 발생했음을 나타냅니다, 데드락은 두 개 이상의 트랜잭션이 서로가 보유한 리소스의 락을 기다리면서 무한 대기 상태에 빠지는 상황을 의미합니다.

CUBRID는 데드락을 감지하고 이를 해결하기 위해 일부 트랜잭션을 중단시키는데, 이 경우 해당 트랜잭션이 데드락으로 인해 타임아웃되었음을 알려줍니다.

**ERROR CODE: -968, 'Your transaction (index %1\$d, %2\$s@%3\$s|%4\$d) timed out waiting on %5\$s lock on instance %6\$d|%7\$d|%8\$d of class %9\$s because of deadlock. You are waiting for user(s) %10\$s to finish.'**

- 이 메시지는 데드락 상황에서 특정 객체(레코드)에 대한 잠금을 획득하지 못하고 타임아웃이 발생할 때 나타납니다.

**ERROR CODE: -1002, 'The current transaction could not acquire a database lock it requires to continue processing.'**

- 이 메시지는 트랜잭션이 필요한 리소스에 대한 잠금을 획득하지 못했을 때 발생합니다.

**ERROR CODE: -1021, 'A deadlock cycle is detected. %1\$s.'**

- 이 메시지는 데이터베이스 시스템에서 두 개 이상의 트랜잭션이 서로가 점유하고 있는 잠금(Lock)이 해제되기를 기다리면서 더 이상 진행하지 못하는 순환 대기 상태, 즉 교착 상태(Deadlock)가 발생했음을 나타냅니다, CUBRID의 잠금 관리자(Lock Manager)가 이러한 교착 상태를 감지하고 시스템 전체의 멈춤을 방지하기 위해 관련된 트랜잭션 중 하나를 강제로 종료(롤백)시키면서 이 오류를 보고합니다.

**ERROR CODE: -1154, 'MVCC row %1\$d|%2\$d|%3\$d does not satisfies reevaluation.'**

- 이 메시지는 CUBRID 데이터베이스에서 MVCC(Multi-Version Concurrency Control) 환경에서 특정 레코드에 대한 조건 재평가(reevaluation)가 실패했을 때 발생합니다, 즉, 트랜잭션이 특정 레코드에 접근하려고 할 때, 해당 레코드가 현재 트랜잭션의 MVCC 조건을 만족하지 않음을 시스템이 감지했을 때 발생하는 오류입니다, 이런 상황은 일반적으로 동시성 제어 과정에서 레코드의 버전이 변경되었거나, 트랜잭션 격리 수준에 따른 가시성 조건이 만족되지 않을 때 발생합니다, 이는 데이터베이스의 일관성과 동시성 제어를 위한 보호성 오류입니다.(단, isolation level에 따라 재평가 달라짐)

**ERROR CODE: -1156, 'MVCC can't get snapshot'**

- 이 메시지는 CUBRID 데이터베이스의 MVCC(Multi-Version Concurrency Control) 시스템에서 트랜잭션의 스냅샷을 생성할 수 없을 때 발생하는 오류를 나타냅니다. MVCC 스냅샷은 트랜잭션이 시작될 때 데이터베이스의 일관된 상태를 캡처하는 중요한 메커니즘으로, 이를 통해 트랜잭션은 일관된 데이터 뷰를 유지할 수 있습니다.

**ERROR CODE: -1157, 'Isolation level value in MVCC must be 'read committed', 'repeatable read' or 'serializable''**

- 이 메시지는 CUBRID 데이터베이스에서 MVCC 환경에서 트랜잭션 격리 수준(Isolation Level)을 설정할 때, 지원되지 않는 값이 지정되었을 때 발생합니다, 즉, CUBRID의 MVCC 모드에서는 'read committed', 'repeatable read', 'serializable' 세 가지 격리 수준만 지원하는데, 이 범위를 벗어나는 값이 설정되려고 할 때 발생하는 검증 오류입니다, 이런 상황은 일반적으로 트랜잭션 격리 수준을



변경하는 SET TRANSACTION ISOLATION LEVEL 문이나 내부 API 호출에서 잘못된 값이 전달될 때 발생합니다, 이는 데이터베이스의 동시성 제어 메커니즘을 보호하기 위한 검증성 오류입니다.

**ERROR CODE: -1176, 'Latch promotion for page %1\$d of volume %2\$d failed.'**

- 이 메시지는 CUBRID 데이터베이스 시스템에서 특정 페이지(b-tree)에 대한 래치(latch) 승격(promote) 작업이 실패했음을 나타냅니다. 래치는 데이터베이스 내부에서 메모리 상의 데이터 구조(예: 페이지 버퍼의 페이지)에 대한 동시성 제어를 위해 사용되는 경량 잠금 메커니즘입니다., “래치 승격”은 일반적으로 공유 래치(shared latch)를 배타적 래치(exclusive latch)로 업그레이드하거나, 페이지에 대한 쓰기 작업을 수행하기 위해 더 높은 수준의 래치를 획득하려는 시도를 의미합니다, 이 실패는 페이지 버퍼 관리, 파일 관리 또는 B-트리 작업 중 발생할 수 있으며, 페이지의 일관성 유지에 필수적인 래치 획득에 문제가 생겼음을 시사합니다. 이는 동시성 문제, 페이지 상태 불일치 또는 내부적인 오류로 인해 발생할 수 있으며, 트랜잭션 실패나 데이터베이스 불안정으로 이어질 수 있습니다.

## CUBRID HA 오류

**ERROR CODE: -898, 'Replication error: "%1\$s"'**

- 이 메시지는 CUBRID 데이터베이스의 HA(High Availability) 복제 환경에서 복제 작업 중에 다양한 문제가 발생할 때 나타납니다, 즉, 데이터베이스 복제 과정에서 메모리 할당, 클래스 정보 처리, 복제 로그 생성 시 발생합니다.

**ERROR CODE: -970, 'Server HA mode is changed from '%1\$s' to '%2\$s''**

- 이 메시지는 CUBRID HA(High Availability) 복제 환경에서 모드가 변경될 때 발생하는 시스템 알림입니다. 서버의 상태가 active, standby, maintenance 등의 상태 간에 전환될 때 발생하며, 이는 시스템의 정상적인 HA 운영 과정의 일부입니다.

**ERROR CODE: -975, 'Current version of replication does not allow changing multiple rows with a single UPDATE statement which can violate the UNIQUE constraint.'**

- 이 메시지는 CUBRID HA(High Availability) 복제 환경에서 발생하는 특별한 제한사항과 관련이 있습니다. 일반적으로 단일 UPDATE 문으로 여러 행을 변경하는 것은 허용되지만, HA 복제가 활성화된 환경에서는 UNIQUE 제약 조건을 위반할 수 있는 다중 행 UPDATE 작업이 제한됩니다, 이는 복제 환경에서 데이터 일관성을 보장하고, 마스터와 슬레이브 간의 동기화 문제를 방지하기 위한 보호성 오류입니다. HA 모드가 활성화된 경우(`ha\_mode`가 `off`가 아닌 경우), 시스템은 UNIQUE 제약 조건 위반 가능성이 있는 다중 행 UPDATE 작업을 거부합니다.

**ERROR CODE: -986, 'CUBRID heartbeat feature started.'**

- 이 메시지는 CUBRID HA(High Availability) 복제 환경에서 heartbeat 서브시스템이 성공적으로 시작 되었음을 알리는 정보성 알림입니다. 하트비트 기능은 CUBRID 클러스터 환경에서 각 노드의 상태를 주기적으로 확인하고, 장애 발생 시 이를 감지하여 자동 전환(Failover)을 수행하는 데 필수적인 역할을 합니다. 이 메시지는 일반적으로 CUBRID 마스터 프로세스 시작 시, cubrid heartbeat

start 명령 실행 시, 또는 HA 모드가 활성화될 때 발생하며, 시스템이 고가용성 모드로 정상 작동하기 시작했음을 의미합니다.

**ERROR CODE: -987, 'CUBRID heartbeat feature stopped.'**

- 이 메시지는 CUBRID HA(High Availability) 복제 환경에서 HA heartbeat 서브시스템이 비활성화 되었음을 알리는 알림성 이벤트입니다. 관리자 중지 명령, 마스터 종료/재시작, HA 모드 OFF 전환, 또는 비활성화 요청 처리 흐름에서 발생하며, 이후 장애 감지와 자동 전환 기능이 중단됩니다.

**ERROR CODE: -988, 'Node event: %1\$s.'**

- 이 메시지는 CUBRID HA(High Availability) 시스템에서 노드의 상태 변경이나 이벤트 발생을 알리는 정보성 메시지입니다. 노드의 상태 모니터링과 관련된 다양한 이벤트(연결, 연결 해제, 상태 변경 등)를 기록합니다.

**ERROR CODE: -989, 'Process event: %1\$s. %2\$s'**

- 이 메시지는 CUBRID HA(High Availability) 시스템에서 프로세스 이벤트 발생을 알리는 시스템 메시지입니다. 서버의 상태 변화나 중요 이벤트를 추적하고 기록하는 데 사용되는 핵심 모니터링 메시지입니다.

**ERROR CODE: -990, 'Command execution: %1\$s. %2\$s'**

- 이 메시지는 CUBRID HA(High Availability) 시스템 내부에서 특정 명령을 실행하는 과정에서 문제가 발생 했음을 나타냅니다. 특히, 이 에러는 CUBRID 마스터 프로세스의 핵심 기능인 heartbeat 관리 또는 요청 처리 중에 명령 실행 실패가 발생 했음을 시사합니다, 실제 오류 발생 시 구체적인 명령 내용이나 실패 원인에 대한 추가 정보를 제공되며, 이는 시스템 내부의 중요한 프로세스가 정상적으로 작동하지 못했음을 의미하고, 데이터베이스의 안정성과 가용성에 직접적인 영향을 미칠 가능성이 있습니다.

**ERROR CODE: -1023, 'log applier filter: %1\$s.'**

- 이 메시지는 CUBRID HA(High Availability) 복제 환경에서, 액티브(Active) 서버(마스터)의 변경 로그를 스탠바이(Standby) 서버(슬레이브/리플리카)에 적용하는 로그 적용기(Log Applier)가 사용자가 정의한 복제 필터 규칙을 처리하던 중 일반적인(Generic) 오류를 만났을 때 발생합니다, 복제 필터는 특정 테이블이나 데이터 변경 연산(INSERT, UPDATE, DELETE)을 복제 대상에서 제외하거나 포함시키는 규칙을 정의하는데, 이 규칙의 처리 과정에서 문제가 생겼음을 의미합니다.

**ERROR CODE: -1024, 'log writer: failed to get log page(s) starting from page id %1\$lld.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 copylogdb 프로세스가 복제 대상 server로 부터 복제 로그를 요청시 발생합니다.

**ERROR CODE: -1025, 'HA delay info: The replication delay (%1\$d secs) has exceeded the limit (%2\$d secs):'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 마스터 서버와 스탠바이 서버 간의 복제 지연 시간이 사용자가 설정한 임계값(ha\_delay\_limit, ha\_delay\_limit\_delta)을 초과했음을 나타냅니다.

즉, 스탠바이 서버가 마스터 서버의 최신 변경 사항을 따라잡는 데 설정된 시간보다 오래 걸리고 있다는 의미입니다. 이는 마스터와 스탠바이 간의 데이터 동기화 상태가 양호하지 않음을 알리는 경고성 또는 정보성 메시지일 수 있으며, 설정에 따라 특정 동작(예: 마스터의 추가 트랜잭션 지연)을 유발할 수도 있습니다.

**ERROR CODE: -1026, 'HA delay info: The replication delay (%1\$d secs) is decreased below the acceptable level (%2\$d secs).'**

- 이 메시지는 이전에 발생했던 CUBRID HA(High Availability) 환경에서 복제 지연 상태가 해소되었음을 알리는 정보성 로그입니다. 즉, 액티브(마스터) 서버와 스탠바이 서버 간의 복제 지연 시간이 다시 사용자가 설정한 임계값(ha\_delay\_limit, ha\_delay\_limit\_delta) 이하로 줄어들어, 데이터 동기화 상태가 정상 범위로 복귀했음을 의미합니다.

**ERROR CODE: -1027, 'log applier: failed to change apply state from '%1\$s' to '%2\$s'.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 내부 시스템로 부터 applylogdb의 동작 상태 변경 요청시 변경할 수 없는 경우 발생하는 오류 입니다.

**ERROR CODE: -1028, 'log applier: unexpected EOF record in archive log. LSA: %1\$lld|%2\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 복제된 archive log에서 예상치 않은 로그(END of LOG)을 만나는 경우 발생하는 오류 메세지로 applylogdb 프로세스는 해당 로그 레코드를 적용하지 않고 진행됩니다.

**ERROR CODE: -1029, 'log applier: invalid replication log page/offset. page HDR: %1\$lld|%2\$d, final: %3\$lld|%4\$d, append LSA: %5\$lld|%6\$d, EOF LSA: %7\$lld|%8\$d, ha file status: %9\$d, is end-of-log: %10\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 applylogdb 프로세스가 복제 로그를 처리하는 과정에서 유효하지 않은 페이지 ID나 오프셋을 발견했을 때 발생합니다. 로그 레코드의 위치 정보가 올바르지 않아 복제 작업을 진행할 수 없는 상태를 나타냅니다.

**ERROR CODE: -1030, 'log applier: invalid replication record. LSA: %1\$lld|%2\$d, forw LSA: %3\$lld|%4\$d, backw LSA: %5\$lld|%6\$d, Trid: %7\$d, prev tran LSA: %8\$lld|%9\$d, type: %10\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 applylogdb 프로세스가 복제 로그 레코드를 검증하는 과정에서 발생합니다, 검증 중 로그 레코드의 LSA 체인(순방향·역방향 연결)이나 트랜잭션 정보가 올바르지 않은 것으로 판단될 경우 해당 오류가 발생합니다.

**ERROR CODE: -1031, 'log applier: failed to apply the following statement. class: "%1\$s", statement: "%2\$s", server error: %3\$d, %4\$s.'**

- 이 메시지는 CUBRID HA(High Availability) 환경의 스탠바이 서버에서 applylogdb 프로세스가 마스터 서버로부터 받은 변경 로그(SQL 문장 형태)를 스탠바이 데이터베이스에 실제로 적용(실행)하는 과정에서 실패했을 때 발생합니다. 즉, 로그 자체는 정상적으로 수신하고 해석했지만, 스탠바이 서버의 현재 데이터베이스 상태에서 해당 SQL(주로 DDL) 구문을 실행할 수 없었음을 의미합니다.

다. 실패 원인은 스키마 불일치, 데이터 충돌(예: 고유 키 위반), 제약 조건 위반, 리소스 부족 등 다양하며, 메시지에 포함된 '서버 오류' 상세 오류 정보가 구체적인 실패 사유를 알려줍니다.

**ERROR CODE: -1032, 'log applier: failed to apply insert replication log. class: "%1\$s", key: "%2\$s", server error: %3\$d, %4\$s'**

- 이 메시지는 CUBRID HA(High Availability) 환경의 스탠바이 서버에서 applylogdb 프로세스가 마스터 서버로부터 복제된 INSERT 로그를 스탠바이 데이터베이스에 적용(실행)하려다 실패했을 때 발생합니다. 가장 흔한 원인은 스탠바이 서버에 해당 키를 가진 레코드가 이미 존재하여 기본 키 또는 고유 키 제약 조건 위반(Duplicate Key)이 발생하는 경우입니다. 그 외에도 스키마 불일치, 다른 제약 조건 위반 등 INSERT 문 실행을 막는 다양한 원인이 있을 수 있으며, 포함된 '서버 오류' 정보가 실패의 직접적인 이유를 알려줍니다.

**ERROR CODE: -1033, 'log applier: failed to apply update replication log. class: "%1\$s", key: "%2\$s", server error: %3\$d, %4\$s'**

- 이 메시지는 CUBRID HA(High Availability) 환경의 스탠바이 서버에서 applylogdb 프로세스가 마스터 서버로부터 복제된 UPDATE 로그를 스탠바이 데이터베이스에 적용(실행)하려다 실패했을 때 발생합니다. 이는 특정 사례로, UPDATE 작업이 스탠바이 서버에서 거부되었음을 의미합니다. 주요 원인으로서는 업데이트 대상 레코드가 스탠바이 서버에 존재하지 않거나, 업데이트 결과가 고유 키/기본 키 제약 조건 또는 다른 제약 조건(NOT NULL, FK 등)을 위반하거나, 스키마 불일치 등이 있고, 메시지에 포함된 '서버 오류' 정보가 실패의 구체적인 이유입니다.

**ERROR CODE: -1034, 'log applier: failed to apply delete replication log. class: "%1\$s", key: "%2\$s", server error: %3\$d, %4\$s'**

- 이 메시지는 CUBRID HA(High Availability) 환경의 스탠바이 서버에서 applylogdb 프로세스가 마스터 서버로부터 복제된 DELETE 로그를 스탠바이 데이터베이스에 적용(실행)하려다 실패했을 때 발생합니다. 이는 특정 사례로, DELETE 작업이 스탠바이 서버에서 거부되었음을 의미합니다. 주요 원인으로서는 업데이트 대상 레코드가 스탠바이 서버에 존재하지 않거나, 업데이트 결과가 고유 키/기본 키 제약 조건 또는 다른 제약 조건(NOT NULL, FK 등)을 위반하거나, 스키마 불일치 등이 있고, 메시지에 포함된 '서버 오류' 정보가 실패의 구체적인 이유입니다.

**ERROR CODE: -1035, 'log applier: mem size(%1\$d MB) of log applier is greater than max mem size (%2\$d MB) or has been grow more than 2 times (%3\$d MB). required LSA: %4\$lld|%5\$d. last committed LSA: %6\$lld|%7\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경의 스탠바이 서버에서 applylogdb 프로세스가 사용하는 메모리 양이 설정된 최대 허용치(*ha\_log\_applier\_max\_mem\_size*)를 초과하거나, 비정상적으로 빠르게 (단기간에 2배 이상) 증가했을 때 발생합니다. 이는 스탠바이 서버의 자원 고갈을 방지하고 시스템 안정성을 유지하기 위한 보호 메커니즘으로 주로 마스터 서버에서 매우 큰 트랜잭션 (많은 로그를 생성하는)이 발생했거나, 스탠바이 서버의 처리 성능 부족 또는 네트워크 지연 등으로 인해 로그 적용이 지연되어 처리해야 할 로그가 메모리에 과도하게 쌓일 때 발생할 수 있습니다. 'required LSA'와 'last committed LSA'의 차이가 크다면, 로그 적용이 상당히 지연되고 있음을 나타냅니다.

**ERROR CODE: -1036, 'log applier: log applier shut itself down by signal.'**

- 이 메시지는 CUBRID의 HA(High Availability) 시스템에서 발생하는 applylogdb 프로세스의 정상적인 종료 상황을 나타냅니다, 시스템 시그널을 받아 로그 복제 프로세스가 정상적인 종료 절차를 수행했음을 의미합니다. 이는 일반적으로 계획된 시스템 종료나 관리자의 의도적인 종료 요청에 의해 발생합니다.

**ERROR CODE: -1037, 'log writer: log writer shut itself down by signal.'**

- 이 메시지는 CUBRID의 HA(High Availability) 시스템에서 발생하는 copylogdb 로그 복제 프로세스의 정상적인 종료 상황을 나타냅니다, 시스템 시그널을 받아 로그 복제 프로세스가 정상적인 종료 절차를 수행했음을 의미합니다, 이는 일반적으로 계획된 시스템 종료나 관리자의 의도적인 종료 요청에 의해 발생합니다.

**ERROR CODE: -1038, 'log applier: log applier started. required LSA: %1\$lld|%2\$d. last committed LSA: %3\$lld|%4\$d. last committed repl LSA: %5\$lld|%6\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 applylogdb 프로세스가 시작될 때 출력되는 정보성 메시지입니다. 시작 시점의 LSA 정보를 포함하여 로그 복제의 시작 위치와 마지막 커밋 상태를 나타냅니다. LSA는 로그의 위치를 식별하는 주소값으로, 데이터 동기화의 기준점이 됩니다

**ERROR CODE: -1039, 'log writer: log writer started. mode: %1\$d.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 copylogdb 프로세스가 시작될 때 출력되는 정보성 메시지입니다. 시작 시점의 LSA 정보를 포함하여 로그 복제의 시작 위치와 마지막 커밋 상태를 나타냅니다. LSA는 로그의 위치를 식별하는 주소값으로, 데이터 동기화의 기준점이 됩니다, 이는 copylogdb 프로세스의 정상 구동을 알리는 메시지입니다.

**ERROR CODE: -1040, 'HA generic: %1\$s.'**

- 이 메시지는 CUBRID HA(High Availability) 시스템 내에서 발생한 문제 중, 특정 오류 코드로 분류되지 않은 예외적이거나 일반적인 오류 상황을 나타냅니다. 즉, HA 관련 기능(heartbeat, copylogdb, applylogdb 등)을 수행하는 도중 예상치 못한 내부 상태나 외부 요인으로 인해 더 이상 진행할 수 없을 때 사용되는 포괄적인 오류 코드입니다. 따라서 여러 메시지에 포함된 여러 상세한 오류 메시지들의 내용이 실제 오류의 원인을 진단하는데 결정적인 단서가 됩니다.

**ERROR CODE: -1105, 'Internal error: partially failed to flush dirty object.'**

- 이 메시지는 CUBRID 데이터베이스에서 메모리에 있는 변경된 데이터(dirty object)를 디스크로 저장하는 과정에서 일부만 성공하고 일부가 실패했을 때 발생합니다, 즉, 트랜잭션의 일부 변경사항은 성공적으로 디스크에 저장되었지만, 나머지 일부는 저장에 실패했음을 의미합니다, 이는 데이터 무결성과 시스템 안정성을 보장하기 위한 내부 오류로, 주로 applylogdb에서 데이터 반영시 발생할 수 있는 오류로 복제(replication) 과정에서 발생할 수 있습니다.

**ERROR CODE: -1122, 'Time out in receiving data.'**

- 이 메시지는 CUBRID HA(High Availability) 시스템 내에서 발생한 문제 중 copylogdb 프로세스에서 발생하는 실행 오류로 cub\_server 프로세스로 부터 log page를 요청 후 timeout 까지 요청한 log page를 받지 못한 경우 발생합니다.

**ERROR CODE: -1133, 'Copylogdb for %1\$s already exists.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 copylogdb 가 cub\_master와 연결 요청시 이미 copylogdb가 구동 중일때 발생합니다.

**ERROR CODE: -1134, 'Applylogdb for %1\$s already exists.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 applylogdb가 cub\_master와 연결 요청시 이미 applylogdb가 구동 중일때 발생합니다.

**ERROR CODE: -1139, 'Handshake error (peer host %1\$s): incompatible read/write mode. (client: %2\$s, server: %3\$s)'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 데이터베이스 클라이언트와 서버 간의 핸드셰이크 과정에서 읽기/쓰기 모드의 호환성 문제가 발생할 때 나타납니다, 즉, 데이터베이스 클라이언트와 데이터베이스 서버간 핸드셰이크 과정에서 클라이언트와 서버의 모드(RW/RO)가 다른 경우 발생합니다.

**ERROR CODE: -1140, 'Handshake error (peer host %1\$s): HA replication delayed.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 데이터베이스 서버 간 복제 과정에서 지연이 발생할 때 나타납니다. 마스터-슬레이브 간의 데이터 동기화가 설정된 임계값을 초과하여 지연되는 상황을 나타내는 오류입니다.

**ERROR CODE: -1141, 'Handshake error (peer host %1\$s): replica-only client to non-replica server.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 리플리카 전용으로 설정된 서버에 클라이언트가 직접 접근을 시도할 때 발생합니다, 즉, 데이터베이스 클라이언트와 데이터베이스 서버간 핸드셰이크 과정에서 리플리카 전용 클라이언트(replica broker)가 리플리카 서버가 아닌 마스터, 슬레이브 서버로 연결 요청 시 발생합니다.

**ERROR CODE: -1142, 'Handshake error (peer host %1\$s): unidentified server version.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 클라이언트가 데이터베이스 서버에 연결을 시도할 때 서버의 버전 정보를 확인할 수 없는 경우 발생합니다. 클라이언트-서버 간의 핸드셰이크 과정에서 서버 버전 정보가 올바르게 전달되지 않거나 호환되지 않는 상황을 나타냅니다.

**ERROR CODE: -1143, 'Handshake error (peer host %1\$s): remote access to server not allowed.'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 데이터베이스 클라이언트가(csql, cas, copylogdb, applylogdb등..) 서버 간의 버전 정보를 식별할 수 없을 때 발생하는 오류입니다.



**ERROR CODE: -1144, 'Timed out attempting to connect to %1\$s. (timeout: %2\$d sec(s))'**

- 이 메시지는 CUBRID 클라이언트 또는 구성 요소(예: CSQL, JDBC 드라이버 등)가 지정된 서버에 연결을 시도했으나 타임아웃 설정된 시간 이내에 응답을 받지 못해 연결이 실패했을 때 발생하는 오류입니다, 일반적으로 네트워크 장애, 서버 비정상 상태, 방화벽 차단, 포트 오픈 실패 등의 원인으로 발생할 수 있습니다.
- 참조: 이 메시지는 큐브리드 매뉴얼 CUBRID HA 복제 불일치 감지 부분에 “장비 down”으로 정리되어 있습니다.

**ERROR CODE: -1170, 'Failed to flush replication items (error code : %1\$d).'**

- 이 메시지는 CUBRID HA(High Availability) 환경에서 데이터베이스의 복제(replication) 과정에서 복제 항목, 즉 마스터 데이터베이스에서 발생한 변경 사항들을 슬레이브(replica) 데이터베이스에 적용하기 위해 버퍼에서 디스크로 쓰거나 다음 단계로 전달하는 작업이 실패했음을 나타냅니다. 이는 네트워크 문제, 디스크 I/O 문제, 슬레이브 데이터베이스의 내부 상태 문제, 또는 복제 항목 자체의 문제 등 다양한 원인으로 발생할 수 있습니다.

## CUBRID HA

High Availability(HA)란, 하드웨어, 소프트웨어, 네트워크 등에 장애가 발생해도 지속적인 서비스를 제공하는 기능이다. 이 기능은 하루 24시간 1년 내내 서비스를 제공해야 하는 네트워킹 컴퓨팅 부분에서 필수적인 요소이다. HA 시스템은 두 대 이상의 서버 시스템으로 구성하여 시스템 구성 요소 중의 한 요소에 장애가 발생해 서비스를 중단 없이 제공할 수 있다.

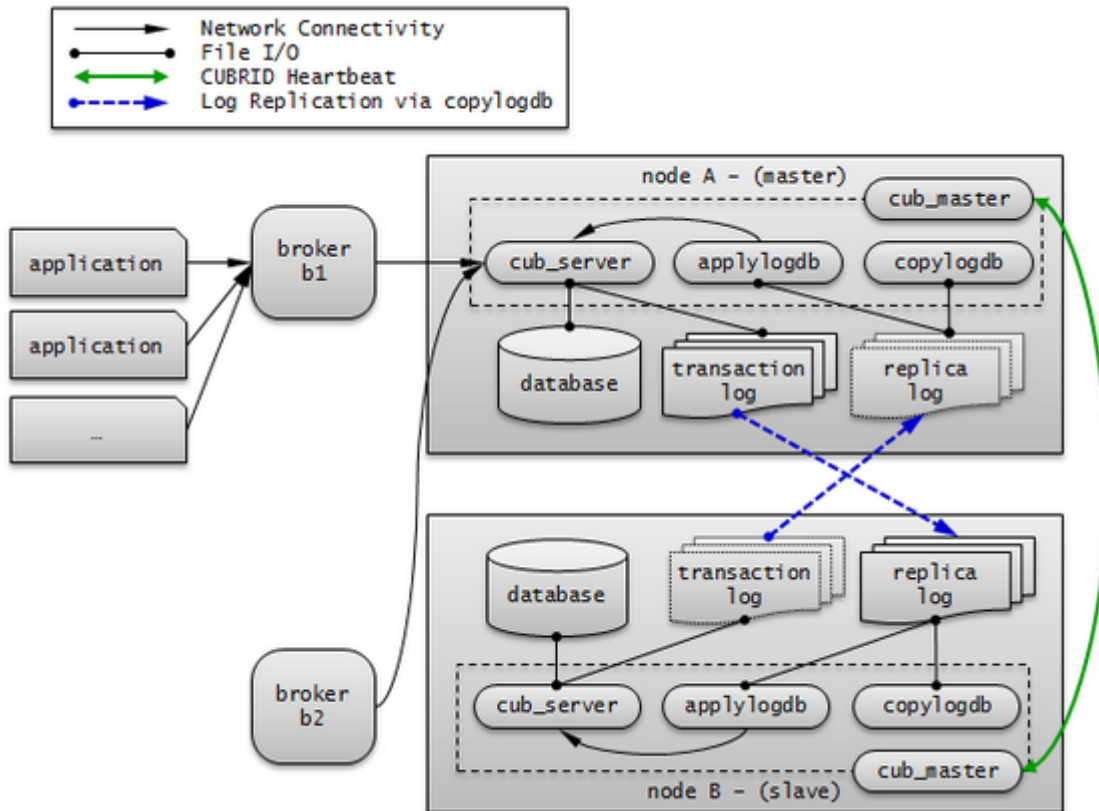
High Availability 기능을 CUBRID에 적용한 것이 CUBRID HA 기능이다. CUBRID HA 기능은 여러 서버 시스템에서 데이터베이스를 항상 동기화된 상태로 유지하여 서비스를 제공한다. 서비스를 수행 중인 시스템에 예상치 못한 장애가 발생하면 자동으로 다른 시스템이 서비스를 수행하도록 하여 서비스 중단 시간을 최소화한다.

CUBRID의 HA 기능은 shared-nothing 구조이며, 액티브 서버(active server)에서 스탠바이 서버(standby server)로 데이터를 동기화하기 위해 다음 두 단계를 수행한다.

1. 트랜잭션 로그 다중화: 액티브 서버에서 생성되는 트랜잭션 로그를 실시간으로 다른 노드에 복제한다.
2. 트랜잭션 로그 반영: 실시간으로 복제된 트랜잭션 로그를 분석하여 스탠바이 서버에 데이터를 반영한다.

CUBRID HA 기능은 위 두 단계를 수행하여 액티브 서버와 스탠바이 서버에 항상 동기화된 데이터를 유지한다. 따라서, 서비스를 제공 중이던 마스터 노드(master node)에 예상치 못한 장애가 발생하여 액티브 서버가 정상적으로 동작하지 못하면 슬레이브 노드(slave node)의 스탠바이 서버가 액티브 서버를 대신하여 중단 없는 서비스를 제공할 수 있다. CUBRID HA 기능은 시스템과 CUBRID의 상태

를 실시간으로 감시하고 장애가 발생하면 자동 failover를 수행하기 위해 heartbeat 메시지를 사용한다.



## CUBRID HA 기본 개념

### 노드와 그룹

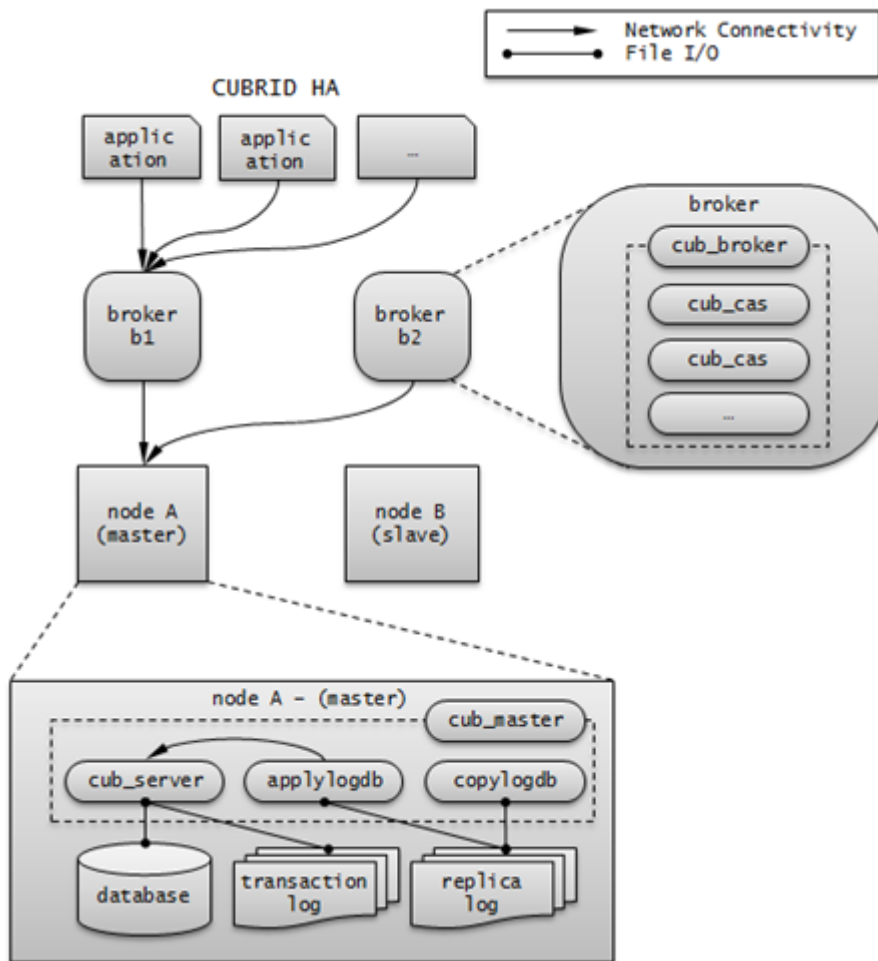
노드는 CUBRID HA를 구성하는 논리적인 단위로, 노드는 상태에 따라 마스터 노드(master node), 슬레이브 노드(slave node), 레플리카 노드(replica node) 등으로 나눈다.

- **마스터 노드**: 복제의 대상이 되는 노드로, 액티브 서버를 사용한 읽기, 쓰기 등 모든 서비스를 제공한다.
- **슬레이브 노드**: 마스터 노드와 동일한 내용을 갖는 노드로, 마스터 노드의 변경이 자동으로 반영된다. 스탠바이 서버를 사용한 읽기 서비스를 제공하며 마스터 노드 장애 시 failover가 일어난다.
- **레플리카 노드**: 마스터 노드와 동일한 내용을 갖는 노드로, 마스터 노드의 변경이 자동으로 반영된다. 스탠바이 서버를 사용한 읽기 서비스를 제공하며 마스터 노드 장애 시 failover가 일어나지 않는다.

CUBRID HA 그룹은 위와 같은 노드들로 이루어지며, 그룹의 멤버는 **cubrid\_ha.conf**의 **ha\_node\_list** 및 **ha\_replica\_list**로 설정할 수 있다. 그룹 내의 노드들은 동일한 내용을 가지며, 주기적으로 상태 확인 메시지를 주고 받고 마스터 노드에 장애가 발생하면 failover가 일어난다.



노드에는 마스터 프로세스(cub\_master), 데이터베이스 서버 프로세스(cub\_server), 복제 로그 복사 프로세스(copylogdb) 및 복제 로그 반영 프로세스(applylogdb) 등이 포함된다.

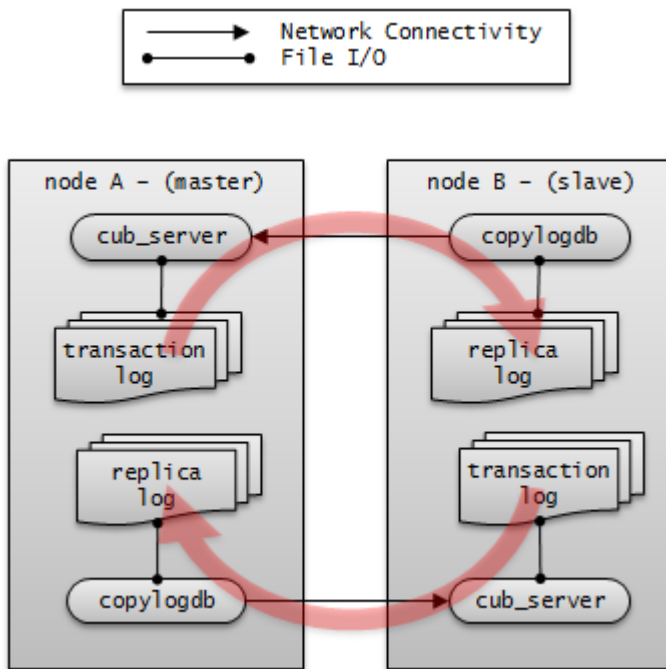


## 프로세스

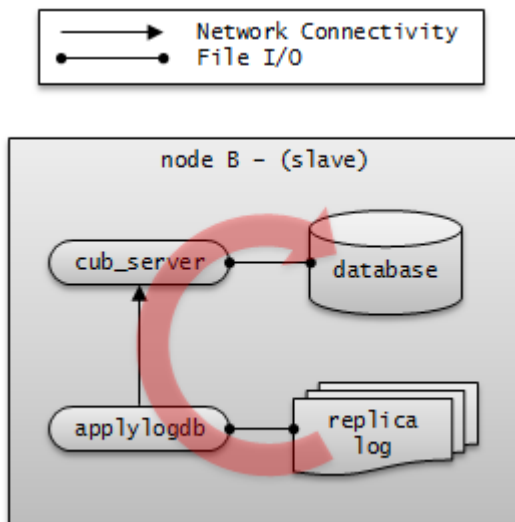
CUBRID HA 노드는 하나의 마스터 프로세스(cub\_master), 하나 이상의 데이터베이스 서버 프로세스(cub\_server), 하나 이상의 복제 로그 복사 프로세스(copylogdb), 하나 이상의 복제 로그 반영 프로세스(applylogdb)로 이루어져 있다. 하나의 데이터베이스를 설정하면 데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스가 구동된다. 복제 로그의 복사와 반영은 서로 다른 프로세스에 의해 수행되므로 복제 반영의 지연은 실행 중인 트랜잭션에 영향을 주지 않는다.

- **마스터 프로세스(cub\_master)** : heartbeat 메시지를 주고 받으며 CUBRID HA 내부 관리 프로세스들을 제어한다.
- **데이터베이스 서버 프로세스(cub\_server)** : 사용자에게 읽기, 쓰기 등의 서비스를 제공한다. 자세한 내용은 [서버](#) 를 참고한다.
- **복제 로그 복사 프로세스(copylogdb)** : 그룹 내의 모든 트랜잭션 로그를 복사한다. 복제 로그 복사 프로세스가 대상 노드의 데이터베이스 서버 프로세스에 트랜잭션 로그를 요청하면, 해당 데이터베이스 서버 프로세스는 적절한 로그를 전달한다. 트랜잭션 로그가 복사되는 위치는 **cubrid\_ha.conf**의 **ha\_copy\_log\_base**로 설정할 수 있다. 복사된 복제 로그의 정보는 **applyinfo** 유

유틸리티로 확인할 수 있다. 복제 로그 복사는 SYNC, ASYNC의 두 가지 모드가 있으며, 모드는 **cubrid\_ha.conf**의 **ha\_copy\_sync\_mode**로 설정할 수 있다. 모드에 대한 자세한 내용은 **로그 다중화**를 참고한다.



- 복제 로그 반영 프로세스(**applylogdb**) : 복제 로그 복사 프로세스에 의해 복사된 로그를 노드에 반영한다. 반영한 복제 정보는 내부 카탈로그(**\_db\_ha\_apply\_info**)에 저장하며, 이 정보는 **applyinfo** 유틸리티로 확인할 수 있다.

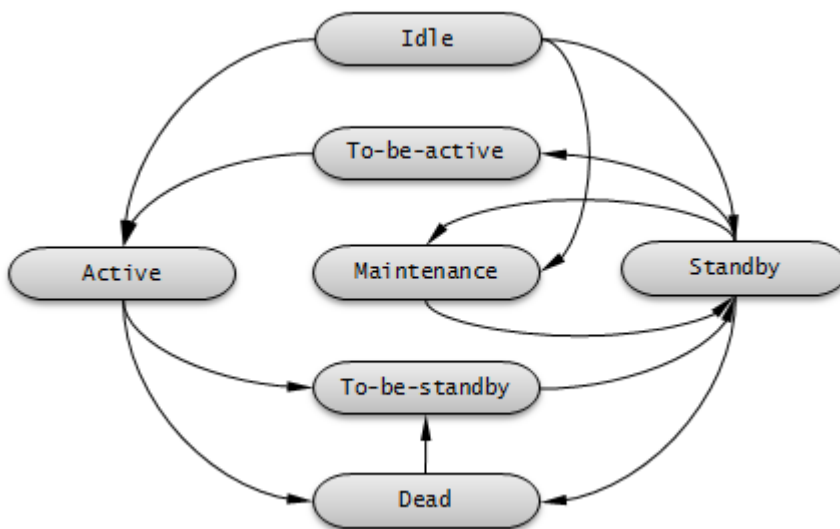


## 서버

서버란 데이터베이스 서버 프로세스를 논리적으로 표현하는 단어로, 상태에 따라 액티브 서버(active server), 스탠바이 서버(standby server)로 나눈다.

- **액티브 서버** : 마스터 노드에 속하는 서버로, active 상태이다. 액티브 서버는 사용자에게 읽기, 쓰기 등 모든 서비스를 제공한다.
- **스탠바이 서버** : 마스터 노드 외의 노드에 속하는 서버로, standby 상태이다. 스탠바이 서버는 사용자에게 읽기 서비스만을 제공한다.

서버 상태는 노드 상태에 따라 변경된다. **cubrid changemode** 유틸리티를 이용하면 서버 상태를 조회할 수 있다. maintenance 모드는 운영 편의를 위한 것으로, **cubrid changemode** 유틸리티를 통해 변경할 수 있다.



- **active** : 일반적으로 마스터 노드에서 실행 중인 서버들은 active 상태이다. 읽기, 쓰기 등 모든 서비스를 제공한다.
- **standby** : 슬레이브 노드 또는 레플리카 노드에서 실행 중인 서버들은 standby 상태이다. 읽기 서비스만을 제공한다.
- **maintenance** : 운영 편의를 위해 수동으로 변경 가능한 상태로, 로컬 호스트의 csq1만 접속할 수 있으며, 사용자에게는 서비스를 제공할 수 없다.
- **to-be-active** : 스탠바이 서버가 failover 등의 이유로 인해 액티브 서버가 되기 전의 상태이다. 기존의 마스터 노드로부터 받은 트랜잭션 로그를 자신의 서버에 반영하는 등 액티브 서버가 되기 위한 준비를 한다. 해당 상태의 노드에는 SELECT 질의만 수행할 수 있다.
- 기타: 내부적으로 사용하는 상태이다.

노드 상태가 변경되면 cub\_master 프로세스 로그와 cub\_server 프로세스 로그에 각각 다음과 같은 에러 메시지가 저장된다. 단, **cubrid.conf**에서 **error\_log\_level**의 값이 **error** 이하인 경우에 저장된다.

- cub\_master 프로세스의 로그 정보는 **\$CUBRID/log/<hostname>\_master.err** 파일에 저장되며 다음의 내용이 기록된다.

```
HA generic: Send changemode request to the server. (state:
1[active], args:[cub_server demodb], pid:25728).
HA generic: Receive changemode response from the server. (state:
1[active], args:[cub_server demodb], pid:25728).
```

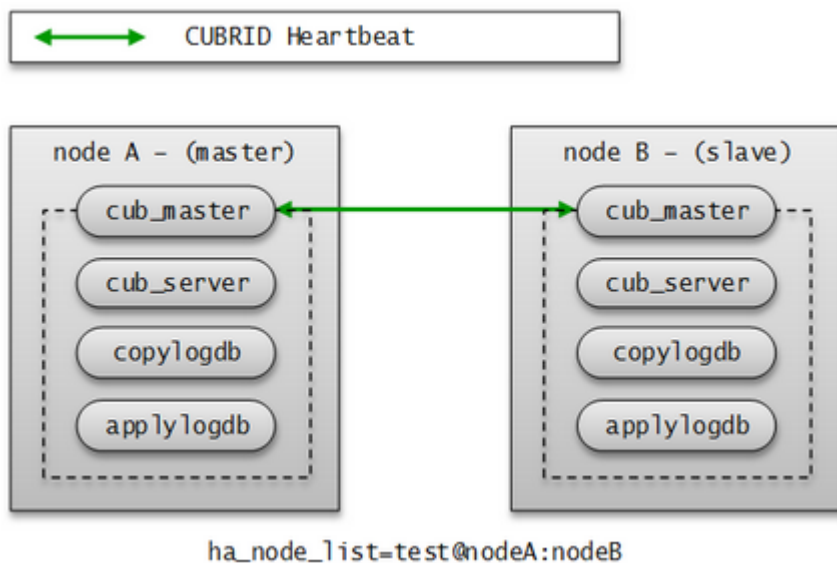
- cub\_server 프로세스의 로그 정보는 **\$CUBRID/log/server/<db\_name>\_<date>\_<time>.err** 파일에 저장되며 다음의 내용이 기록된다.

```
Server HA mode is changed from 'to-be-active' to 'active'.
```

## heartbeat 메시지

HA 기능을 제공하기 위한 핵심 구성 요소로, 마스터 노드, 슬레이브 노드, 레플리카 노드가 다른 노드의 상태를 감시하기 위해 주고 받는 메시지이다. 마스터 프로세스는 그룹 내의 모든 마스터 프로세스와 주기적으로 heartbeat 메시지를 주고 받는다. heartbeat 메시지는 **cubrid\_ha.conf**의 **ha\_port\_id** 파라미터에 설정된 UDP 포트로 주고 받는다. heartbeat 메시지 주기는 내부적으로 설정된 값을 따른다.

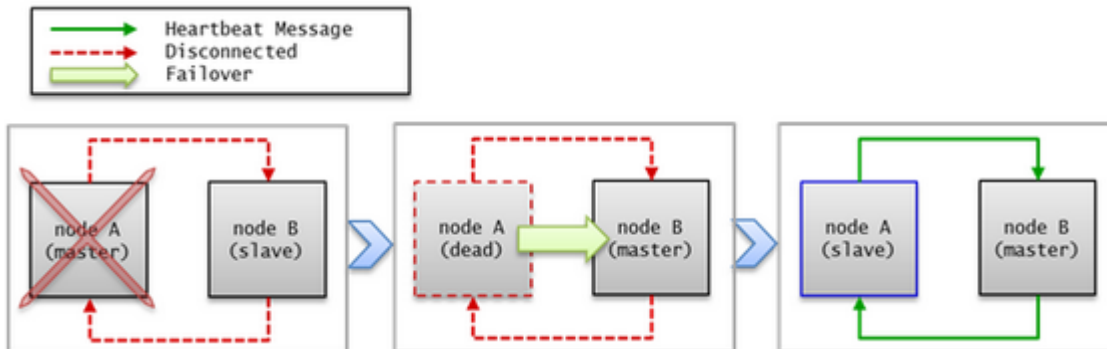
마스터 노드의 장애가 감지되면 슬레이브 노드로 failover가 이루어진다.



## failover와 failback

failover란, 마스터 노드에 장애가 발생하여 서비스를 제공할 수 없는 상태가 되면 우선순위가 가장 높은 슬레이브 노드가 자동으로 마스터 노드가 되는 것이다. 마스터 프로세스는 수집한 CUBRID HA 그룹 내의 노드들의 정보를 바탕으로 스코어를 계산하여 적절한 시점에 해당 프로세스가 속한 노드의 상태를 마스터 노드로 변경하고, 관리 프로세스에 변경된 상태를 전파한다.

failback은 마스터 노드였던 장애 노드가 복구되면 자동으로 다시 마스터 노드가 되는 것이며, CUBRID HA는 서버의 failback을 지원하지 않는다.



heartbeat 메시지가 정상적으로 전달되지 않으면 failover가 일어나므로, 네트워크가 불안정한 환경에서는 장애가 발생하지 않아도 failover가 일어날 수 있다. 이와 같은 상황에서 failover가 일어나는 것을 막기 위해 **ha\_ping\_hosts** 를 설정할 수 있다. **ha\_ping\_hosts** 를 설정하면, heartbeat 메시지가 정상적으로 전달되지 못했을 때 **ha\_ping\_hosts** 로 설정한 노드로 ping 메시지를 보내서 원인이 네트워크 불안정인지 확인하는 절차를 거친다. **ha\_ping\_hosts** 설정에 대한 좀 더 자세한 설명은 [cubrid\\_ha.conf](#) 를 참고한다.

## 브로커 모드

브로커는 DB 서버에 **Read Write, Read Only, Standby Only** 이렇게 세 가지 모드 중 한 가지로 접속할 수 있으며, 사용자가 브로커 모드를 설정할 수 있다.

브로커는 DB 서버 연결 순서에 의해 연결을 시도하여 자신의 모드에 맞는 DB 서버를 선택하여 연결한다. 조건이 맞지 않아 연결되지 않으면 다음 순서의 연결을 시도하고, 모든 순서를 수행해도 적절한 DB 서버를 찾지 못하면 해당 브로커는 DB 서버 연결에 실패한다.

브로커 모드 설정 방법은 [cubrid\\_broker.conf](#)를 참고한다.

DB 서버 연결은 [cubrid\\_broker.conf](#)의 **PREFERRED\_HOSTS**, **CONNECT\_ORDER**와 **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP** 파라미터의 영향을 받는다. 이들에 의한 영향은 [브로커와 DB 연결](#)을 참고한다.

다음은 위의 파라미터들을 설정하지 않은 경우에 대한 설명이다.

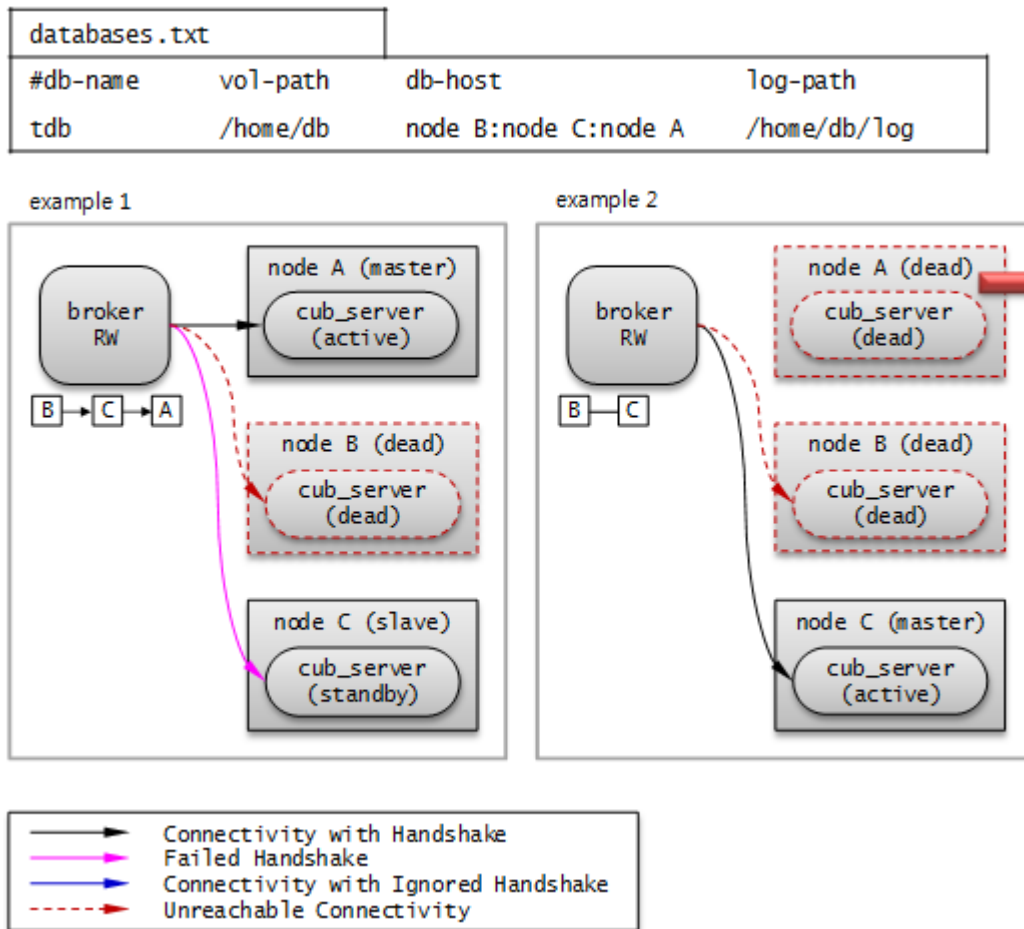
### Read Write

“ACCESS\_MODE=RW”

읽기, 쓰기 서비스를 제공하는 브로커이다. 이 브로커는 일반적으로 액티브 서버에 연결하며, 연결 가능한 액티브 서버가 없으면 일시적으로 스탠바이 서버에 연결한다. 따라서 Read Write 브로커는 일시적으로 스탠바이 서버와 연결될 수 있다.

일시적으로 스탠바이 서버와 연결되면 트랜잭션이 끝날 때마다 스탠바이 서버와 연결을 끊고, 다음 트랜잭션이 시작되면 다시 액티브 서버와 연결을 시도한다. 스탠바이 서버와 연결되면 읽기 서비스만 가능하며, 쓰기 요청에 대해서는 서버에서 오류가 발생한다.

다음 그림은 **db-host** 설정을 통해 호스트에 연결하는 예이다.



databases.txt의 db-host가 node B:node C:node A 순이므로, B, C, A 순으로 접속을 시도한다. 이때 db-host에 명시된 “node B:node C:node A”는 /etc/hosts 파일에 정의된 실제 호스트 이름이다.

- Example 1. node B 는 비정상 종료된 상태이고, node C 는 standby 상태이며, node A 는 active 상태이다. 따라서 최종적으로 node A 와 연결한다.
- Example 2. node B 는 비정상 종료된 상태이고, node C 는 active 상태이다. 따라서 최종적으로 node C 와 연결한다.

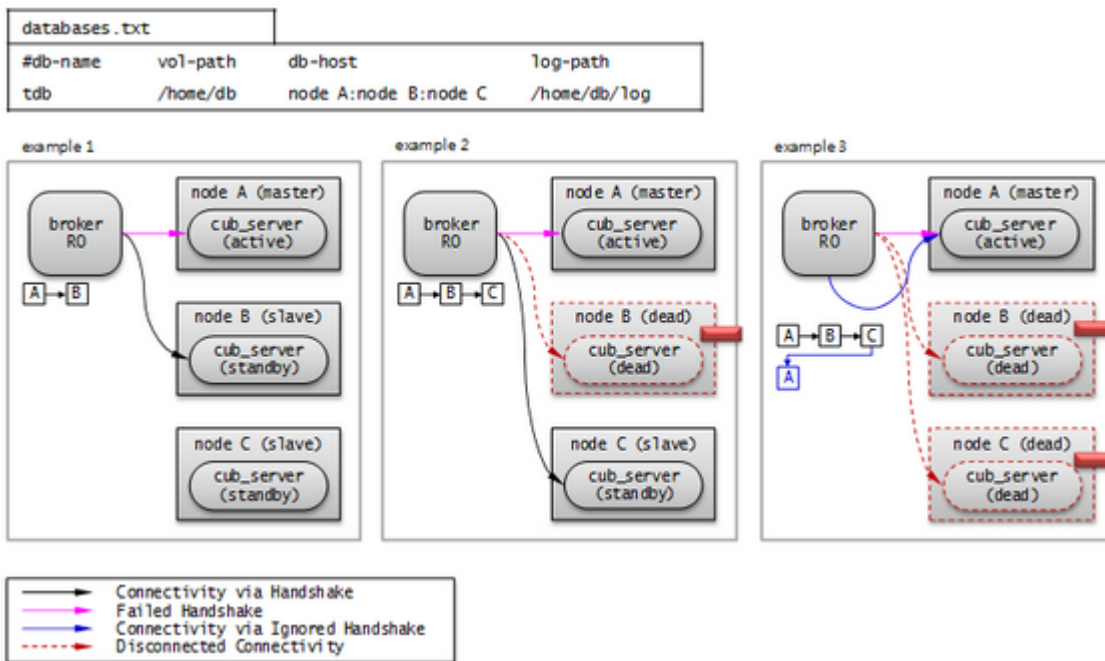
### Read Only

“ACCESS\_MODE=RO”

읽기 서비스를 제공하는 브로커이다. 이 브로커는 가능한 스탠바이 서버에 연결하며, 스탠바이 서버가 없으면 액티브 서버에 연결한다. 따라서 Read Only 브로커는 일시적으로 액티브 서버와 연결될 수 있다.

액티브 서버와 연결된 후 **RECONNECT\_TIME** 설정 시간이 지나면 연결을 끊고 재연결을 시도한다. 또는 **cubrid broker reset** 명령을 실행하여 기존 연결을 끊고 새롭게 스탠바이 서버에 연결할 수 있다. Read Only 브로커에 쓰기 요청이 전달되면 브로커에서 오류가 발생하므로, 액티브 서버와 연결되어도 읽기 서비스만 가능하다.

다음 그림은 **db-host** 설정을 통해 호스트에 연결하는 예이다.



databases.txt의 db-host가 node A:node B:node C 순이므로, A, B, C 순으로 접속을 시도한다. 이때 db-host에 명시된 “node A:node B:node C”는 /etc/hosts 파일에 정의된 실제 호스트 이름이다.

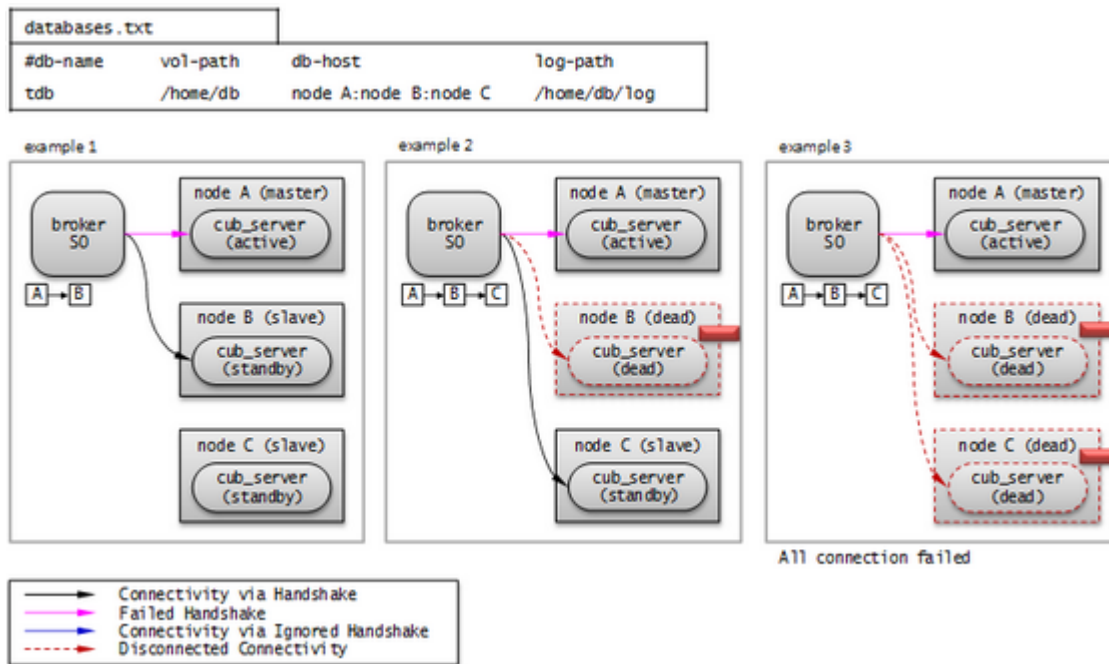
- Example 1. node A 는 active 상태이고, node B 는 standby 상태이다. 따라서 최종적으로 node B 와 연결된다.
- Example 2. node A 는 active 상태이고, node B 는 비정상 종료된 상태이며, node C 는 standby 상태이다. 따라서 최종적으로 node C 와 연결된다.
- Example 3. node A 는 active 상태이고, node B 와 node C 는 비정상 종료된 상태이다. 따라서 최종적으로 node A 와 연결된다.

### Standby Only

“ACCESS\_MODE=SO”

읽기 서비스를 제공하는 브로커이다. 이 브로커는 스탠바이 서버에 연결하며, 스탠바이 서버가 없으면 서비스를 제공하지 않는다.

다음 그림은 **db-host** 설정을 통해 호스트에 연결하는 예이다.



databases.txt의 db-host가 node A:node B:node C 순이므로, A, B, C 순으로 접속을 시도한다. 이때 db-host에 명시된 “node A:node B:node C”는 /etc/hosts 파일에 정의된 실제 호스트 이름이다.

- Example 1. node A 는 active 상태이고, node B 는 standby 상태이다. 따라서 최종적으로 node B 와 연결된다.
- Example 2. node A 는 active 상태이고, node B 는 비정상 종료된 상태이며, node C 는 standby 상태이다. 따라서 최종적으로 node C 와 연결된다.
- Example 3. node A 는 active 상태이고, node B 와 node C 는 비정상 종료된 상태이다. 따라서 최종적으로 어떤 노드와도 연결되지 않는다. 이 부분이 Read Only 브로커와의 차이점이다.

## CUBRID HA 기능

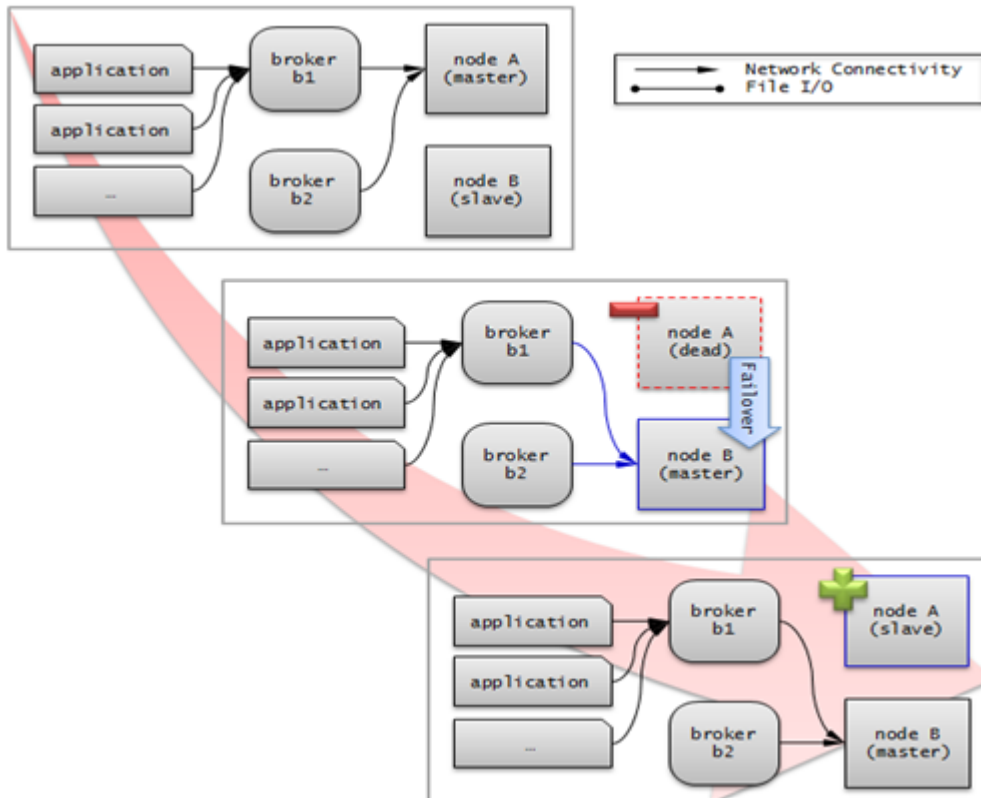
### 서버 이중화

서버 이중화란 CUBRID HA 기능을 제공하기 위해 물리적인 하드웨어 장비를 중복으로 구성하여 시스템을 구축하는 것이다. 이러한 구성을 통해 하나의 장비에 장애가 발생해도 응용 프로그램에서는 지속적인 서비스를 제공할 수 있다.

### 서버 failover

브로커는 서버의 접속 순서를 정의하고 그 순서에 따라 서버에 접속한다. 접속한 서버에 장애가 발생하면 브로커는 다음 순위로 설정된 서버에 접속하며, 응용 프로그램에서는 별도의 처리가 필요 없다. 브로커가 다음 서버에 접속할 때의 동작은 브로커의 모드에 따라 다를 수 있다. 서버의 접속 순서 및 브로커 모드의 설정 방법은 [cubrid\\_broker.conf](#)를 참고한다.





## 서버 failback

CUBRID HA는 자동으로 서버 failback을 지원하지 않는다. 따라서 failback을 수동으로 적용하려면 비정상 종료되었던 마스터 노드를 복구하여 슬레이브 노드로 구동한 후, failover로 인해 슬레이브에서 마스터로 역할이 바뀐 노드를 의도적으로 종료하여 다시 각 노드의 역할을 서로 바꾼다.

예를 들어 *nodeA*가 마스터, *nodeB*가 슬레이브일 때 failover 이후에는 역할이 바뀌어 *nodeB*가 마스터, *nodeA*가 슬레이브가 된다. *nodeB*를 종료(**cubrid heartbeat stop**)한 후, *nodeA*가 마스터, 즉 노드 상태가 active로 바뀌었는지 확인(**cubrid heartbeat status**)한다. 그리고 나서 *nodeB*를 시작(**cubrid heartbeat start**)하면, *nodeB*는 슬레이브가 된다.

## 브로커 이중화

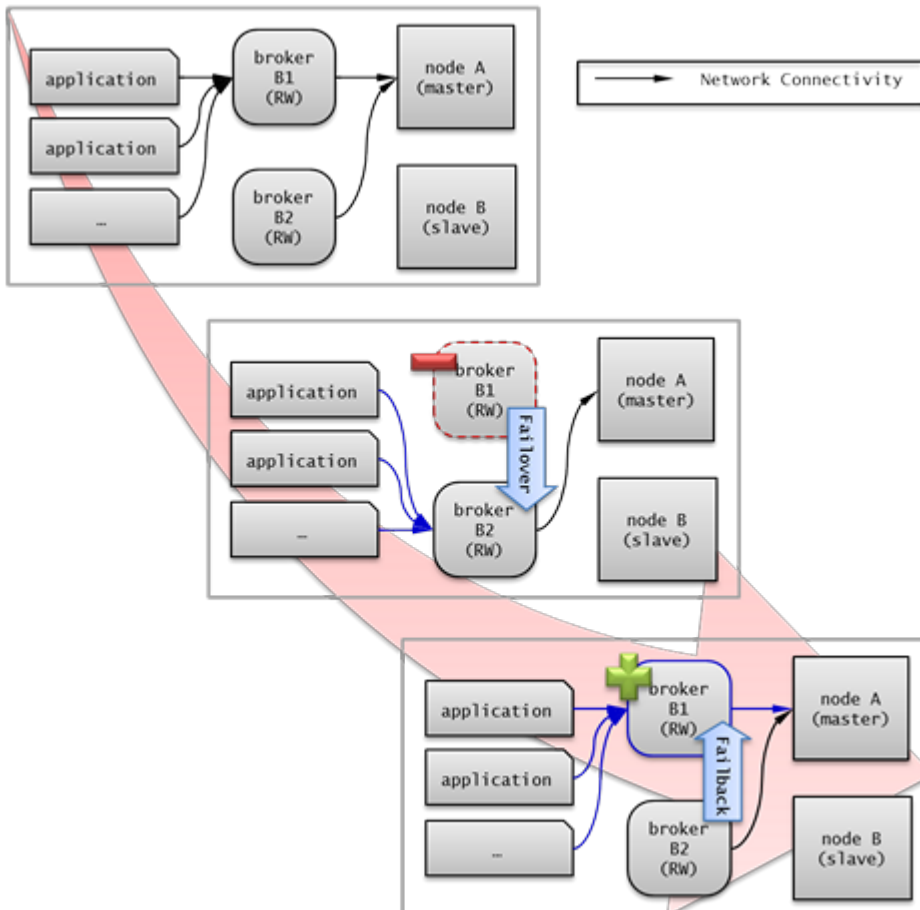
CUBRID는 3-tier DBMS로, 응용 프로그램과 데이터베이스 서버를 중계하는 역할을 수행하는 브로커라는 미들웨어가 있다. CUBRID HA 기능을 제공하기 위해 브로커도 물리적인 하드웨어를 중복으로 구성하여, 하나의 브로커에 장애가 발생해도 응용 프로그램에서는 지속적인 서비스를 제공할 수 있다.

브로커 이중화의 구성은 서버 이중화의 구성에 따라 결정되는 것이 아니며, 사용자의 선호에 맞게 변형이 가능하다. 또한, 별도의 장비로 분리가 가능하다.

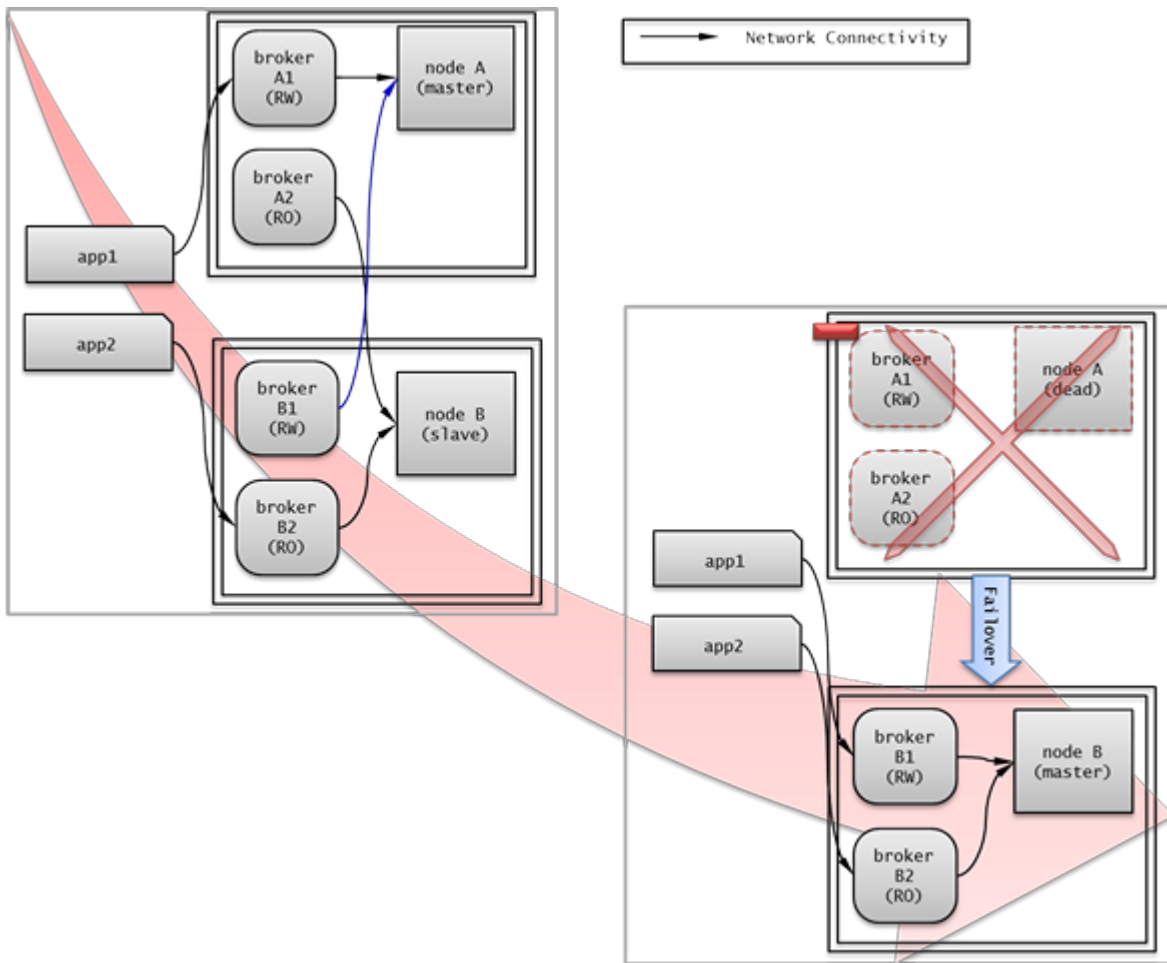
브로커의 failover, failback 기능을 사용하려면 JDBC, CCI 또는 PHP의 접속 URL에 **altHosts** 속성을 추가해야 한다. 이에 대한 설명은 JDBC 설정, CCI 설정 또는 PHP 설정을 참고한다.

브로커를 설정하려면 **cubrid\_broker.conf** 파일을 설정해야 하고, 데이터베이스 서버의 failover 순서를 설정하려면 **databases.txt** 파일을 설정해야 한다. 이에 대한 설명은 **브로커 설정, 시작 및 확인**을 참고한다.

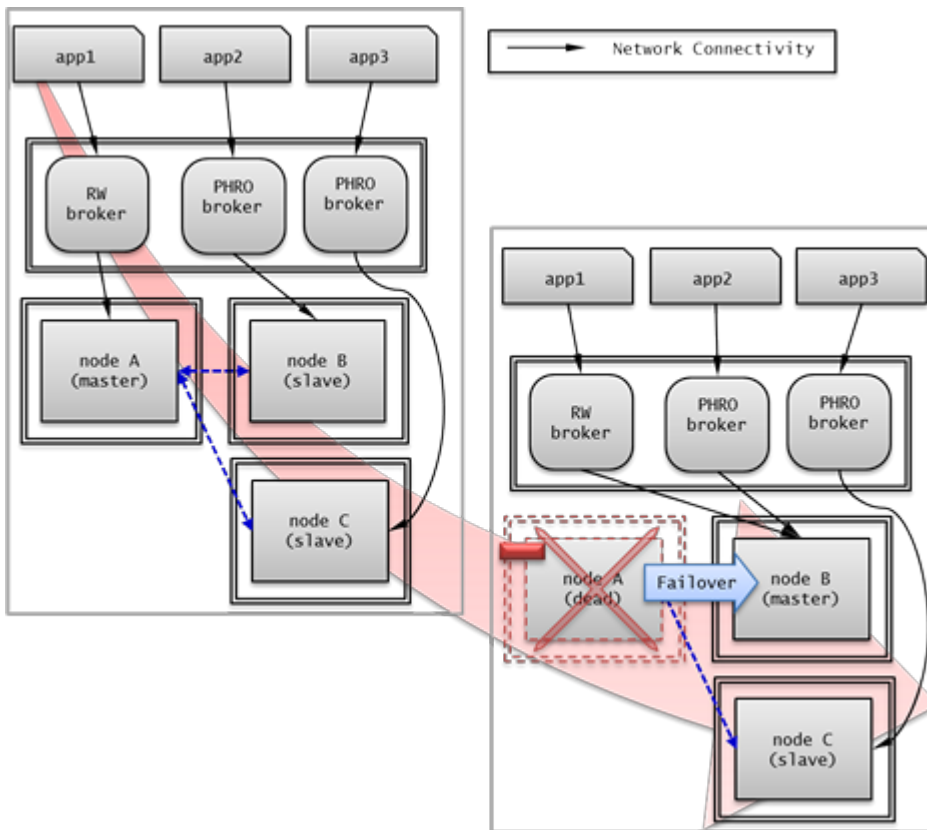
다음은 2개의 Read Write(RW) 브로커를 구성한 예이다. application URL의 첫 번째 접속 브로커를 *broker B1* 으로 하고 두 번째 접속 브로커를 *broker B2* 로 설정하면, application이 *broker B1* 에 접속할 수 없는 경우 *broker B2* 에 접속하게 된다. 이후 *broker B1* 이 다시 접속 가능해지면 application은 *broker B1* 에 재접속하게 된다.



다음은 마스터 노드, 슬레이브 노드의 각 장비 내에 Read Write(RW) 브로커와 Read Only(RO) 브로커를 구성한 예이다. *app1*과 *app2* URL의 첫 번째 접속은 각각 *broker A1 (RW)*, *broker B2 (RO)* 이고, 두 번째 접속(**altHosts**)은 각각 *broker A2 (RO)*, *broker B1 (RW)*이다. *nodeA* 를 포함한 장비가 고장나면, *app1*과 *app2*는 *nodeB*를 포함한 장비의 브로커에 접속한다.



다음은 브로커 장비를 별도로 구성하여 Read Write 브로커 한 개, Preferred Host Read Only 브로커 두 개를 두고, 한 개의 마스터 노드와 두 개의 슬레이브 노드를 구성한 예이다. Preferred Host Read Only 브로커들은 각각 *nodeB* 와 *nodeC* 에 연결함으로써 읽기 부하를 분산하였다.



### 브로커 failover

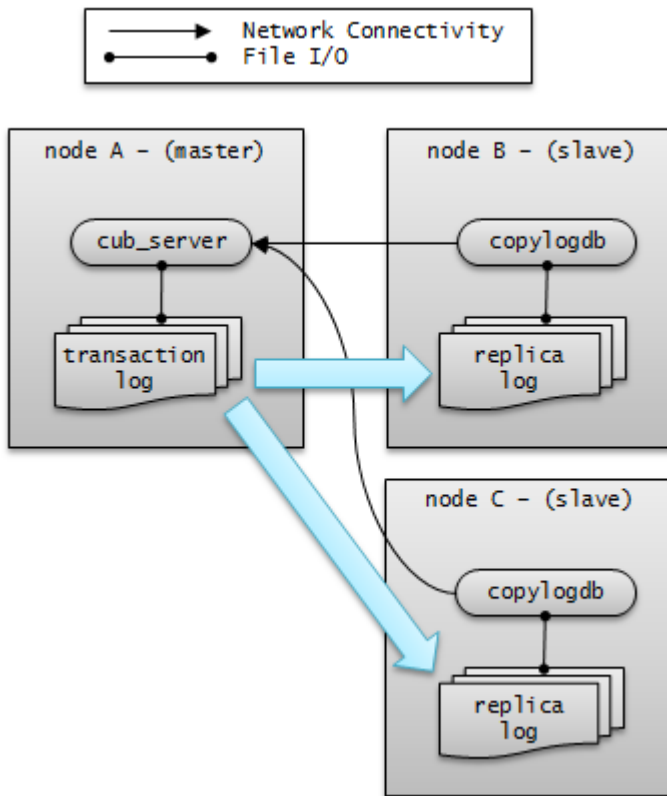
브로커 failover는 시스템 파라미터의 설정에 의해 자동으로 failover되는 것이 아니며, JDBC, CCI, PHP 응용 프로그램에서는 접속 URL의 **altHosts**에 브로커 호스트들을 설정해야 브로커 failover가 가능하다. 설정한 우선순위가 가장 높은 브로커에 접속하고, 접속한 브로커에 장애가 발생하면 접속 URL에 다음 순위로 설정한 브로커에 접속한다. 응용 프로그램에서는 접속 URL의 **altHosts**를 설정하는 것 외에는 별도의 처리가 필요 없으며, JDBC, CCI, PHP 드라이버 내부에서 처리한다.

### 브로커 fallback

브로커 failover 이후 장애 브로커가 복구되면 기존 브로커와 접속을 끊고 이전에 연결했던 우선순위가 가장 높은 브로커에 다시 접속한다. 응용 프로그램에서는 별도의 처리가 필요 없으며, JDBC, CCI, PHP 드라이버 내부에서 처리한다. 브로커 fallback을 수행하는 시간은 JDBC 접속 URL에 설정한 값을 따른다. 이에 대한 설명은 [JDBC 설정](#)을 참고한다.

## 로그 다중화

CUBRID HA는 CUBRID HA 그룹에 포함된 모든 노드에 트랜잭션 로그를 복사하고 이를 반영함으로써 CUBRID HA 그룹 내의 모든 노드를 동일한 DB로 유지한다. CUBRID HA의 로그 복사 구조는 마스터 노드와 슬레이브 노드 사이의 상호 복사 형태로, 전체 로그의 양이 많아지는 단점이 있으나 체인 형태의 복사 구조보다 구성 및 장애 처리 측면에서 유연하다는 장점이 있다.



트랜잭션 로그를 복사하는 모드는 **SYNC**, **ASYNC**의 두 가지가 있으며, 사용자가 `cubrid_ha.conf`로 설정할 수 있다.

### SYNC 모드

트랜잭션이 커밋되면, 발생한 트랜잭션 로그가 슬레이브 노드에 복사되어 파일에 저장되고 이에 대한 성공 여부를 전달받은 후에 트랜잭션 커밋이 완료된다. 따라서 **ASYNC** 모드에 비해 커밋 수행 시간이 길어질 수 있지만, failover가 발생해도 복사된 트랜잭션 로그는 스탠바이 서버에 반영되어 있음을 보장할 수 있으므로 가장 안전하다.

### ASYNC 모드

트랜잭션이 커밋되면, 슬레이브 노드로 트랜잭션 로그가 전송 완료되었는지 확인하지 않고 커밋이 완료된다. 따라서 마스터 노드에서 커밋이 완료된 트랜잭션이 슬레이브 노드에 반영되지 못하는 경우가 발생할 수 있다.

**ASYNC** 모드는 로그 복제로 인한 커밋 수행 시간 지연은 거의 없으므로 성능상 유리하지만, 노드 간의 데이터가 완전히 일치하지 않을 수 있다.

### 참고

**SEMISSYNC** 모드는 사용이 중단될 예정(deprecated)이며, 현재 **SYNC** 모드와 동일하게 동작한다.

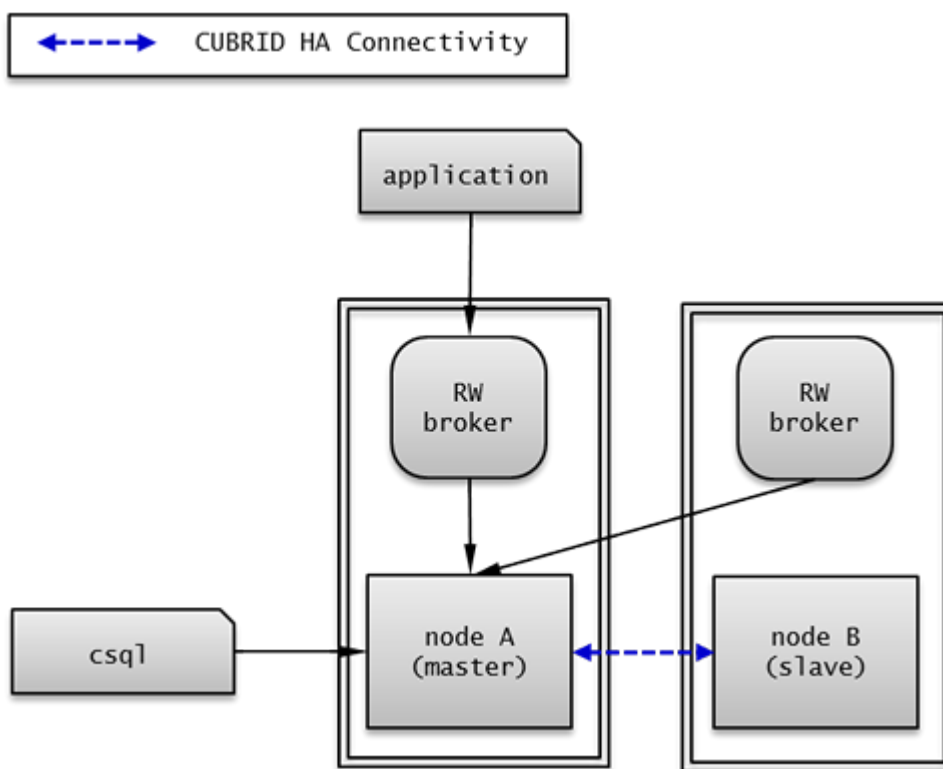
## 빠른 시작

DB 생성 시점부터 마스터 노드와 슬레이브 노드를 1:1로 구축하는 방법에 대해 간단히 설명한다. 다양한 복제 구축 방법에 대한 자세한 방법은 [복제 구축](#)을 참고한다.

## 준비

### 구성도

CUBRID HA를 처음 접하는 사용자가 CUBRID HA를 쉽게 사용할 수 있도록 아래 그림과 같이 간단하게 구성된 CUBRID HA를 설정하는 과정을 설명한다.



### 사양

마스터 노드와 슬레이브 노드로 사용할 장비에는 Linux와 CUBRID 2008 R2.2 이상 버전이 설치되어 있어야 한다. CUBRID HA는 Windows를 지원하지 않는다.

### CUBRID HA 구성 장비 사양

|            | CUBRID 버전           | OS    |
|------------|---------------------|-------|
| 마스터 노드용 장비 | CUBRID 2008 R2.2 이상 | Linux |

|             | CUBRID 버전           | OS    |
|-------------|---------------------|-------|
| 슬레이브 노드용 장비 | CUBRID 2008 R2.2 이상 | Linux |

### 참고

이 문서는 9.2 이상 버전의 HA 구성에 대해 설명하고 있으며, 그 이전 버전과는 설정 방법이 조금 다르므로 주의한다. 예를 들어, **cubrid\_ha.conf** 는 2008 R4.0 이상 버전에서 도입되었다. **ha\_make\_slavedb.sh**는 2008 R4.1 Patch 2 이상 버전에서 도입되었다.

## 데이터베이스 생성 및 서버 설정

### 데이터베이스 생성

CUBRID HA에 포함할 데이터베이스를 모든 CUBRID HA 노드에서 동일하게 생성한다. 데이터베이스 생성 옵션은 필요에 따라 적절히 변경한다.

```
[nodeA]$ cd $CUBRID_DATABASES
[nodeA]$ mkdir testdb
[nodeA]$ cd testdb
[nodeA]$ mkdir log
[nodeA]$ cubrid createdb -L ./log testdb en_US
Creating database with 512.0M size. The total amount of disk space
needed is 1.5G.

CUBRID 10.0

[nodeA]$
```

### cubrid.conf

**\$CUBRID/conf/cubrid.conf** 의 **ha\_mode** 를 모든 HA 노드에 동일하게 설정한다. 특히, 로깅 관련 파라미터인 **log\_max\_archives** 와 **force\_remove\_log\_archives**, HA 관련 파라미터인 **ha\_mode** 의 설정에 주의한다.

```
Service parameters
[service]
service=server,broker,manager

Common section
```

```
[common]
service=server,broker,manager

Server parameters
server=testdb
data_buffer_size=512M
log_buffer_size=4M
sort_buffer_size=2M
max_clients=100
cubrid_port_id=1523
db_volume_size=512M
log_volume_size=512M

HA 구성 시 추가 (Logging parameters)
log_max_archives=100
force_remove_log_archives=no

HA 구성 시 추가 (HA 모드)
ha_mode=on
```

### cubrid\_ha.conf

**\$CUBRID/conf/cubrid\_ha.conf** 의 **ha\_port\_id**, **ha\_node\_list**, **ha\_db\_list** 를 모든 HA 노드에 동일하게 설정한다. 다음 예에서 마스터 노드의 호스트 이름은 *nodeA*, 슬레이브 노드의 호스트 이름은 *nodeB*라고 가정한다.

```
[common]
ha_port_id=59901
ha_node_list=cubrid@nodeA:nodeB
ha_db_list=testdb
ha_copy_sync_mode=sync:sync
ha_apply_max_mem_size=500
```

### databases.txt

**\$CUBRID\_DATABASES/databases.txt** (**\$CUBRID\_DATABASES** 가 설정 안 된 경우 **\$CUBRID/databases/databases.txt**)의 db-host에 마스터 노드와 슬레이브 노드의 호스트 이름을 설정 (*nodeA:nodeB*)한다.

```
#db-name vol-path db-host log-path lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /home/cubrid/DB/testdb/log
file:/home/cubrid/DB/testdb/lob
```

## CUBRID HA 시작 및 확인

### CUBRID HA 시작



CUBRID HA 그룹 내의 각 노드에서 **cubrid heartbeat start**를 수행한다. **cubrid heartbeat start** 를 가장 먼저 수행한 노드가 마스터 노드가 되므로 유의해야 한다. 이하의 예에서 마스터 노드의 호스트 이름은 *nodeA*, 슬레이브 노드의 호스트 이름은 *nodeB*라고 가정한다.

- 마스터 노드

```
[nodeA]$ cubrid heartbeat start
```

- 슬레이브 노드

```
[nodeB]$ cubrid heartbeat start
```

## CUBRID HA 상태 확인

CUBRID HA 그룹 내의 각 노드에서 **cubrid heartbeat status**를 수행하여 구성 상태를 확인한다.

```
[cubrid@nodeA]$ cubrid heartbeat status
@cubrid heartbeat list
HA-Node Info (current nodeA, state master)
 Node nodeB-node-name (priority 2, state slave)
 Node nodeA-node-name (priority 1, state master)
HA-Process Info (nodeA 9289, state nodeA)
 Applylogdb testdb@localhost:/home1/cubrid1/DB/testdb_nodeB (pid
9423, state registered)
 Copylogdb testdb@nodeB-node-name:/home1/cubrid1/DB/testdb_nodeB
(pid 9418, state registered)
 Server testdb (pid 9306, state registered_and_active)

[nodeA]$
```

CUBRID HA 그룹 내의 각 노드에서 **cubrid changemode** 유틸리티를 이용하여 서버의 상태를 확인한다.

- 마스터 노드

```
[nodeA]$ cubrid changemode testdb@localhost
The server 'testdb@localhost''s current HA running mode is active.
```

- 슬레이브 노드

```
[nodeB]$ cubrid changemode testdb@localhost
The server 'testdb@localhost''s current HA running mode is standby.
```

## CUBRID HA 동작 여부 확인

마스터 노드의 액티브 서버에서 쓰기를 수행한 후 슬레이브 노드의 스탠바이 서버에 정상적으로 반영되었는지 확인한다. HA 환경에서 CSQL 인터프리터로 각 노드에 접속하려면, 데이터베이스 이름 뒤에 접속 대상 호스트 이름을 반드시 지정해야 한다("@<호스트 이름>"). 호스트 이름을 localhost 로 지정하면, 로컬 노드에 접속하게 된다.

### ⚠ 경고

복제가 정상적으로 수행되기 위해서는 테이블을 생성할 때 기본키(primary key)가 반드시 존재해야 한다는 점을 주의한다

#### · 마스터 노드

```
[nodeA]$ csql -u dba testdb@localhost -c "create table abc(a int, b
int, c int, primary key(a));"
[nodeA]$ csql -u dba testdb@localhost -c "insert into abc values
(1,1,1);"
[nodeA]$
```

#### · 슬레이브 노드

```
[nodeB]$ csql -u dba testdb@localhost -l -c "select * from abc;"
=== <Result of SELECT Command in Line 1> ===
<00001> a: 1
 b: 1
 c: 1
[nodeB]$
```

## 브로커 설정, 시작 및 확인

### 브로커 설정

데이터베이스 failover 시 정상적인 서비스를 위해서 **databases.txt** 의 **db-host** 항목에 데이터베이스의 가용 노드를 설정해야 한다. 그리고 **cubrid\_broker.conf** 의 **ACCESS\_MODE** 를 설정하는데, 이를 생략하면 기본값인 Read Write 모드로 설정된다. 브로커를 별도의 장비로 분리하는 경우 브로커 장비에 **cubrid\_broker.conf** 와 **databases.txt** 를 반드시 설정해야 한다.

#### · databases.txt

```
#db-name vol-path db-host log-
path lob-base-path
testdb /home1/cubrid1/CUBRID/testdb nodeA:nodeB /
```

```
home1/cubrid1/CUBRID/testdb/log file:/home1/cubrid1/CUBRID/testdb/lob
```

• cubrid\_broker.conf

```
[%testdb_RWbroker]
SERVICE =ON
BROKER_PORT =33000
MIN_NUM_APPL_SERVER =5
MAX_NUM_APPL_SERVER =40
APPL_SERVER_SHM_ID =33000
LOG_DIR =log/broker/sql_log
ERROR_LOG_DIR =log/broker/error_log
SQL_LOG =ON
TIME_TO_KILL =120
SESSION_TIMEOUT =300
KEEP_CONNECTION =AUTO
CCI_DEFAULT_AUTOCOMMIT =ON

broker mode parameter
ACCESS_MODE =RW
```

### 브로커 시작 및 상태 확인

브로커는 JDBC나 CCI, PHP 등의 응용에서 접근하기 위해 사용하는 것이다. 따라서 간단한 서버 이중화 동작을 시험하고 싶다면 브로커를 시작할 필요 없이 서버 프로세스에 직접 접속하는 CSQL 인터프리터만 실행해서 확인할 수 있다. 브로커는 **cubrid broker start** 를 실행하여 시작하고 **cubrid broker stop** 을 실행하여 정지한다.

다음은 브로커를 마스터 노드에서 실행한 예이다.

```
[nodeA]$ cubrid broker start
@ cubrid broker start
++ cubrid broker start: success
[nodeA]$ cubrid broker status
@ cubrid broker status
% testdb_RWbroker

ID PID QPS LQS PSIZE STATUS

1 9532 0 0 48120 IDLE
```

### 응용 프로그램 설정

응용 프로그램이 연결할 브로커의 호스트 이름(*nodeA\_broker*, *nodeB\_broker*)과 포트를 연결 URL에 명시한다. 브로커와의 연결 장애가 발생한 경우 다음으로 연결을 시도할 브로커는 **altHosts** 속성에

명시한다. 아래는 JDBC 프로그램의 예이며, CCI, PHP에 대한 예와 자세한 설명은 [CCI 설정](#), [PHP 설정](#)을 참고한다.

```
Connection connection =
DriverManager.getConnection("jdbc:CUBRID:nodeA_broker:
33000:testdb:::charset=utf-8&altHosts=nodeB_broker:33000", "dba",
"");
```

## 환경 설정

다음은 CUBRID HA 기능을 수행하기 위해 필요한 환경 설정에 대한 설명이다. 브로커와 DB 사이의 연결 절차에 대한 자세한 설명은 [브로커와 DB 연결](#)을 참고한다.

### cubrid.conf

**cubrid.conf** 파일은 **\$CUBRID/conf** 디렉터리에 위치하며, CUBRID의 전반적인 설정 정보를 담고 있다. 여기에서는 **cubrid.conf** 중 CUBRID HA가 사용하는 파라미터를 설명한다.

### HA 여부

#### ha\_mode

CUBRID HA 기능을 설정하는 파라미터이다. 기본값은 **off** 이다. CUBRID HA 기능은 Windows를 지원하지 않고 Linux에서만 사용할 수 있으므로 이 값은 Linux용 CUBRID에서만 의미가 있다.

- **off** : CUBRID HA 기능을 사용하지 않는다.
- **on** : CUBRID HA 기능을 사용하며, 해당 노드는 failover의 대상이 된다.
- **replica** : CUBRID HA 기능을 사용하며, 해당 노드는 failover의 대상이 되지 않는다.

**ha\_mode** 가 **on** 이면 **cubrid\_ha.conf** 를 읽어 CUBRID HA를 설정한다.

이 파라미터는 동적으로 변경할 수 없으며, 변경하면 해당 노드를 다시 시작해야 한다.

### 로깅

#### log\_max\_archives

보존할 보관 로그 파일의 최소 개수를 설정하는 파라미터이다. 최소값은 0이며 기본값은 **INT\_MAX** (2147483647)이다. CUBRID 설치 시 **cubrid.conf**에는 0으로 설정되어 있다. 이 파라미터의 동작은 **force\_remove\_log\_archives**의 영향을 받는다.

**force\_remove\_log\_archives**의 설정값이 **no**이면, 활성화된 트랜잭션이 참조하고 있는 기존 보관 로그 파일 또는 HA 환경에서 슬레이브 노드에 반영되지 않은 마스터 노드의 보관 로그 파일이 삭제되지 않는다. 이에 대한 자세한 내용은 아래의 **force\_remove\_log\_archives**를 참고한다.

**log\_max\_archives**에 대한 자세한 내용은 **로깅 관련 파라미터**를 참고한다.

### force\_remove\_log\_archives

**ha\_mode** 를 on으로 설정하여 HA 환경을 구축하려면 **force\_remove\_log\_archives** 를 no로 설정하여 HA 관련 프로세스에 의해 사용할 보관 로그(archive log)를 항상 유지하는 것을 권장한다.

**force\_remove\_log\_archives**를 yes로 설정하면 HA 관련 프로세스가 사용할 보관 로그 파일까지 삭제될 수 있고, 이로 인해 데이터베이스 복제 노드 간 데이터 불일치가 발생할 수 있다. 이러한 위험성을 감수하더라도 디스크의 여유 공간을 유지하고 싶다면 **force\_remove\_log\_archives**를 yes로 설정한다.

**force\_remove\_log\_archives**에 대한 자세한 내용은 **로깅 관련 파라미터**를 참고한다.

#### 참고

2008 R4.3 버전부터, 레플리카 노드에서는 **force\_remove\_log\_archives** 값의 설정과 무관하게 **log\_max\_archives** 파라미터에 설정된 개수의 보관 로그 파일을 제외하고는 항상 삭제한다.

## 접속

### max\_clients

데이터베이스 서버에 동시에 연결할 수 있는 클라이언트의 최대 수를 지정하는 파라미터이다. 기본값은 **100**이다.

CUBRID HA 기능을 사용하면 기본적으로 복제 로그 복사 프로세스와 복제 로그 반영 프로세스가 구동되므로, 해당 노드를 제외한 CUBRID HA 그룹 내 노드 수의 두 배를 고려하여 설정해야 한다. 또한 failover가 일어날 때 다른 노드에 접속하고 있던 클라이언트가 해당 노드에 접속할 수 있으므로 이를 고려해야 한다.

**max\_clients**에 대한 자세한 내용은 **접속 관련 파라미터**를 참고한다.

### 노드 간 반드시 값이 동일해야 하는 시스템 파라미터

- **log\_buffer\_size** : 로그 버퍼 크기. 서버와 로그를 복사하는 **copylogdb** 간 프로토콜에 영향을 주는 부분이므로 반드시 동일해야 한다.
- **log\_volume\_size** : 로그 볼륨 크기. CUBRID HA는 원본 트랜잭션 로그와 복제 로그의 형태와 내용이 동일하므로 반드시 동일해야 한다. 그 외 각 노드에서 별도로 DB를 생성하는 경우 **cubrid**

**createdb** 옵션(--db-volume-size, --db-page-size, --log-volume-size, --log-page-size 등)이 동일해야 한다.

- **cubrid\_port\_id** : 서버와의 연결 생성을 위한 TCP 포트 번호. 서버와 로그를 복사하는 **copylogdb**의 연결을 위해 반드시 동일해야 한다.
- **HA 관련 파라미터** : **cubrid\_ha.conf**에 포함된 HA 관련 파라미터는 기본적으로 동일해야 하나, 아래 파라미터는 예외적으로 다르게 설정할 수 있다.

노드에 따라 다르게 설정할 수 있는 파라미터

- 레플리카 노드의 **ha\_mode** 파라미터
- **ha\_copy\_sync\_mode** 파라미터
- **ha\_ping\_hosts** 파라미터
- **ha\_tcp\_ping\_hosts** 파라미터

예시

다음은 **cubrid.conf** 설정의 예이다. 특히, 로깅 관련 파라미터인 **log\_max\_archives** 와 **force\_remove\_log\_archives**, HA 관련 파라미터인 **ha\_mode** 의 설정에 주의한다.

```
Service Parameters
[service]
service=server,broker,manager

Server Parameters
server=testdb
data_buffer_size=512M
log_buffer_size=4M
sort_buffer_size=2M
max_clients=200
cubrid_port_id=1523
db_volume_size=512M
log_volume_size=512M

HA 구성 시 추가 (Logging parameters)
log_max_archives=100
force_remove_log_archives=no

HA 구성 시 추가 (HA 모드)
ha_mode=on
log_max_archives=100
```

## cubrid\_ha.conf

**cubrid\_ha.conf** 파일은 **\$CUBRID/conf** 디렉터리에 위치하며, CUBRID의 HA 기능의 전반적인 설정 정보를 담고 있다. CUBRID HA 기능은 Windows를 지원하지 않고 Linux에서만 사용할 수 있으므로 이 값은 Linux용 CUBRID에서만 의미가 있다.

브로커와 DB 사이의 연결 절차에 대한 자세한 설명은 [브로커와 DB 연결](#)을 참고한다.

## 노드

### ha\_node\_list

CUBRID HA 그룹 내에서 사용할 그룹 이름과 failover의 대상이 되는 멤버 노드들의 호스트 이름을 명시한다. @ 구분자로 나누어 @ 앞이 그룹 이름, @ 뒤가 멤버 노드들의 호스트 이름이다. 여러 개의 호스트 이름은 쉼표(,) 또는 콜론(:)으로 구분한다. 기본값은 **localhost@localhost** 이다.

#### 참고

이 파라미터에서 명시한 멤버 노드들의 호스트 이름은 IP로 대체할 수 없으며, 사용자는 반드시 **/etc/hosts**에 등록되어 있는 것을 사용해야 한다.

호스트 이름이 제대로 설정되지 않은 경우 server.err 에러 로그 파일에 다음과 같은 메시지가 출력된다.

```
Time: 04/10/12 17:49:45.030 - ERROR *** file ../../src/connection/
tcp.c, line 121 ERROR CODE = -353 Tran = 0, CLIENT = (unknown):
(unknown)(-1), EID = 1 Cannot make connection to master server on
host "Wrong_HOST_NAME".... Connection timed out
```

**ha\_mode**를 **on**으로 설정한 노드는 **ha\_node\_list**에 해당 노드가 반드시 포함되어 있어야 한다. CUBRID HA 그룹 내의 모든 노드는 **ha\_node\_list**의 값이 동일해야 한다. failover가 일어날 때 이 파라미터에 설정된 순서에 따라 마스터 노드가 된다.

이 파라미터는 동적으로 변경할 수 있으며, 변경하면 **cubrid heartbeat reload**를 실행해야 한다.

### ha\_replica\_list

CUBRID HA 그룹 내에서 사용할 그룹 이름과 레플리카 노드 이름, 즉 failover의 대상이 되지 않는 멤버 노드들의 호스트 이름을 명시한다. 레플리카 노드를 구성하지 않는 경우에는 지정할 필요가 없다. @ 구분자로 나누어 @ 앞이 그룹 이름, @ 뒤가 멤버 노드들의 호스트 이름이다. 여러 개의 호스트 이름은 쉼표(,) 또는 콜론(:)으로 구분한다. 기본값은 **NULL**이다.

그룹 이름은 **ha\_node\_list**에서 명시한 이름과 같아야 한다. 이 파라미터에서 명시하는 멤버 노드들의 호스트 이름 및 해당 노드의 호스트 이름을 지정할 때는 반드시 **/etc/hosts**에 등록되어 있는 것

을 사용해야 한다. **ha\_mode** 를 **replica** 로 설정한 노드는 **ha\_replica\_list** 에 해당 노드가 반드시 포함되어 있어야 한다. CUBRID HA 그룹 내의 모든 노드는 **ha\_replica\_list** 의 값이 동일해야 한다.

이 파라미터는 동적으로 변경할 수 있으며, 변경하면 **cubrid heartbeat reload**를 실행해야 한다.

#### 참고

이 파라미터에서 명시한 멤버 노드들의 호스트 이름은 IP로 대체할 수 없으며, 사용자는 반드시 **/etc/hosts** 에 등록되어 있는 것을 사용해야 한다.

### ha\_db\_list

CUBRID HA 모드로 구동할 데이터베이스 이름을 명시한다. 기본값은 **NULL** 이다. 여러 개의 데이터베이스 이름은 쉼표(,) 또는 콜론(:)으로 구분한다.

### 접속

#### ha\_port\_id

CUBRID HA 그룹 내의 노드들이 heartbeat 메시지를 주고 받으며 노드 장애를 감지할 때 사용할 UDP 포트 번호를 명시한다. 기본값은 **59901** 이다.

서비스 환경에 방화벽이 있으면, 설정한 포트 값이 방화벽을 통과하도록 방화벽을 설정해야 한다.

#### ha\_ping\_hosts

슬레이브 노드에서 failover가 시작되는 순간 연결을 확인하여 네트워크에 의한 failover인지 확인할 때 사용할 호스트를 명시한다. 기본값은 **NULL** 이다. 여러 개의 호스트 이름은 쉼표(,) 또는 콜론(:)으로 구분한다.

이 파라미터에서 명시한 멤버 노드들의 호스트 이름은 IP로 대체할 수 있으며, 호스트 이름을 사용하는 경우에는 반드시 **/etc/hosts** 에 등록되어 있어야 한다.

CUBRID는 1시간 주기로 **ha\_ping\_hosts**에 명시된 호스트를 점검하여 모든 호스트가 문제 있을 경우 일시적으로 핑 체크(ping check)를 중지하고 5분 단위로 해당 호스트들이 정상화되었는지 검사한다.

이 파라미터를 설정하면 불안정한 네트워크로 인해 상대 마스터 노드가 비정상 종료된 것으로 오인한 슬레이브 노드가 마스터 노드로 역할이 변경되면서 동시에 두 개의 마스터 노드가 존재하게 되는 split-brain 현상을 방지할 수 있다.

그러나 ICMP 프로토콜이 비활성화 되어 있는 경우 핑 체크는 정상적으로 동작하지 않는다. CUBRID는 이 경우를 대비하여 **ha\_tcp\_ping\_hosts**를 지원한다.

이 파라미터는 동적으로 변경할 수 있으며, 변경하면 **cubrid heartbeat reload**를 실행해야 한다.



## ha\_tcp\_ping\_hosts

**ha\_tcp\_ping\_hosts**는 ICMP 프로토콜이 비활성화 되어 **ha\_ping\_hosts**를 사용할 수 없는 경우 대안으로 사용될 수 있다. **ha\_tcp\_ping\_hosts**는 **ha\_ping\_hosts**와 똑같이 동작하지만 핑 체크를 위해 IP 계층 대신 TCP 계층을 사용한다는 차이가 있다. 기본값은 **NULL**이며, 여러 개의 호스트 이름은 쉼표(,)로 호스트 이름과 포트 번호는 콜론(:)으로 구분한다. 예를들어, "ha\_tcp\_ping\_hosts=host1:port1,host2:port2"와 같은 형식으로 설정할 수 있다. TCP 계층을 이용한 핑 체크가 정상적으로 동작하기 위해서는 반드시 **ha\_tcp\_ping\_hosts**에 설정된 호스트 상에서 해당 포트 번호를 갖는 소켓이 네트워크 요청을 정상적으로 수신할 수 있어야 하며, 방화벽에 의해 네트워크 요청이 차단되지 않아야 한다. **ha\_ping\_hosts**가 설정된 경우 **ha\_tcp\_ping\_hosts**는 사용되지 않고 무시된다.

이 파라미터는 동적으로 변경할 수 있으며, 변경하면 **cubrid heartbeat reload**를 실행해야 한다.

## 복제

### ha\_copy\_sync\_mode

트랜잭션 로그의 복사본인 복제 로그를 저장하는 모드를 설정한다. 기본값은 **SYNC**이다.

**SYNC**, **ASNYC**를 값으로 설정할 수 있다. **ha\_node\_list**에 지정한 노드의 수만큼 설정해야 하고 순서가 같아야 한다. 쉼표(,) 또는 콜론(:)으로 구분한다. 레플리카 노드는 이 값의 설정과 관계없이 항상 **ASNYC** 모드로 동작한다.

자세한 내용은 **로그 다중화** 를 참고한다.

### ha\_copy\_log\_base

복제 로그를 저장할 상위 경로를 지정한다. 기본값은 **\$CUBRID\_DATABASES** 환경 변수에 설정된 디렉토리 경로이다. 복제 로그들은 서버와 데이터베이스명에 따라 <db\_name>\_<host\_name>의 하위 디렉토리에 저장된다.

복제 로그 경로는 상대 경로 또는 절대 경로 설정이 가능하다. 다음은 각각의 설정 예제이다.

예1) **ha\_copy\_log\_base=copylog** : 상대 경로로 간주되어, 복제 로그 경로로 **\$CUBRID\_DATABASES/copylog**가 설정된다. 예2) **ha\_copy\_log\_base=/log/copy\_log** : 절대 경로로 간주되어, 복제 로그 경로로 **/log/copy\_log** **copylog base**가 설정된다.

### ha\_copy\_log\_max\_archives

복제 로그의 최대 보존 개수를 지정한다. 기본값은 1이다. 하지만, 복제 로그가 지정한 개수를 초과하더라도, 데이터베이스에 반영되지 않은 복제 로그 파일은 삭제되지 않는다.

불필요한 디스크의 공간 낭비를 방지하기 위해 이 값을 기본값인 1로 유지할 것을 권장한다.

### ha\_apply\_max\_mem\_size

CUBRID HA의 복제 로그 반영 프로세스가 사용할 수 있는 최대 메모리를 설정한다. 기본값은 **500** (단위: MB)이며, 최대값은 **INT\_MAX** (2147483647)이다. 이 값을 시스템이 허용하는 크기보다 너무 크게 설정하면 메모리 할당에 실패하면서 HA 복제 반영 프로세스가 오동작을 일으킬 수 있으므로, 메모리 자원이 설정한 값을 충분히 사용할 수 있는지 확인한 후 설정하도록 한다.

### ha\_applylogdb\_ignore\_error\_list

CUBRID HA의 복제 로그 반영 프로세스에서 에러가 발생해도 이를 무시하고 계속 복제를 진행하기 위해 이 값을 설정한다. 쉼표(,)로 구분하여 무시할 에러 코드를 나열한다. 이 설정 값은 높은 우선순위를 가지므로, **ha\_applylogdb\_retry\_error\_list** 파라미터나 “재시도 에러 리스트”에 의해 설정된 에러 코드와 값이 겹치면 이들을 무시하고 해당 에러를 유발한 작업을 재시도하지 않는다. “재시도 에러 리스트”는 아래 **ha\_applylogdb\_retry\_error\_list**의 설명을 참고한다.

### ha\_applylogdb\_retry\_error\_list

CUBRID HA의 복제 로그 반영 프로세스에서 에러가 발생하면 해당 에러를 유발한 작업이 성공할 때까지 반복적으로 재시도하기 위해 이 값을 설정한다. 쉼표(,)로 구분하여 재시도할 에러 코드를 나열한다. 이 값을 설정하지 않아도 기본으로 설정된 “재시도 에러 리스트”는 다음 표와 같다. 하지만 이 값들이 **ha\_applylogdb\_ignore\_error\_list**에 존재하면 에러를 무시하고 계속 복제를 진행한다.

#### 재시도 에러 리스트

| 에러 코드 이름                         | 에러 코드 |
|----------------------------------|-------|
| ER_LK_UNILATERALLY_ABORTED       | -72   |
| ER_LK_OBJECT_TIMEOUT_SIMPLE_MSG  | -73   |
| ER_LK_OBJECT_TIMEOUT_CLASS_MSG   | -74   |
| ER_LK_OBJECT_TIMEOUT_CLASSOF_MSG | -75   |
| ER_LK_PAGE_TIMEOUT               | -76   |
| ER_PAGE_LATCH_TIMEDOUT           | -836  |
| ER_PAGE_LATCH_ABORTED            | -859  |

| 에러 코드 이름                                | 에러 코드 |
|-----------------------------------------|-------|
| ER_LK_OBJECT_DL_TIMEOUT_SIMPLE_MS<br>G  | -966  |
| ER_LK_OBJECT_DL_TIMEOUT_CLASS_MSG       | -967  |
| ER_LK_OBJECT_DL_TIMEOUT_CLASSOF_M<br>SG | -968  |
| ER_LK_DEADLOCK_CYCLE_DETECTED           | -1021 |

### ha\_replica\_delay

마스터와 레플리카 사이의 데이터 복제 반영 시간 간격을 지정한다. 지정한 시간만큼 CUBRID는 의도적으로 복제 반영을 지연한다. ms, s, min, h 단위를 지정할 수 있으며 각각 milliseconds, seconds, minutes, hours를 의미한다. 단위 생략 시 기본 단위는 밀리초(ms)이다. 기본값은 0이다.

### ha\_replica\_time\_bound

마스터 노드에서 파라미터로 지정한 시간까지 수행된 트랜잭션만 레플리카 노드에 반영한다. 포맷은 "YYYY-MM-DD hh:mi:ss"이며, 기본값은 없다.

#### 참고

다음은 **cubrid\_ha.conf** 설정의 예이다.

```
[common]
ha_node_list=cubrid@nodeA:nodeB
ha_db_list=testdb
ha_copy_sync_mode=sync:sync
ha_apply_max_mem_size=500
```

#### 참고

다음은 멤버 노드의 호스트 이름이 *nodeA* 이고 IP 주소가 192.168.0.1일 때 */etc/hosts*를 설정한 예이다.

```
127.0.0.1 localhost.localdomain localhost
192.168.0.1 nodeA
```

## ha\_delay\_limit

**ha\_delay\_limit**은 CUBRID가 스스로 복제 지연 상태임을 판단하는 기준 시간이고 **ha\_delay\_limit\_delta**는 복제 지연 시간에서 복제 지연 해제 시간을 뺀 값이다. 한번 복제 지연이라고 판단된 서버는 복제 지연 시간이 (**ha\_delay\_limit** - **ha\_delay\_limit\_delta**) 이하로 낮아질 경우에 복제 지연이 해소되었다고 판단한다. 슬레이브 노드 또는 레플리카 노드가 복제 지연 여부를 판단하는 대상 서버, 즉 standby 상태의 DB 서버에 해당한다.

예를 들어 복제 지연 시간을 10분으로 설정하고 복제 지연 해제는 8분으로 하고 싶다면, **ha\_delay\_limit**의 값은 600s(또는 10min), **ha\_delay\_limit\_delta**의 값은 120s(또는 2min)이다.

복제 지연으로 판단되면 CAS는 현재 접속 중인 standby DB가 작업 처리에 문제가 있다고 판단하고, 다른 standby DB로 재접속을 시도한다.

복제 지연으로 인해 우선 순위가 낮은 DB에 연결된 CAS는 **cubrid\_broker.conf**의 **RECONNECT\_TIME** 파라미터로 명시한 시간이 경과하면 복제 지연이 해소되었을 것으로 기대하여, 우선 순위가 높은 standby DB에 재접속을 시도한다.

## ha\_delay\_limit\_delta

위에 기술한 **ha\_delay\_limit** 을 참조하라.

## ha\_copy\_log\_timeout

어떤 노드의 데이터베이스 서버 프로세스가 상대방 노드의 복제 로그 복사 프로세스로부터 응답을 대기하는 최대 시간이다. 기본값은 5(초)이다. 이 값이 -1이면 무한 대기한다. 오직 **SYNC** 로그 복제 모드(**ha\_copy\_sync\_mode**) 파라미터와 함께 작동한다.

## ha\_check\_disk\_failure\_interval

이 파라미터 값에 설정한 시간마다 디스크 장애 여부를 판단한다. 기본값은 15초이며, 단위는 초이다.

- **ha\_copy\_log\_timeout** 파라미터의 값이 -1인 경우, **ha\_check\_disk\_failure\_interval**의 값은 무시되며 디스크 장애 여부를 판단하지 않는다.
- **ha\_check\_disk\_failure\_interval**의 값이 **ha\_copy\_log\_timeout**의 값보다 작게 설정된 경우, **ha\_copy\_log\_timeout** + 20초의 시간마다 디스크 장애 여부를 판단한다.

## ha\_unacceptable\_proc\_restart\_timediff

서버 프로세스의 비정상 상황이 지속되는 경우 서버 재시작이 무한 반복될 수 있고, 이런 경우를 유발하는 노드는 HA 구성에서 제외하는 것이 바람직하다. 비정상 상황이 지속되면 보통 짧은 시간 간

격 이내에 서버가 재시작되므로, 이를 감지하기 위해 이 파라미터로 시간 간격을 명시한다. 명시한 시간 간격 이내에 서버가 재시작되면 CUBRID는 이 서버를 비정상 상태로 간주하고 해당 노드를 HA 구성에서 제외(demote)한다. 기본값은 2min이며, 단위를 지정하지 않으면 밀리초(msec)로 지정된다.

## SQL 로깅

### ha\_enable\_sql\_logging

이 파라미터의 값이 **yes**이면 복제 로그 디렉터리(**ha\_copy\_log\_base**) 이하의 **sql\_log** 디렉터리 이하에 **applylogdb** 프로세스가 DB에 반영하는 SQL에 대한 로그 파일을 생성한다. 기본값은 **no**이다.

로그 파일 이름의 형식은 **<db name>\_<master hostname>.sql.log.<id>**이며, **<id>**는 0부터 시작한다. **ha\_sql\_log\_max\_size\_in\_mbytes**에서 지정한 크기를 초과하면 **<id>**의 값이 하나 증가된 새로운 파일이 생성된다. 예를 들어, "ha\_sql\_log\_max\_size\_in\_mbytes=100"이면 demodb\_nodeA.sql.log.0 파일이 100MB가 되면서 demodb\_nodeA.sql.log.1이 새로 생성된다. **<id>**는 최대값(4,294,967,295)을 초과하게 되면 0부터 재사용하게 된다.

기본적으로, 2개의 최신 SQL 로그 파일만 유지되며, **ha\_sql\_log\_max\_count** 설정을 통해 유지할 최대 파일 개수를 조정할 수 있다.

SQL 로그 형식은 다음과 같다.

• INSERT/DELETE/UPDATE

```
-- 날짜 | SQL id | 샘플링을 위한 select문 길이 | 실제 변환된 SQL문 길이
-- 샘플링을 위한 select
실제 변환된 SQL 문
```

```
-- 2013-01-25 15:16:41 | 40083 | 33 | 114
-- SELECT * FROM [t1] WHERE "c1"=79186;
INSERT INTO [t1]("c1", "c2", "c3") VALUES
(79186, 'b3beb3decd2a6be974', 0);
```

• DDL

```
-- 날짜 | SQL id | 0 | 실제 변환된 SQL문 길이
ddl 구문
(create table의 경우 dba 권한으로 생성된 테이블에 권한을 부여하는 grant문
이 뒤따름)
```

```
-- 2013-01-25 14:22:59 | 1 | 0 | 50
create class t1 (id integer, primary key (id));
```

```
-- 2013-01-25 14:22:59 | 2 | 0 | 38
GRANT ALL PRIVILEGES ON [t1] TO public;
```

### ⚠ 경고

마스터 노드에서 트리거에 의해 수행된 작업이 SQL 로그 파일에 남게 되므로, 별도의 DB를 구축하면서 특정 시점부터 해당 SQL 로그를 반영하는 경우 트리거를 반드시 끈 상태로 적용해야 한다.

- 브로커 설정으로 트리거를 끄는 방법은 **TRIGGER\_ACTION**을 참고한다.
- CSQL 실행 시 트리거를 끄는 방법은 `csql --no-trigger-action` 을 참고한다.

## ha\_sql\_log\_max\_size\_in\_mbytes

**applylogdb** 프로세스가 DB에 반영하는 SQL이 로깅될 때 생성하는 파일의 최대 크기이다. 이 크기를 초과하면 새로운 파일이 생성된다.

## ha\_sql\_log\_max\_count

유지해야할 최대 SQL 로그 파일 개수를 나타내며, SQL 로그 파일이 이를 초과하여 생성될 경우 가장 오래된 로그 파일부터 삭제된다. 이 파라미터의 기본값은 2이며, 2에서 5 범위의 정수값을 설정할 수 있다. **ha\_sql\_log\_max\_count**와 **ha\_sql\_log\_max\_size\_in\_mbytes** 설정을 통해 적절한 양의 SQL 로그를 유지할 수 있다.

## ha\_sql\_log\_path

SQL 로그 파일이 생성될 디렉터리 경로를 나타내며, 상대 경로 또는 절대 경로로 설정할 수 있다. 상대 경로로 설정되는 경우 **ha\_enable\_sql\_logging**에서 설명한 복제 로그 디렉터리를 기준으로 상대 경로를 판단한다. 데이터베이스 서버 프로세스(cub\_server)는 반드시 지정된 경로에 SQL 로그 파일을 생성할 수 있는 권한이 있어야 한다. 이 파라미터가 설정되지 않은 경우에는 **ha\_enable\_sql\_logging**에서 설명한 복제 로그 디렉터리 경로에 SQL 로그 파일을 생성한다.

## cubrid\_broker.conf

**cubrid\_broker.conf** 파일은 **\$CUBRID/conf** 디렉터리에 위치하며, 브로커의 전반적인 설정 정보를 담고 있다. 여기에서는 **cubrid\_broker.conf** 중 CUBRID HA가 사용하는 파라미터를 설명한다.

브로커와 DB 사이의 연결 절차에 대한 자세한 설명은 **브로커와 DB 연결**을 참고한다.

## 접속 대상

### ACCESS\_MODE

브로커의 모드를 설정한다. 기본값은 **RW** 이다.

**RW** (Read Write), **RO** (Read Only), **SO** (Standby Only)를 값으로 설정할 수 있다. 자세한 내용은 **브로커 모드**를 참고한다.

## REPLICA\_ONLY

**REPLICA\_ONLY**의 값이 **ON**이면 CAS가 레플리카에만 접속된다. 기본값은 **OFF**이다. **REPLICA\_ONLY**의 값이 **ON**이고 **ACCESS\_MODE**의 값이 **RW**이면 레플리카 DB에도 쓰기 작업을 수행할 수 있다.

## 접속 순서

### CONNECT\_ORDER

CAS가 연결할 호스트 순서를 결정할 때 **\$CUBRID\_DATABASES/databases.txt**의 **db-host**에 설정된 호스트에서 순서대로 연결을 시도할지 랜덤한 순서대로 연결을 시도할지를 지정하는 파라미터이다.

기본값은 **SEQ**이며 순서대로 연결을 시도한다. **RANDOM**이면 랜덤한 순서대로 연결을 시도한다. **PREFERRED\_HOSTS** 파라미터 값이 주어지면 먼저 **PREFERRED\_HOSTS**에 명시된 호스트의 순서대로 연결을 시도한 후 실패할 경우에만 **db-host**의 설정 값을 사용한다. 그리고 **CONNECT\_ORDER**는 **PREFERRED\_HOSTS**의 순서에는 영향을 주지 않는다.

한 곳으로 DB 접속이 집중되는 상황이 우려되는 경우 이 값을 **RANDOM**으로 설정한다.

### PREFERRED\_HOSTS

호스트 이름을 나열하여 연결할 순서를 지정한다. 기본값은 **NULL**이다.

여러 노드를 지정할 수 있으며 콜론(:)으로 구분한다. 먼저 **PREFERRED\_HOSTS** 파라미터에 설정된 호스트 순서대로 연결을 시도한 후 **\$CUBRID\_DATABASES/databases.txt**에 설정된 호스트 순서대로 연결을 시도한다.

다음은 **cubrid\_broker.conf** 설정의 예이다. localhost에 우선 접속하기 위해 **PREFERRED\_HOSTS**를 localhost로 명시했다.

```
[%PHRO_broker]
SERVICE =ON
BROKER_PORT =33000
MIN_NUM_APPL_SERVER =5
MAX_NUM_APPL_SERVER =40
APPL_SERVER_SHM_ID =33000
LOG_DIR =log/broker/sql_log
ERROR_LOG_DIR =log/broker/error_log
SQL_LOG =ON
TIME_TO_KILL =120
SESSION_TIMEOUT =300
KEEP_CONNECTION =AUTO
CCI_DEFAULT_AUTOCOMMIT =ON
```

```
Broker mode setting parameter
ACCESS_MODE =RO
PREFERRED_HOSTS =localhost
```

## 접속 제한

### MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP

**databases.txt**의 **db-host**에 여러 대의 DB 서버를 명시한 HA 환경에서 거의 모든 DB 서버에서 복제 지연이 발생하는 경우, **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP** 파라미터에서 명시한 대수의 복제 지연 서버까지만 연결 여부를 검토한 후 연결을 결정한다(어떤 DB 서버의 복제 지연 여부는 standby 상태의 호스트만을 대상으로 판단하며, **ha\_delay\_limit** 파라미터의 설정에 따라 결정됨). 또한 **PREFERRED\_HOSTS**에는 **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP**이 적용되지 않는다.

예를 들어 **db-host**가 “host1:host2:host3:host4:host5”로 명시되고 “MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP=2”일 때, 만약 호스트들의 상태는 다음과 같을 경우 :

- host1 : active 상태
- host2 : standby 상태, 복제 지연
- host3 : 접속 불가
- host4 : standby 상태, 복제 지연
- host5 : standby 상태, 복제 지연 없음

이면 브로커는 먼저 복제 지연 상태인 2개의 호스트 *host2* , *host4* 까지 접속을 시도하고, *host4* 에 접속하는 것으로 결정한다.

이렇게 동작하는 이유는 **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP**에서 명시한 개수까지만 복제 지연이 있다면 이후의 호스트들에도 복제 지연이 있을 것이라는 가정을 하기 때문이며, 따라서 더 이상 뒤의 호스트에 대해 접속 시도를 하지 않고 복제 지연이 있지만 가장 마지막에 접속을 시도했던 호스트에 접속하기로 결정하는 것이다. 단, **PREFERRED\_HOSTS**가 같이 명시되는 경우 **PREFERRED\_HOSTS**에 명시된 모든 호스트들에 대해 접속을 먼저 시도한 후 다시 db-host 리스트의 처음부터 접속을 시도한다.

브로커가 DB에 접속하기 위한 단계는 1차 연결과 2차 연결로 나뉜다.

- 1차 연결: 브로커가 DB에 접속하기 위해 최초에 접속을 시도하는 단계. DB 상태(active/standby)와 복제 지연 여부를 확인.

먼저 **PREFERRED\_HOSTS**의 호스트들에 접속을 시도한 후, **databases.txt**의 호스트들에 접속을 시도한다. 이때는 **ACCESS\_MODE**에 따라 DB의 상태가 active인지, standby인지도 검사하여 접속을 결정한다.



- 2차 연결: 1차 연결 실패 후 실패한 위치에서부터 두번째로 접속을 시도하는 단계. DB 상태(active/standby)와 복제 지연 여부를 무시. 단, **SO** 브로커는 항상 standby DB에만 접속 허용.

이때는 DB의 상태(active/standby) 및 복제 지연 여부와 무관하게 접속이 가능하면 접속을 결정한다. 하지만 질의 수행 단계에서 에러가 발생할 수 있다. 예를 들어 **ACCESS\_MODE** 가 **RW**인데 standby 상태의 서버에 접속하면 INSERT 질의 수행 시 에러가 발생한다. 에러 발생과는 무관하게, standby로 연결되어 트랜잭션이 수행된 이후에는 1차 연결을 다시 시도한다. 단, **SO**브로커는 절대로 active DB에 연결될 수 없다.

**MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP**의 값에 따라 접속을 시도하는 호스트의 개수가 제한되는 방법은 다음과 같다:

- **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP=-1**

이 파라미터를 지정하지 않은 것과 같으며, 기본값이다. 이 경우 1차 연결에서는 끝까지 복제 지연 여부와 DB의 상태를 검사하여 연결을 결정한다. 2차 연결에서는 복제 지연이 있더라도, 또는 원하는 DB 상태(active/standby)가 아니더라도 가장 마지막에 연결 가능했던 호스트에 연결한다.

- **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP=0**

1차 연결에서 **PREFERRED\_HOSTS**에만 연결을 시도한 후 2차 연결이 진행되며, 2차 연결에서는 복제 지연이 있는 DB 서버이거나 원하는 DB 상태(active/standby)가 아니더라도 연결을 시도한다. 즉, 2차 연결이므로 **RW** 브로커도 standby 호스트에 연결될 수 있으며, **RO** 브로커도 active 호스트에 연결될 수 있다. 단, **SO** 브로커는 절대로 active DB에 연결될 수 없다.

- **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP=n(>0)**

지정된 개수의 복제 지연 호스트까지만 연결 시도한다. 1차 연결에서는 명시된 개수의 복제 지연 DB 서버까지 검사하고 난 이후, 2차 연결에서는 복제 지연이 있는 호스트에 연결한다.

## 재접속

### RECONNECT\_TIME

브로커가 **PREFERRED\_HOSTS**가 아닌 DB 서버에 접속하려고 하거나, **RO** 브로커가 active DB 서버에 접속하려고 하거나, 브로커가 복제 지연 DB 서버에 접속하려는 경우, **RECONNECT\_TIME**(기본값: 10분)을 초과하면 DB 서버에 재연결을 시도한다.

보다 자세한 내용은 **RECONNECT\_TIME**을 참고한다.

## databases.txt

**databases.txt** 파일은 **\$CUBRID\_DATABASES** (설정되어 있지 않은 경우 **\$CUBRID/databases**) 디렉터리에 위치하며, **db\_hosts** 값을 설정하여 브로커의 CAS가 접속을 시도하는 서버의 순서를 결정할 수 있다. 여러 노드를 설정하려면 콜론(:)으로 구분한다. **cubrid\_broker.conf**의 **CONNECT\_ORDER** 파라미

터 값이 **RANDOM**이면 무작위한 순서로 접속 순서를 결정한다. 하지만 **PREFERRED\_HOSTS** 파라미터 값이 설정된 경우 명시된 호스트로의 접속을 우선 시도한다.

다음은 **databases.txt** 설정의 예이다.

```
#db-name vol-path db-host log-path lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /home/cubrid/DB/
testdb/log file:/home/cubrid/DB/testdb/lob
```

## JDBC 설정

JDBC에서 CUBRID HA 기능을 사용하려면 브로커(*nodeA\_broker*)에 장애가 발생했을 때 다음으로 연결할 브로커(*nodeB\_broker*)의 연결 정보를 연결 URL에 추가로 지정해야 한다. CUBRID HA를 위해 지정되는 속성은 장애가 발생했을 때 연결할 하나 이상의 브로커 노드 정보인 **altHosts**이다. 이에 대한 자세한 설명은 [연결 설정](#)를 참고한다.

다음은 JDBC 설정의 예이다.

```
Connection connection =
DriverManager.getConnection("jdbc:CUBRID:nodeA_broker:
33000:testdb:::charset=utf-8&altHosts=nodeB_broker:33000", "dba",
"");
```

## CCI 설정

CCI에서 CUBRID HA 기능을 사용하려면 브로커에 장애가 발생했을 때 연결할 브로커의 연결 정보를 연결 URL에 추가로 지정할 수 있는 `cci_connect_with_url()` 함수를 사용하여 브로커와 연결해야 한다. CUBRID HA를 위해 지정되는 속성은 장애가 발생했을 때 연결할 하나 이상의 브로커 노드 정보인 **altHosts**이다.

다음은 CCI 설정의 예이다.

```
con = cci_connect_with_url ("cci:CUBRID:nodeA_broker:33000:testdb:::
altHosts=nodeB_broker:33000", "dba", NULL);
if (con < 0)
{
 printf ("cannot connect to database\n");
 return 1;
}
```

## PHP 설정

PHP에서 CUBRID HA 기능을 사용하려면 브로커에 장애가 발생했을 때 연결할 브로커의 연결 정보를 연결 URL에 추가로 지정할 수 있는 `cubrid_connect_with_url` 함수를 사용하여 브로커와 연결해야 한다. CUBRID HA를 위해 지정되는 속성은 장애가 발생했을 때 연결할 하나 이상의 브로커 노드 정보인 **altHosts**이다.

다음은 PHP 설정의 예이다.

```
<?php
$con = cubrid_connect_with_url ("cci:CUBRID:nodeA_broker:
33000:testdb:::altHosts=nodeB_broker:33000", "dba", NULL);
if ($con < 0)
{
 printf ("cannot connect to database\n");
 return 1;
}
?>
```

### 참고

altHosts를 설정하여 브로커 절체(failover)가 가능하도록 설정한 환경에서, 브로커 절체가 원활하게 되려면 URL에 **disconnectOnQueryTimeout** 값을 **true** 로 설정해야 한다.

이 값이 true면 질의 타임아웃 발생 시 응용 프로그램은 즉시 기존에 접속되었던 브로커와의 접속을 해제하고 **altHosts**에 지정한 브로커로 접속한다.

## 브로커와 DB 연결

HA 환경에서 브로커는 여러 개의 DB 서버 중 하나와 접속을 결정해야 한다. 이때 브로커와 DB 서버의 설정에 따라 어떤 DB 서버와 어떻게 접속할 것인지가 달라진다. 이 장에서는 HA 환경에서 설정에 따라 브로커가 DB 서버를 어떻게 선택하는지를 중심으로 살펴본다. 환경 설정에서 사용되는 각 파라미터들에 대한 설명은 **환경 설정**을 참고한다.

다음은 브로커와 DB가 연결될 때 사용되는 주요 파라미터들이다.

| 위치    | 설정 파일       | 파라미터 이름 | 설명 |
|-------|-------------|---------|----|
| DB 서버 | cubrid.conf | ha_mode |    |

| 위치  | 설정 파일              | 파라미터 이름                      | 설명                                                                                                                        |
|-----|--------------------|------------------------------|---------------------------------------------------------------------------------------------------------------------------|
|     |                    |                              | DB 서버의 HA 모드 (on/off/replica). 기본값: off                                                                                   |
|     | cubrid_ha.conf     | ha_delay_limit               | DB 서버에서 복제 지연 여부를 판단하는 복제 지연 기준이 되는 시간.                                                                                   |
|     |                    | ha_delay_limit_delta         | 복제 지연 기준 시간에서 복제 지연 해소 시간을 뺀 시간.                                                                                          |
| 브로커 | cubrid_broker.conf | ACCESS_MODE                  | 브로커 모드(RW/RO/SO). 기본값: RW                                                                                                 |
|     |                    | REPLICA_ONLY                 | REPLICA 서버로만 연결 가능 여부(ON/OFF). 기본값: OFF                                                                                   |
|     |                    | PREFERRED_HOSTS              | databases.txt의 db-host에서 설정한 호스트보다 우선하여 여기에서 지정한 호스트에 연결                                                                  |
|     |                    | MAX_NUM_DELAYED_HOSTS_LOOKUP | databases.txt에서 복제 지연으로 판단할 호스트의 개수. 명시한 개수의 호스트까지 복제 지연으로 판단되면 가장 마지막에 확인한 호스트와 연결.<br><br>• -1: databases.txt에 명시한 모든 호 |

| 위치 | 설정 파일 | 파라미터 이름        | 설명                                                                                                                                                           |
|----|-------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    |       |                | <p>스트의 복제 지연 여부를 확인.</p> <ul style="list-style-type: none"> <li>• 0: 복제 지연 여부를 확인하지 않고 바로 2차 연결을 수행.</li> <li>• n(&gt;0): n개의 호스트까지 복제 지연 여부를 확인.</li> </ul> |
|    |       | RECONNECT_TIME | <p>적합하지 않은 DB 서버에 연결된 이후 재연결을 시도하는 시간. 기본값: 600s. 이 값이 0이면 재연결을 시도하지 않음.</p>                                                                                 |
|    |       | CONNECT_ORDER  | <p>databases.txt에 설정된 호스트에서 순서대로 연결을 시도할지 랜덤한 순서대로 연결을 시도할지를 지정하는 파라미터 (SEQ/RANDOM). 기본값: SEQ</p>                                                            |

## 접속 절차

브로커가 DB 서버에 접속할 때, 1차 연결을 먼저 시도하고 실패하면 2차 연결을 시도한다.

• 1차 연결: DB 상태(active/standby)와 복제 지연 여부를 확인.

1. **PREFERRED\_HOSTS**에 명시된 순서로 접속을 시도한다. **ACCESS\_MODE**와 맞지 않는 상태의 DB 또는 복제 지연이 발생하는 DB에는 접속을 거부한다.

2. **CONNECT\_ORDER**의 값에 따라 **databases.txt**에 명시된 순서 혹은 무작위로 접속을 시도한다. **ACCESS\_MODE**에 따라 DB 서버의 상태를 확인하며, **MAX\_NUM\_DELAYED\_HOSTS\_LOOKUP** 개수까지 복제 지연 여부도 확인한다.

• 2차 연결: DB 상태(active/standby)와 복제 지연 여부를 무시. 단, **SO** 브로커는 항상 standby DB에만 접속 허용.

1. **PREFERRED\_HOSTS**에 명시된 순서로 접속을 시도한다. DB 서버의 상태가 **ACCESS\_MODE**와 맞지 않거나 DB에서 복제 지연이 발생하더라도 접속을 허용한다. 단, **SO** 브로커는 절대로 active DB에 연결될 수 없다.

2. **CONNECT\_ORDER**의 값에 따라 **databases.txt**에 명시된 순서 혹은 무작위로 접속을 시도한다. DB 서버의 상태 및 복제 지연 여부와 무관하게 접속이 가능하면 된다.

## 파라미터 설정에 따른 동작의 예

다음은 파라미터 설정에 따른 동작의 예이다.

### 호스트 DB 상태

- *host1*: active
- *host2*: standby, 복제 지연
- *host3*: standby, replica, 접속 불가
- *host4*: standby, replica, 복제 지연
- *host5*: standby, replica, 복제 지연

호스트 DB의 상태가 위와 같을 때, 설정에 따른 동작의 예는 다음과 같다.

### 설정에 따른 동작

- 2-1, 2-2, 2-3: 2로부터 (+)는 추가, (#)은 변경.
- 3-1, 3-2, 3-3: 3으로부터 (+)는 추가, (#)은 변경.

| 번호 | 설정                                                                                                                       | 동작                                                                                                                                                               |
|----|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1  | <ul style="list-style-type: none"> <li>• <b>ACCESS_MODE=RW</b></li> <li>• <b>PREFERRED_HOSTS=host2: host3</b></li> </ul> | <p>1차 연결 시도 시 DB 상태가 active인지 확인한다.</p> <ul style="list-style-type: none"> <li>• <b>PREFERRED_HOSTS</b>의 <i>host2</i>는 복제 지연이고 <i>host3</i>는 접속 불가이므로</li> </ul> |

| 번호  | 설정                                                                                                                                                                                                 | 동작                                                                                                                                                                                                                                                                                                                                                 |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <ul style="list-style-type: none"> <li>• db-host=host1:host2:host3:host4:host5</li> <li>• MAX_NUM_DELAYED_HOSTS_LOOKUP=-1</li> <li>• CONNECT_ORDER=SEQ</li> </ul>                                  | <p>db-host에 접속을 시도한다.</p> <ul style="list-style-type: none"> <li>• host1 이 active이므로 접속에 성공한다.</li> </ul> <p>PREFERRED_HOSTS에 접속하지 않았으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p>                                                                                                                                                                     |
| 2   | <ul style="list-style-type: none"> <li>• <b>ACCESS_MODE=RO</b></li> <li>• db-host=host1:host2:host3:host4:host5</li> <li>• MAX_NUM_DELAYED_HOSTS_LOOKUP=-1</li> <li>• CONNECT_ORDER=SEQ</li> </ul> | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• DB 상태가 standby인 호스트는 모두 복제 지연 또는 접속 불가이므로 프라이머리 연결에는 실패한다.</li> </ul> <p>2차 연결 시도 시 DB 상태와 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• 가장 마지막으로 접속했던 host5 가 접속에 성공한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p> |
| 2-1 | <ul style="list-style-type: none"> <li>• (+)PREFERRED_HOSTS=host1:host3</li> </ul>                                                                                                                 | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 host1 은 active이고 host3 는 접속이 불가하므로 db-host에 접속을 시도한다.</li> <li>• DB 상태가 standby인 호스트는 모두 복제 지연 또는</li> </ul>                                                                                                                                    |

| 번호  | 설정                                                                                                                              | 동작                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----|---------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     |                                                                                                                                 | <p>접속 불가이므로 1차 연결에는 실패한다.</p> <p>2차 연결 시도 시 DB 상태와 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이지만 접속이 가능하므로 브로커와의 접속에 성공한다.</li> </ul> <p>active 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p>                                                                                                                                                               |
| 2-2 | <ul style="list-style-type: none"> <li>• (+)PREFERRED_HOSTS=host1:host3</li> <li>• (#)MAX_NUM_DELAYED_HOSTS_LOOKUP=0</li> </ul> | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이고 <i>host3</i> 는 접속이 불가하다.</li> <li>• db-host에는 접속을 시도하지 않는다.</li> </ul> <p>2차 연결 시도 시 DB 상태와 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이지만 접속이 가능하므로 접속에 성공한다.</li> </ul> <p>active 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p> |



| 번호 | 설정                                                                                                                                                                                                 | 동작                                                                                                                                                                                                                                                                                                                                                                              |
|----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <ul style="list-style-type: none"> <li>• (#)MAX_NUM_DELAYED_HOSTS_LOOKUP=2</li> </ul>                                                                                                              | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• DB 상태가 standby 호스트에서 <i>host2</i> , <i>host4</i> 까지 복제 지연임을 확인한 후, 1차 연결에는 실패한다.</li> </ul> <p>2차 연결 시도 시 DB 상태와 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• 가장 마지막으로 접속했던 <i>host4</i> 가 접속에 성공한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p> |
| 3  | <ul style="list-style-type: none"> <li>• <b>ACCESS_MODE=SO</b></li> <li>• db-host=host1:host2:host3:host4:host5</li> <li>• MAX_NUM_DELAYED_HOSTS_LOOKUP=-1</li> <li>• CONNECT_ORDER=SEQ</li> </ul> | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• DB 상태가 standby인 호스트는 모두 복제 지연 또는 접속 불가이므로 1차 연결에는 실패한다.</li> </ul> <p>2차 연결 시도 시 DB 상태가 standby인지 확인하지만 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• 가장 마지막으로 접속했던 <i>host5</i> 가 접속에 성공한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p>          |

| 번호  | 설정                                                                                                                           | 동작                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-----|------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <ul style="list-style-type: none"> <li>• (+)PREFERRED_HOSTS=host1:host3</li> </ul>                                           | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이고 <i>host3</i> 는 접속이 불가하므로 db-host에 접속을 시도한다.</li> <li>• DB 상태가 standby인 호스트는 모두 복제 지연 또는 접속 불가이므로 1차 연결에는 실패한다.</li> </ul> <p>2차 연결 시도 시 DB 상태가 standby인지 확인하지만 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i>은 active이고 <i>host3</i> 는 접속 불가이므로 db-host에 접속을 시도한다.</li> <li>• DB 상태가 standby인 첫번째 호스트는 <i>host2</i> 이므로 <i>host2</i> 에 연결한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p> |
| 3-2 | <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS=host1:host3</li> <li>• (#)MAX_NUM_DELAYED_HOSTS_LOOKUP=0</li> </ul> | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이고 <i>host3</i> 는 접속이 불가하다.</li> <li>• db-host에는 접속을 시도하지 않는다.</li> </ul> <p>2차 연결 시도 시 DB 상태가 standby인지 확인하지만 복</p>                                                                                                                                                                                                                                                                                                                                                 |

| 번호  | 설정                                                                                    | 동작                                                                                                                                                                                                                                                                                                                                                                                                          |
|-----|---------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     |                                                                                       | <p>제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• PREFERRED_HOSTS의 <i>host1</i> 은 active이고 <i>host3</i> 는 접속 불가이므로 db-host에 접속을 시도한다.</li> <li>• DB 상태가 standby인 첫번째 호스트는 <i>host2</i> 이므로 <i>host2</i> 에 연결한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p>                                                                                                       |
| 3-3 | <ul style="list-style-type: none"> <li>• (#)MAX_NUM_DELAYED_HOSTS_LOOKUP=2</li> </ul> | <p>1차 연결 시도 시 DB 상태가 standby인지 확인한다.</p> <ul style="list-style-type: none"> <li>• DB 상태가 standby 호스트에서 <i>host2</i> , <i>host4</i> 까지 복제 지연임을 확인한 후, 1차 연결에는 실패한다.</li> </ul> <p>2차 연결 시도 시 DB 상태가 standby인지 확인하지만 복제 지연 여부는 확인하지 않는다.</p> <ul style="list-style-type: none"> <li>• 1차 연결에서 가장 마지막에 복제 지연 상태를 확인한 <i>host4</i> 에 연결한다.</li> </ul> <p>복제 지연 서버와 접속했으므로 RECONNECT_TIME 시간이 지나면 재접속을 시도한다.</p> |

## 구동 및 모니터링

## cubrid heartbeat 유틸리티

**cubrid heartbeat** 명령은 줄여서 **cubrid hb**로도 실행할 수 있다.

### start

해당 노드의 CUBRID HA 기능을 활성화하고 구성 프로세스(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)를 모두 구동한다. **cubrid heartbeat start**를 실행하는 순서에 따라 마스터 노드와 슬레이브 노드가 결정되므로, 순서를 주의해야 한다.

사용법은 다음과 같다.

```
$ cubrid heartbeat start
```

HA 모드로 설정된 데이터베이스 서버 프로세스는 **cubrid server start** 명령으로 시작할 수 없다.

노드 내에서 특정 데이터베이스의 HA 구성 프로세스들(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)만 구동하려면 명령의 마지막에 데이터베이스 이름을 지정한다. 예를 들어, 데이터베이스 *testdb*만 구동하려면 다음 명령을 사용한다.

```
$ cubrid heartbeat start testdb
```

### stop

해당 노드의 CUBRID HA 기능을 비활성화하고 구성 프로세스(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)를 모두 종료한다. 이 명령을 실행한 노드의 HA 기능은 종료되고 HA 구성에 있는 다음 순위의 슬레이브 노드로 failover가 일어난다.

사용법은 다음과 같다.

```
$ cubrid heartbeat stop
```

HA 모드로 설정된 데이터베이스 서버 프로세스는 **cubrid server stop** 명령으로 정지할 수 없다.

노드 내에서 특정 데이터베이스의 HA 구성 프로세스들(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)만 정지하려면 명령의 마지막에 데이터베이스 이름을 지정한다. 예를 들어, 데이터베이스 *testdb*를 정지하려면 다음 명령을 사용한다.

```
$ cubrid heartbeat stop testdb
```

CUBRID HA 기능을 즉각 비활성화하려면 “cubrid heartbeat stop” 명령에 **-i** 옵션을 추가한다. 이 옵션은 DB 서버 프로세스가 비정상적인 동작을 수행하고 있어 빠른 절체가 필요한 경우 사용한다.

```
$ cubrid heartbeat stop -i
or
$cubrid heartbeat stop --immediately
```

### copylogdb

CUBRID HA 구성에서 특정 *peer\_node*의 *db\_name*에 대한 트랜잭션 로그를 복사하는 **copylogdb** 프로세스를 시작 또는 정지한다. 운영 도중 복제 재구축을 위해 로그 복사를 일시 정지했다가 재구축하고 싶은 경우 사용할 수 있다.

**cubrid heartbeat copylogdb start** 명령만 성공한 경우에도 노드 간 장애 감지 및 복구 기능이 수행되며, failover의 대상이 되어 슬레이브 노드인 경우 마스터 노드로 역할이 변경될 수 있다.

사용법은 다음과 같다.

```
$ cubrid heartbeat copylogdb <start|stop> [-h <host-name>] db_name
peer_node
```

- <host-name> : copylogdb 명령이 수행될 원격 호스트명

명령을 수행하는 노드가 *nodeB* 이고, *peer\_node* 가 *nodeA* 라면, 다음과 같이 명령을 수행할 수 있다.

```
[nodeB]$ cubrid heartbeat copylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat copylogdb start testdb nodeA
```

**copylogdb** 프로세스의 시작/정지 시 **cubrid\_ha.conf**의 설정 정보를 사용하므로 한 번 정한 설정은 가급적 바꾸지 않을 것을 권장하며, 바꾸어야만 하는 경우 노드 전체를 재구축할 것을 권장한다.

### applylogdb

CUBRID HA 구성에서 특정 *peer\_node*의 *db\_name*에 대한 트랜잭션 로그를 반영하는 **applylogdb** 프로세스를 시작 또는 정지한다. 운영 도중 복제 재구축을 위해 로그 반영을 일시 정지했다가 재구축하고 싶은 경우 사용할 수 있다.

**cubrid heartbeat applylogdb start** 명령만 성공한 경우에도 노드 간 장애 감지 및 복구 기능이 수행되며, failover의 대상이 되어 슬레이브 노드인 경우 마스터 노드로 역할이 변경될 수 있다.

사용법은 다음과 같다.

```
$ cubrid heartbeat applylogdb <start|stop> [-h <host-name>]
db_name peer_node
```

- <host-name>: applylogdb 명령이 수행될 원격 호스트명

명령을 수행하는 노드가 *nodeB*이고, *peer\_node*가 *nodeA*라면, 다음과 같이 명령을 수행할 수 있다.

```
[nodeB]$ cubrid heartbeat applylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat applylogdb start testdb nodeA
```

**applylogdb** 프로세스의 시작/정지 시 **cubrid\_ha.conf**의 설정 정보를 사용하므로 한 번 정한 설정은 가급적 바꾸지 않을 것을 권장하며, 바꾸어야만 하는 경우 노드 전체를 재구동할 것을 권장한다.

## reload

**cubrid\_ha.conf**에서 CUBRID HA 구성 정보를 다시 읽는다. 노드를 추가하거나 삭제하는 경우 사용하며, **reload** 명령 이후에 추가/삭제된 노드의 HA 복제 프로세스를 일괄적으로 구동/정지하려면 “**cubrid heartbeat replication start/stop**” 명령을 사용할 수 있다.

사용법은 다음과 같다.

```
$ cubrid heartbeat reload
```

변경할 수 있는 구성 정보는 **ha\_node\_list**와 **ha\_replica\_list**이다. **reload** 명령이 실행된 후 **status** 명령으로 노드의 재구성이 잘 반영되었는지 확인한다. 재구성에 실패한 경우 원인을 찾아 해소하도록 한다.

## replication(또는 repl) start

특정 노드와 관련된 HA 프로세스(copylogdb/applylogdb)를 일괄 구동하기 위한 명령으로, 일반적으로 **cubrid heartbeat reload** 이후 추가된 노드의 HA 복제 프로세스들을 일괄적으로 시작하기 위해 실행한다.

**replication** 명령은 줄여서 **repl**로도 사용할 수 있다.

```
cubrid heartbeat repl start <node_name>
```

- *node\_name*: cubrid\_ha.conf의 **ha\_node\_list**에 명시된 노드 이름 중 하나

## replication(또는 repl) stop

특정 노드와 관련된 HA 프로세스(copylogdb/applylogdb)를 일괄 정지하기 위한 명령으로, 일반적으로 **cubrid heartbeat reload** 이후 삭제된 노드의 HA 복제 프로세스들을 일괄적으로 정지하기 위해 실행한다.

**replication** 명령은 줄여서 **repl**로도 사용할 수 있다.

```
cubrid heartbeat repl stop <node_name>
```

- *node\_name*: cubrid\_ha.conf의 **ha\_node\_list**에 명시된 노드 이름 중 하나

status

```
$ cubrid heartbeat status [-v] [-h <host-name>]
```

- <host-name>: status 명령이 수행될 원격 호스트명

CUBRID HA 그룹 정보와 CUBRID HA 구성 요소의 정보를 확인할 수 있다. 사용법은 다음과 같다.

```
$ cubrid heartbeat status
@ cubrid heartbeat status

HA-Node Info (current nodeB, state slave)
 Node nodeB (priority 2, state slave)
 Node nodeA (priority 1, state master)

HA-Process Info (master 2143, state slave)
 Applylogdb testdb@localhost:/home/cubrid/DB/testdb_nodeA (pid
2510, state registered)
 Copylogdb testdb@nodeA:/home/cubrid/DB/testdb_nodeA (pid 2505,
state registered)
 Server testdb (pid 2393, state registered_and_standby)
```

-v의 경우, 해당노드의 상세정보를 출력한다. **score**는 노드의 우선순위를 나타내며, 낮을 수록 높은 우선순위를 가진다. **missed heartbeat**은 서로의 노드를 인식하는 heartbeat의 유실율을 나타내며, 해당 값이 높은 경우 환경설정/네트워크/방화벽 등을 점검해야 한다.

Applylogdb, Copylogdb, Server 프로세스의 이벤트 발생 시간이며,이벤트가 발생하지 않은 경우 00:00:00.000으로 표기된다.

- registered-time : 명령어를 통하여 프로세스 구동 요청 시간
- deregistered-time : 명령어를 통하여 원격 프로세스 정지 요청 시간 (copylogdb와 applylogdb만 해당)
- shutdown-time : HA 매니저(cub\_master)가 프로세스를 정지한 시간
- start-time : HA 매니저(cub\_master)가 프로세스를 재 구동시간

예시

```

$ cubrid heartbeat status -v
@ cubrid heartbeat status

HA-Node Info (current cubrid1, state master)
 Node cubrid2 (priority 2, state slave)
 - score 2
 - missed heartbeat 0
 Node cubrid1 (priority 1, state master)
 - score -32767
 - missed heartbeat 0

HA-Process Info (master 7392, state master)
Copylogdb testdb@cubrid2:/home/cubha/CUBRID-11.4-latest-Linux.x86_64/
databases/testdb_cubrid2 (pid 7841, state registered)
 - exec-path [/home/cubha/CUBRID-11.4-latest-Linux.x86_64/bin/
cub_admin]
 - argv [cub_admin copylogdb -L /home/cubha/CUBRID-11.4-latest-
Linux.x86_64/databases/testdb_cubrid2 -m sync testdb@bagus2]
 - registered-time 08/26/24 14:28:37.019
 - deregistered-time 00/00/00 00:00:00.000
 - shutdown-time 08/26/24 14:28:35.010
 - start-time 08/26/24 14:28:36.012
Applylogdb testdb@localhost:/home/cubha/CUBRID-11.4-latest-
Linux.x86_64/databases/testdb_cubrid2 (pid 7746, state registered)
 - exec-path [/home/cubha/CUBRID-11.4-latest-Linux.x86_64/bin/
cub_admin]
 - argv [cub_admin applylogdb -L /home/cubha/CUBRID-11.4-latest-
Linux.x86_64/databases/testdb_cubrid2 --max-mem-size=300
testdb@localhost]
 - registered-time 08/26/24 14:27:14.566
 - deregistered-time 00/00/00 00:00:00.000
 - shutdown-time 08/26/24 14:27:12.552
 - start-time 08/26/24 14:27:13.558
Server testdb (pid 7904, state registered_and_active)
 - exec-path [/home/cubha/CUBRID-11.4-latest-Linux.x86_64/bin/
cub_server]
 - argv [cub_server testdb]
 - registered-time 08/26/24 14:29:28.955
 - deregistered-time 00/00/00 00:00:00.000
 - shutdown-time 08/26/24 14:29:27.593
 - start-time 08/26/24 14:29:28.594

```

## ❗ 참고

CUBRID 9.0 미만 버전에서 사용되었던 **act**, **deact**, **deregister** 명령은 더 이상 사용되지 않는다.



## cubrid service에 HA 등록

CUBRID 서비스에 heartbeat를 등록하면 **cubrid service** 유틸리티를 사용하여 한 번에 관련된 프로세스들을 모두 구동/정지하거나 상태를 알아볼 수 있어 편리하다. CUBRID 서비스 등록은 **cubrid.conf** 파일의 **[service]** 섹션에 있는 **service** 파라미터에 설정할 수 있다. 이 파라미터에 **heartbeat**를 포함하면 **cubrid service start / stop** 명령을 사용하여 서비스의 프로세스 및 HA 관련 프로세스를 모두 한 번에 구동/중지할 수 있다.

다음은 **cubrid.conf** 파일을 설정하는 예이다.

```
cubrid.conf

...

[service]

...

service=broker,heartbeat

...

[common]

...

ha_mode=on
```

## applyinfo

CUBRID HA의 복제 로그 복사 및 반영 상태를 확인한다.

```
cubrid applyinfo [options] <database-name>
```

- *database-name* : 확인하려는 서버의 데이터베이스 이름을 명시한다. 노드 이름은 입력하지 않는다.

**cubrid applyinfo**에서 사용하는 [options]는 다음과 같다.

### -r, --remote-host-name=HOSTNAME

트랜잭션 로그를 복사하는 대상 노드의 호스트 이름을 설정한다. 이 옵션을 설정하면 대상 노드의 액티브 로그 정보(Active Info.)를 출력한다.

### -a, --applied-info

cubrid applyinfo를 수행한 노드(localhost)의 복제 반영 정보(Applied Info.)를 출력한다. 이 옵션을 사용하기 위해서는 반드시 **-L** 옵션이 필요하다.

**-L, --copied-log-path=PATH**

상대 노드의 트랜잭션 로그를 복사해 온 위치를 설정한다. 이 옵션이 설정된 경우 상대 노드에서 복사해 온 트랜잭션 로그의 정보(Copied Active Info.)를 출력한다.

**-p, --pageid=ID**

-L 옵션을 설정한 경우 설정 가능하며, 복사해 온 로그의 특정 페이지 정보를 출력한다. 기본값은 0으로, 활성 페이지(active page)를 의미한다.

**-v**

더 자세한 내용을 출력한다.

**-i, --interval=SECOND**

트랜잭션 로그 복사 또는 반영 상태 정보를 지정한 초마다 주기적으로 출력한다. 복제가 지연되는 상태를 확인하려면 이 옵션을 반드시 지정해야 한다

**예시**

다음은 슬레이브 노드에서 **applyinfo** 를 실행하여 마스터 노드의 트랜잭션 로그 정보(Active Info.), 슬레이브 노드의 로그 복사 상태 정보(Copied Active Info.)와 로그 반영 상태 정보(Applied Info.)를 확인하는 예이다.

- Applied Info. : 슬레이브 노드가 복제 로그를 반영한 상태 정보를 나타낸다.
- Copied Active Info. : 슬레이브 노드가 복제 로그를 복사한 상태 정보를 나타낸다.
- Copied Archive Info. : 슬레이브 노드가 복사한 보관로그의 상태 정보를 나타낸다.
- Active Info. : 마스터 노드가 트랜잭션 로그를 기록한 상태 정보를 나타낸다.
- Delay in Copying Active Log: 트랜잭션 로그 복사 지연 상태를 나타낸다.
- Delay in Applying Copied Log: 트랜잭션 로그 반영 지연 상태를 나타낸다.

```
[nodeB] $ cubrid applyinfo -L /home/cubrid/DB/testdb_nodeA -r nodeA -a -p 0 -i 3 testdb
```

```
*** Applied Info. ***
```

```
Insert count : 289492
Update count : 71192
Delete count : 280312
Schema count : 20
Commit count : 124917
Fail count : 0
```

```
*** Copied Active Info. ***
```

```
DB name : testdb
DB creation time : 04:29:00.000 PM 11/04/2012
(1352014140)
Vol creation time : 04:29:10.000 PM 11/04/2012
```

```

(1352014150)
EOF LSA : 27722 | 10088
Append LSA : 27722 | 10088
HA server state : active

*** Copied Archive Info. ***
DB name : testdb
DB creation time : 04:29:00.000 PM 11/04/2012
(1352014140)
Vol creation time : 04:29:20.000 PM 11/04/2012
(1352014160)
Archive number : 0
Log page 0 (phy: 1 pageid: 0, offset 0)
offset:0000 (tid:1 bck p:-1,o:-1 frw p:0,o:96 type:3)
offset:0096 (tid:1 bck p:0,o:0 frw p:0,o:320 type:5)
offset:0320 (tid:1 bck p:0,o:96 frw p:0,o:552 type:4)
offset:0552 (tid:1 bck p:0,o:320 frw p:0,o:608 type:4)
...

*** Active Info. ***
DB name : testdb
DB creation time : 04:29:00.000 PM 11/04/2012
(1352014140)
Vol creation time : 04:29:10.000 PM 11/04/2012
(1352014150)
EOF LSA : 27726 | 2512
Append LSA : 27726 | 2512
HA server state : active

*** Delay in Copying Active Log ***
Delayed log page count : 4
Estimated Delay : 0 second(s)

*** Delay in Applying Copied Log ***
Delayed log page count : 1459
Estimated Delay : 22 second(s)

```

각 상태 정보가 나타내는 항목을 살펴보면 다음과 같다.

• Applied Info.

- Committed page : 복제 로그 반영 프로세스에 의해 마지막으로 반영된 트랜잭션의 커밋된 pageid와 offset 정보. 이 값과 “Copied Active Info.”의 EOF LSA 값의 차이만큼 복제 반영의 지연이 있다.
- Insert Count : 복제 로그 반영 프로세스가 반영한 Insert 쿼리의 개수
- Update Count : 복제 로그 반영 프로세스가 반영한 Update 쿼리의 개수
- Delete Count : 복제 로그 반영 프로세스가 반영한 Delete 쿼리의 개수

- Schema Count : 복제 로그 반영 프로세스가 반영한 DDL 문의 개수
- Commit Count : 복제 로그 반영 프로세스가 반영한 트랜잭션의 개수
- Fail Count : 복제 로그 반영 프로세스가 반영에 실패한 DML 및 DDL 문의 개수
- Copied Active Info.
  - DB name : 복제 로그 복사 프로세스가 로그를 복사하는 대상 데이터베이스의 이름
  - DB creation time : 복제 로그 복사 프로세스가 복사하는 데이터베이스의 생성 시간
  - Vol creation time : 복제 로그 복사 프로세스가 복사하는 볼륨의 생성 시간
  - EOF LSA : 복제 로그 복사 프로세스가 대상 노드에서 복사한 로그의 마지막 pageid와 offset 정보. 이 값과 “Active Info.”의 EOF LSA 값의 차이 및 “Copied Active Info.”의 Append LSA 값의 차이만큼 로그 복사의 지연이 있다.
  - Append LSA : 복제 로그 복사 프로세스가 디스크에 실제로 쓴 로그의 마지막 pageid와 offset 정보. 이는 EOF LSA보다 작거나 같을 수 있다. 이 값과 “Copied Active Info.”의 EOF LSA 값의 차이만큼 로그 복사의 지연이 있다.
  - HA server state : 복제 로그 복사 프로세스가 로그를 받아오는 데이터베이스 서버 프로세스의 상태. 상태에 대한 자세한 설명은 **서버** 를 참고하도록 한다.
- Copied Archive Info.
  - DB name : 복제 로그 복사 프로세스가 로그를 복사하는 대상 데이터베이스의 이름
  - DB creation time : 복제 로그 복사 프로세스가 복사하는 데이터베이스의 생성 시간
  - Vol creation time : 복제 로그 복사 프로세스가 복사하는 볼륨의 생성 시간
  - Archive number : 복제 로그 복사 프로세스가 복사하는 보관로그의 번호
  - Log page : 복제 로그 복사 프로세스가 복사하는 로그의 페이지 정보. pageid와 offset 정보를 포함한다.
  - offset : 로그 페이지의 offset 정보. 이 값은 pageid와 offset 정보를 포함한다.
- Active Info.
  - DB name : **-r** 옵션에 설정한 노드의 데이터베이스의 이름
  - DB creation time : **-r** 옵션에 설정한 노드의 데이터베이스 생성 시간
  - Vol creation time : **-r** 옵션에 설정한 노드의 볼륨 생성 시간
  - EOF LSA : **-r** 옵션에 설정한 노드의 데이터베이스 트랜잭션 로그의 마지막 pageid와 offset 정보. 이 값과 “Copied Active Info.”의 EOF LSA 값의 차이만큼 복제 로그 복사의 지연이 있다.

- Append LSA : **-r** 옵션에 설정한 노드의 데이터베이스 서버가 디스크에 실제로 쓴 트랜잭션 로그의 마지막 pageid와 offset 정보
- HA server state : **-r** 옵션에 설정한 노드의 데이터베이스 서버 상태
- Delay in Copying Active Log
  - Delayed log page count: 복사가 지연된 트랜잭션 로그 페이지 개수
  - Estimated Delay: 트랜잭션 로그 복사 예상 완료 시간
- Delay in Applying Copied Log
  - Delayed log page count: 반영이 지연된 트랜잭션 로그 페이지 개수
  - Estimated Delay: 트랜잭션 로그 반영 예상 완료 시간

레플리카 노드에서 해당 명령을 실행할 때 cubrid\_ha.conf에 “ha\_replica\_delay=30s”가 설정되어 있으면 다음의 정보가 추가로 출력된다.

```
*** Replica-specific Info. ***
Deliberate lag : 30 second(s)
Last applied log record time : 2013-06-20 11:20:10
```

각 상태 정보가 나타내는 항목을 살펴보면 다음과 같다.

- Replica-specific Info.
  - Deliberate lag: ha\_replica\_delay 파라미터에 의해 사용자가 설정한 지연 시간
  - Last applied log record time: 레플리카 노드에서 최근 반영된 복제 로그가 마스터 노드에서 실제로 반영되었던 시간

레플리카 노드에서 해당 명령을 실행할 때 cubrid\_ha.conf에 “ha\_replica\_delay=30s”, “ha\_replica\_time\_bound=2013-06-20 11:31:00”이 설정되어 있으면 ha\_replica\_delay 설정은 무시되며 다음의 정보가 추가로 출력된다.

```
*** Replica-specific Info. ***
Last applied log record time : 2013-06-20 11:25:17
Will apply log records up to : 2013-06-20 11:31:00
```

각 상태 정보가 나타내는 항목을 살펴보면 다음과 같다.

- Replica-specific Info.
  - Last applied log record time: 레플리카 노드에서 최근 반영한 복제 로그가 마스터 노드에서 반영된 시간

- Will apply log records up to: 해당 시간까지 마스터 노드에서 반영된 복제 로그를 레플리카 노드에 반영

ha\_replica\_time\_bound 시간이 되어 applylogdb가 복제 반영을 완전히 멈춘 경우 **\$CUBRID/log/db-name@local-node-name\_applylogdb\_db-name\_remote-node-name.err**에 출력되는 에러 메시지는 다음과 같다.

```
Time: 06/20/13 11:51:05.549 - ERROR *** file ../../src/transaction/
log_applier.c, line 7913 ERROR CODE = -1040 Tran = 1, EID = 3
HA generic: applylogdb paused since it reached a log record
committed on master at 2013-06-20 11:31:00 or later.
Adjust or remove ha_replica_time_bound and restart applylogdb to
resume.
```

## cubrid changemode

CUBRID HA의 서버 상태를 확인하고 변경한다.

```
cubrid changemode [options] <database-name@node-name>
```

- *database-name@node-name*: 확인 또는 변경하고자 하는 서버의 이름을 명시하고 @으로 구분하여 노드 이름을 명시한다. [options]를 생략하면 확인하고자 하는 서버 상태가 출력된다.

**cubrid changemode**에서 사용하는 [options]는 다음과 같다.

### -m, --mode=MODE

서버 상태를 변경한다.

옵션 값으로 **standby**, **maintenance**, **active** 중 하나를 입력할 수 있다.

- 서버의 상태가 **standby**이면 **maintenance**로 변경할 수 있다.
- 서버의 상태가 **maintenance**이면 **standby**로 변경할 수 있다.
- 서버의 상태가 **to-be-active**이면 **active**로 변경할 수 있다. 단, **--force** 옵션과 함께 사용해야 한다. 아래 **--force** 옵션의 설명을 참고한다.

### -f, --force

서버의 상태를 강제로 변경할지 여부를 설정한다.

현재 서버가 to-be-active 상태일 때 active 상태로 강제 변경하려고 하는 경우에는 반드시 사용하며, 이를 설정하지 않으면 active 상태로 변경되지 않는다. 강제 변경 시 복제 노드 간 데이터 불일치가 발생할 수 있으므로 사용하지 않는 것을 권장한다.

### -t, --timeout=SECOND

기본값 5(초). 노드 상태를 **standby**에서 **maintenance**로 변경할 때 진행 중이던 트랜잭션이 정상 종료되기까지 대기하는 시간을 설정한다.

설정된 시간이 지나도 트랜잭션이 진행 중이면 강제 종료 후 **maintenance** 상태로 변경하고, 설정한 시간 이내에 모든 트랜잭션이 정상 종료되면 즉시 **maintenance** 상태로 변경한다.

## 상태 변경 가능 표

다음은 현재 상태에 따라 변경할 수 있는 상태를 표시한 표이다.

|       |               | 변경할 상태 |         |             |
|-------|---------------|--------|---------|-------------|
|       |               | active | standby | maintenance |
| 현재 상태 | standby       | X      | O       | O           |
|       | to-be-standby | X      | X       | X           |
|       | active        | O      | X       | X           |
|       | to-be-active  | O*     | X       | X           |
|       | maintenance   | X      | O       | O           |

\* 서버가 to-be-active 상태일 때 active 상태로 강제 변경하면 복제 노드 간 불일치가 발생할 수 있으므로 관련 내용을 충분히 숙지한 사용자가 아니라면 사용하지 않는 것을 권장한다.

## 예시

다음 예는 localhost 노드의 *testdb* 서버 상태를 maintenance 상태로 변경한다. 이때 진행 중이던 모든 트랜잭션이 정상 종료하기까지 대기하는 시간은 -t 옵션의 기본값인 5초이다. 이 시간 이내에 모든 트랜잭션이 종료되면 즉시 상태를 변경하며, 이 시간이 지나도 진행 중인 트랜잭션이 존재하면 이를 롤백한 후 상태를 변경한다.

```
$ cubrid changemode -m maintenance testdb@localhost
The server 'testdb@localhost's current HA running mode is
maintenance.
```

다음 예는 localhost 노드의 *testdb* 서버의 상태를 조회한다.

```
$ cubrid changemode testdb@localhost
The server 'testdb@localhost''s current HA running mode is active.
```

## CUBRID 매니저 HA 모니터링

CUBRID 매니저는 CUBRID 데이터베이스 관리 및 질의 기능을 GUI 환경에서 제공하는 CUBRID 데이터베이스 전용 관리 도구이다. CUBRID 매니저는 CUBRID HA 그룹에 대한 관계도와 서버 상태를 확인할 수 있는 HA 대시보드를 제공한다. 자세한 설명은 CUBRID 매니저 매뉴얼을 참고한다.

## HA 구성 형태

CUBRID HA 구성에는 HA 기본 구성, 다중 슬레이브 노드 구성, 부하 분산 구성, 다중 스탠바이 서버 구성의 네 가지 형태가 있다. 다음 표에서 M은 마스터 노드, S는 슬레이브 노드, R은 레플리카 노드를 의미한다.

| 구성            | 노드 구성(M:S:R) | 특징                                                                                                                    |
|---------------|--------------|-----------------------------------------------------------------------------------------------------------------------|
| HA 기본 구성      | 1:1:0        | CUBRID HA의 가장 기본적인 구성으로, 하나의 마스터 노드와 하나의 슬레이브 노드로 구성되어 CUBRID HA 고유의 기능인 가용성을 제공한다.                                   |
| 다중 슬레이브 노드 구성 | 1:N:0        | 슬레이브 노드를 여러 개 두어 가용성을 높인 구성이다. 단, 다중 장애 상황에서 CUBRID HA 그룹 내의 데이터가 동일하지 않은 상황이 발생할 수 있으므로 주의해야 한다.                     |
| 부하 분산 구성      | 1:1:N        | HA 기본 구성에 레플리카 노드를 여러 개 둔다. 읽기 서비스의 부하를 분산할 수 있으며, 다중 슬레이브 노드 구성에 비해 HA로 인한 부담이 적다. 레플리카 노드는 failover되지 않으므로 주의해야 한다. |



| 구성      | 노드 구성(M:S:R) | 특징                                                             |
|---------|--------------|----------------------------------------------------------------|
| 다중 스탠바이 | 1:1:0        | HA 기본 구성과 노드 구성은 같으나 여러 서비스의 슬레이브 노드가 하나의 물리적인 서버에 설치되어 서비스된다. |

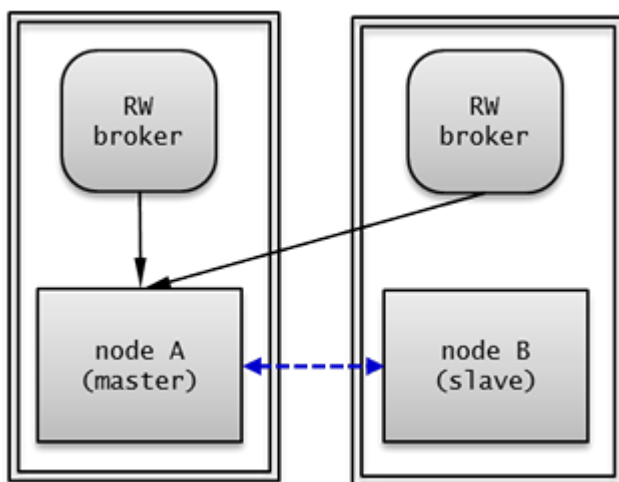
아래의 설명에서 각 노드의 testdb 데이터베이스에는 데이터가 없다고 가정한다. 이미 데이터가 존재하는 데이터베이스를 복제하여 HA를 구성하기 위해서는 **복제 구축** 또는 **복제 재구축 스크립트**를 참고한다.

## HA 기본 구성

CUBRID HA의 가장 기본적인 구성으로, 하나의 마스터 노드와 하나의 슬레이브 노드로 구성된다.

CUBRID HA 고유의 기능인 장애 시 무중단(nonstop) 서비스 기능에 초점을 맞춘 구성으로, 작은 서비스에서 적은 리소스를 투입하여 구성할 수 있다. HA 기본 구성은 하나의 마스터 노드와 하나의 슬레이브 노드로 서비스를 제공하므로, 읽기 부하를 분산하려면 다중 슬레이브 노드 구성 또는 부하 분산 구성이 좋다. 또한, 슬레이브 노드 또는 레플리카 노드 등의 특정 노드에 읽기 전용으로 접속하려면 Read Only 브로커 또는 Preferred Host Read Only 브로커를 구성한다. 브로커 구성에 대한 설명은 **브로커 이중화**를 참고한다.

### 노드 설정 예시



HA 기본 구성의 각 노드는 다음과 같이 설정한다.

- **node A** (마스터 노드)
  - **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=59901
ha_node_list=cubrid@nodeA:nodeB
ha_db_list=testdb
```

- **node B** (슬레이브 노드): *node A* 와 동일하게 설정한다.

브로커 노드의 **databases.txt** 파일에는 **db-host** 에 HA로 구성된 호스트의 목록을 우선순위에 따라 순서대로 설정해야 한다. 다음은 **databases.txt** 파일의 예이다.

```
#db-name vol-path db-host log-path
lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /home/cubrid/DB/
testdb/log file:/home/cubrid/DB/testdb/lob
```

**cubrid\_broker.conf** 파일은 브로커를 어떻게 구성하느냐에 따라 다양하게 설정할 수 있으며 **databases.txt** 파일과 함께 별도의 장비로 구성하여 설정할 수도 있다.

다음 예는 각 노드에 RW 브로커를 설정한 경우이며 *node A*, *node B* 둘 다 같은 값으로 구성한다.

```
[%RW_broker]
...

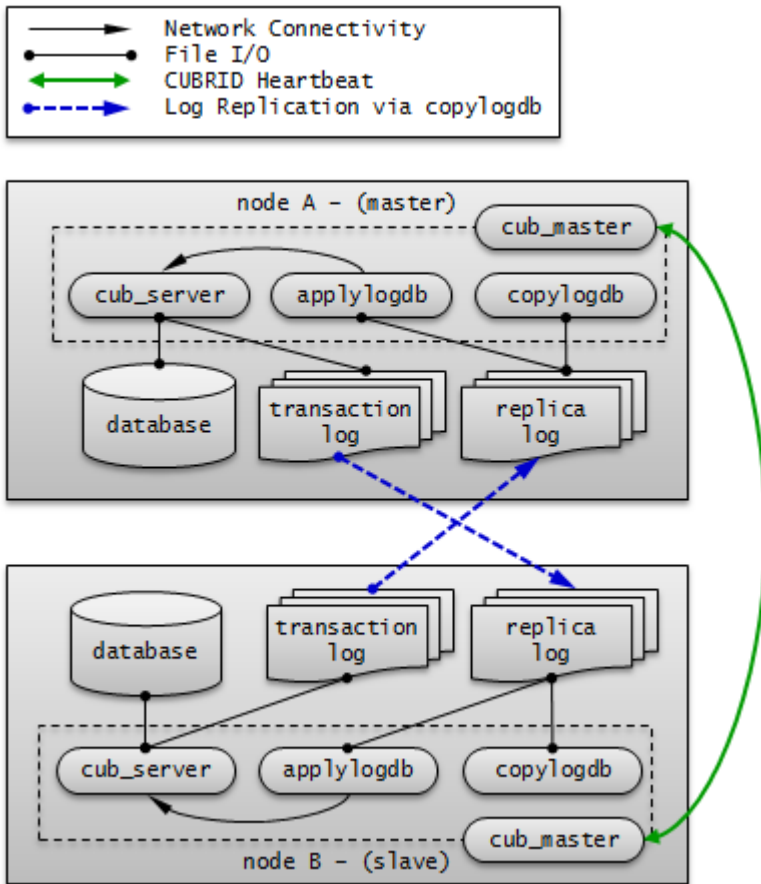
Broker mode setting parameter
ACCESS_MODE =RW
```

## 응용 프로그램 연결 설정

환경 설정의 **JDBC 설정**, **CCI 설정**, **PHP 설정** 을 참고한다.

## 참고

이와 같은 구성에서 트랜잭션 로그의 이동 경로를 중심으로 살펴보면 다음과 같다.



## 다중 슬레이브 노드 구성

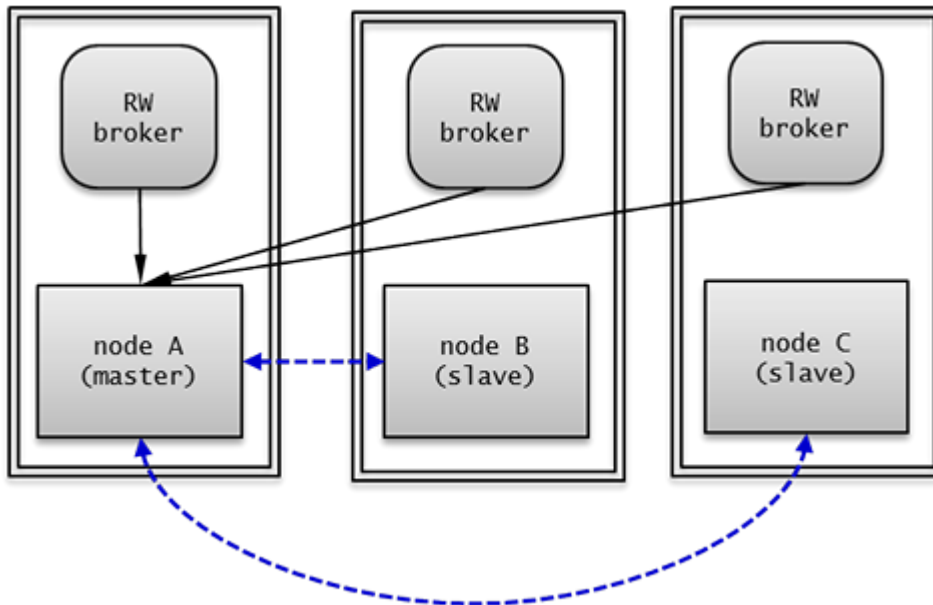
다중 슬레이브 노드 구성은 한 개의 마스터 노드와 여러 개의 슬레이브 노드를 두어 CUBRID의 서비스 가용성을 높인 구성이다.

CUBRID HA 그룹 내의 모든 노드에서 복제 로그 복사 프로세스와 복제 로그 반영 프로세스가 구동되므로 복제 로그를 복사하는 부하가 생긴다. 따라서 CUBRID HA 그룹 내의 모든 노드는 네트워크 및 디스크 사용률이 높다.

HA로 구성된 노드 수가 많으므로 CUBRID HA 그룹 내의 여러 노드에 장애가 발생해도 하나의 노드만 있으면 읽기 쓰기 서비스를 제공할 수 있다.

다중 슬레이브 노드 구성에서 failover가 일어날 때 마스터 노드가 될 노드는 **ha\_node\_list**에 정의한 순서에 따라 지정된다. 만약 **ha\_node\_list** 값이 nodeA:nodeB:nodeC이고 마스터 노드가 node A이면, 마스터 노드에 장애가 발생했을 때 node B가 마스터 노드가 된다.

## 노드 설정 예시



다중 슬레이브 구성의 각 노드는 다음과 같이 설정한다.

- **node A** (마스터 노드)
  - **cubrid.conf** 파일의 **ha\_mode** 를 **on** 으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=59901
ha_node_list=cubrid@nodeA:nodeB:nodeC
ha_db_list=testdb
ha_copy_sync_mode=sync:sync:sync
```

- **node B, node C** (슬레이브 노드): *node A* 와 동일하게 설정한다.

브로커 노드의 **databases.txt** 파일에는 **db-host** 에 HA 구성된 호스트의 목록을 우선순위에 따라 순서대로 설정해야 한다. 다음은 **databases.txt** 파일의 예이다.

| #db-name             | vol-path                        | db-host           | log-   |
|----------------------|---------------------------------|-------------------|--------|
| path                 | lob-base-path                   |                   |        |
| testdb               | /home/cubrid/DB/testdb          | nodeA:nodeB:nodeC | /home/ |
| cubrid/DB/testdb/log | file:/home/cubrid/DB/testdb/lob |                   |        |

**cubrid\_broker.conf** 파일은 브로커를 어떻게 구성하느냐에 따라 다양하게 설정할 수 있으며 **databases.txt** 파일과 함께 별도의 장비로 구성하여 설정할 수도 있다. 예시에서는 *node A*, *node B*, *node C*에 RW 브로커를 설정하였다.

다음은 *node A*, *node B*, *node C* 의 **cubrid\_broker.conf** 의 예이다.

```
[%RW_broker]
...

Broker mode setting parameter
ACCESS_MODE =RW
```

## 응용 프로그램 연결 설정

*node A*, *node B* 또는 *node C* 에 있는 브로커 중 하나와 연결한다.

```
Connection connection = DriverManager.getConnection(
 "jdbc:CUBRID:nodeA:33000:testdb::?charSet=utf-8&altHosts=nodeB:33000,nodeC:33000", "dba", "");
```

기타 자세한 사항은 환경 설정의 [JDBC 설정](#), [CCI 설정](#), [PHP 설정](#) 을 참고한다.

### 참고

이 구성은 다중 장애 시 CUBRID HA 그룹 내의 데이터가 동일하지 않은 상황이 발생할 수 있으며, 그 예는 다음과 같다.

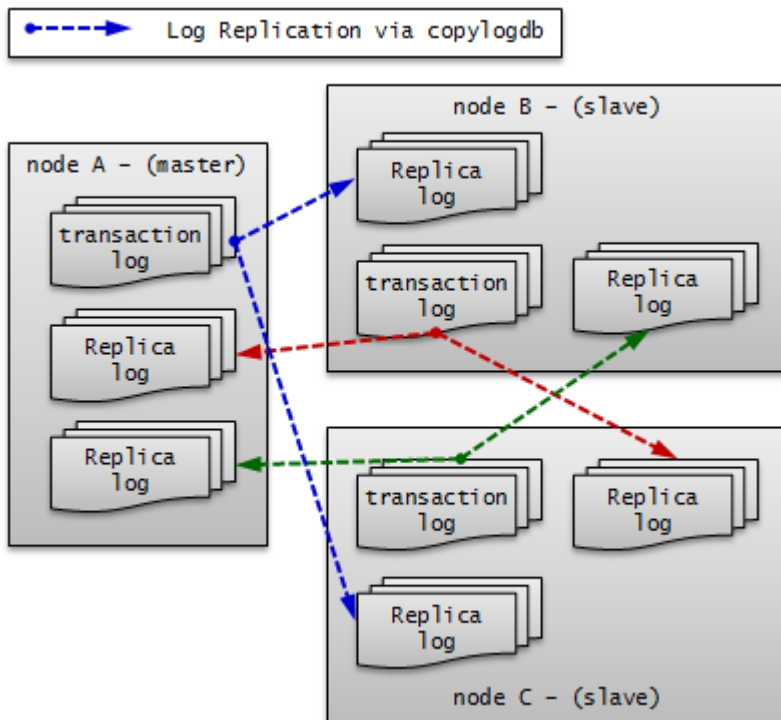
- 두 번째 슬레이브 노드가 재시작으로 인해 복제가 지연될 때 첫 번째 슬레이브로 failover되는 상황
- 빈번한 failover로 인해 새로운 마스터 노드의 복제 반영이 완료되지 않았을 때 다시 failover가 일어나는 상황

이외에 복제 로그 복사 프로세스의 모드가 ASYNC이면 CUBRID HA 그룹 내의 데이터가 동일하지 않은 상황이 발생할 수 있다.

이와 같이 CUBRID HA 그룹 내의 데이터가 동일하지 않은 상황이 발생하면, **복제 구축** 또는 **복제 재구축 스크립트**를 참고하여 CUBRID HA 그룹 내의 데이터를 동일하게 맞춰야 한다.

## 참고

이와 같은 구성에서 트랜잭션 로그의 이동 경로를 중심으로 살펴보면 다음과 같다.



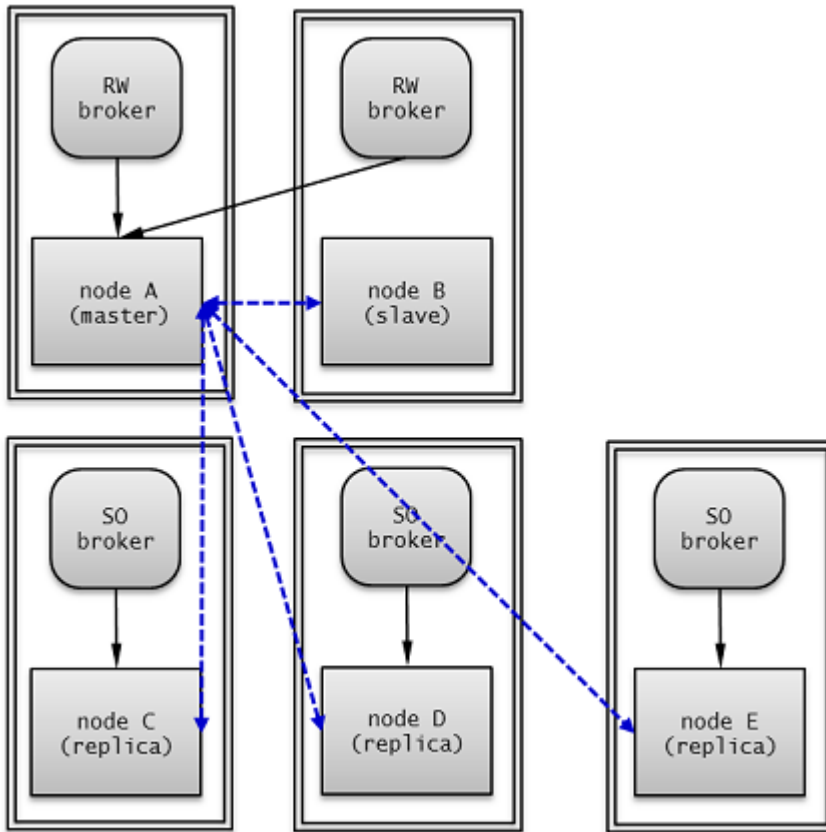
## 부하 분산 구성

부하 분산 구성은 HA 구성(한 개의 마스터 노드와 한 개의 슬레이브 노드)에 여러 개의 레플리카 노드를 두어 CUBRID 서비스의 가용성을 높이고, 많은 읽기 부하를 분산하여 처리할 수 있는 구성이다.

레플리카 노드들은 HA 구성 중 마스터 노드로부터 복제 로그를 받아 데이터를 동일하게 유지하고, 마스터 노드는 레플리카 노드에서 복제 로그를 받지 않으므로 다중 슬레이브 구성에 비해 네트워크 및 디스크 사용률이 낮다.

레플리카 노드는 HA 구성에 포함되지 않으므로 HA 구성 내의 모든 노드에 장애가 발생해도 failover 되지 않고 읽기 서비스만 제공한다.

## 노드 설정 예시



부하 분산 구성의 각 노드는 다음과 같이 설정한다.

- **node A** (마스터 노드)
  - **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=12345
ha_node_list=cubrid@nodeA:nodeB
ha_replica_list=cubrid@nodeC:nodeD:nodeE
ha_db_list=testdb
ha_copy_sync_mode=sync:sync
```

- **node B** (슬레이브 노드): *node A*와 동일하게 설정한다.
- **node C, node D, node E** (레플리카 노드)
  - **cubrid.conf** 파일의 **ha\_mode**를 **replica**로 설정한다.

```
ha_mode=replica
```

- **cubrid\_ha.conf** 파일은 *node A*와 동일하게 설정한다.

**cubrid\_broker.conf** 파일은 브로커를 어떻게 구성하느냐에 따라 다양하게 설정할 수 있으며 **databases.txt** 파일과 함께 별도의 장비로 구성하여 설정할 수도 있다.

예시에서는 브로커와 DB 서버를 같은 장비에 구성했으며, *node A*, *node B*에 RW 브로커를 설정하고, *node C*, *node D*, *node E*에 “CONNECT\_ORDER=RANDOM”과 “PREFERRED\_HOSTS=localhost”를 포함하는 SO 브로커를 설정하였다. *node C*, *node D*, *node E*는 “PREFERRED\_HOSTS=localhost”이기 때문에 로컬 DB 서버에 우선 접속을 시도하며, “CONNECT\_ORDER=RANDOM”이므로 localhost 접속에 실패할 경우 **databases.txt**의 db-host에 명시된 호스트들 중 하나에 랜덤하게 접속을 시도한다.

다음은 *node A*와 *node B*의 **cubrid\_broker.conf** 예이다.

```
[%RW_broker]
...

Broker mode setting parameter
ACCESS_MODE =RW
```

다음은 *node C*, *node D*, *node E*의 **cubrid\_broker.conf** 예이다.

```
[%PHRO_broker]
...

Broker mode setting parameter
ACCESS_MODE =SO
PREFERRED_HOSTS =localhost
```

브로커 노드의 **databases.txt** 파일에는 브로커의 용도에 맞게 HA 또는 부하 분산 서버와 연결될 수 있도록 DB 서버 호스트의 목록을 순서대로 설정해야 한다.

다음은 *node A*와 *node B*의 **databases.txt** 파일의 예이다.

```
#db-name vol-path db-host log-
path lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /home/cubrid/DB/
testdb/log file:/home/cubrid/CUBRID/testdb/lob
```

다음은 *node C*, *node D*, *node E*의 **databases.txt** 파일의 예이다.

```
#db-name vol-path db-host log-
path lob-base-path
testdb /home/cubrid/DB/testdb nodeC:nodeD:nodeE /home/cubrid/
DB/testdb/log file:/home/cubrid/CUBRID/testdb/lob
```



## 응용 프로그램 연결 설정

읽기 쓰기로 접속하기 위한 응용 프로그램은 *node A* 또는 *node B*에 있는 브로커에 연결한다. 다음은 JDBC 응용 프로그램의 예이다.

```
Connection connection = DriverManager.getConnection(
 "jdbc:CUBRID:nodeA:33000:testdb:::charset=utf-8&altHosts=nodeB:33000", "dba", "");
```

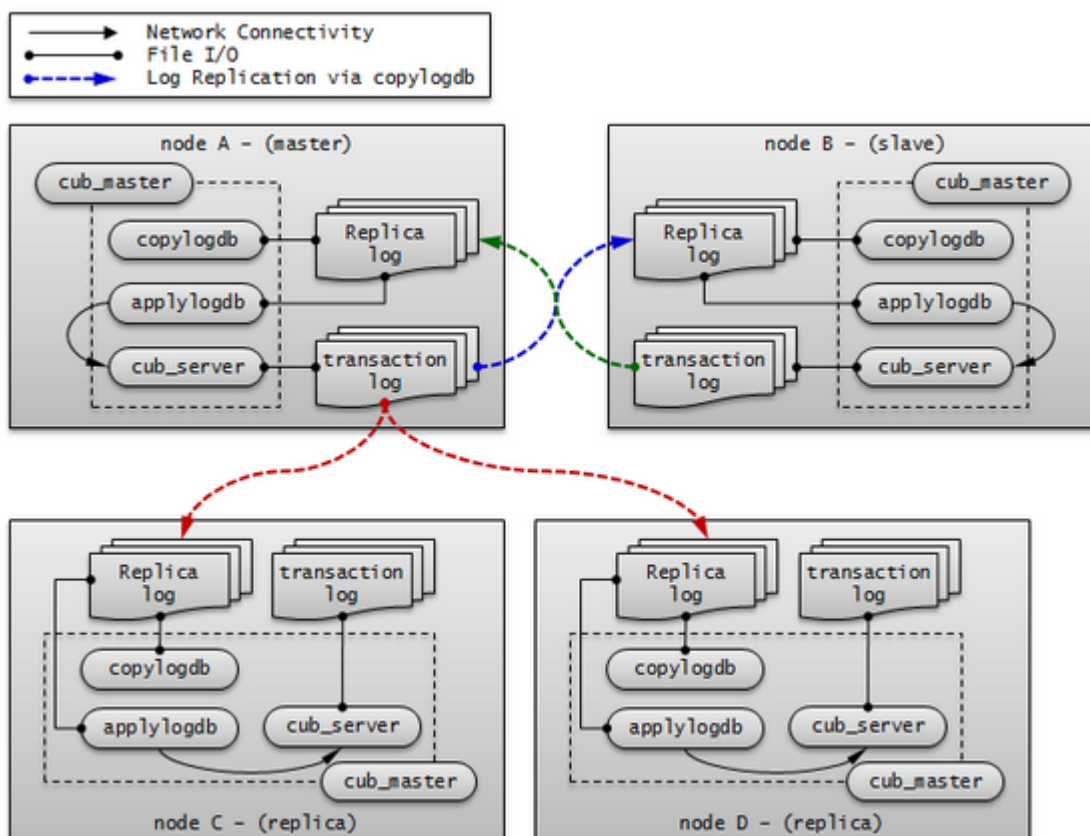
읽기 전용으로 접속하기 위한 응용 프로그램은 *node C*, *node D* 또는 *node E*에 있는 브로커에 연결한다. 다음은 JDBC 응용 프로그램의 예이다. “**loadBalance=sh**”로 설정하여 메인 호스트와 **altHosts**에 지정한 호스트들에 랜덤한 순서로 연결한다.

```
Connection connection = DriverManager.getConnection(
 "jdbc:CUBRID:nodeC:33000:testdb:::charset=utf-8&loadBalance=sh&altHosts=nodeD:33000,nodeE:33000",
 "dba", "");
```

기타 자세한 사항은 환경 설정의 [JDBC 설정](#), [CCI 설정](#), [PHP 설정](#)을 참고한다.

## 참고

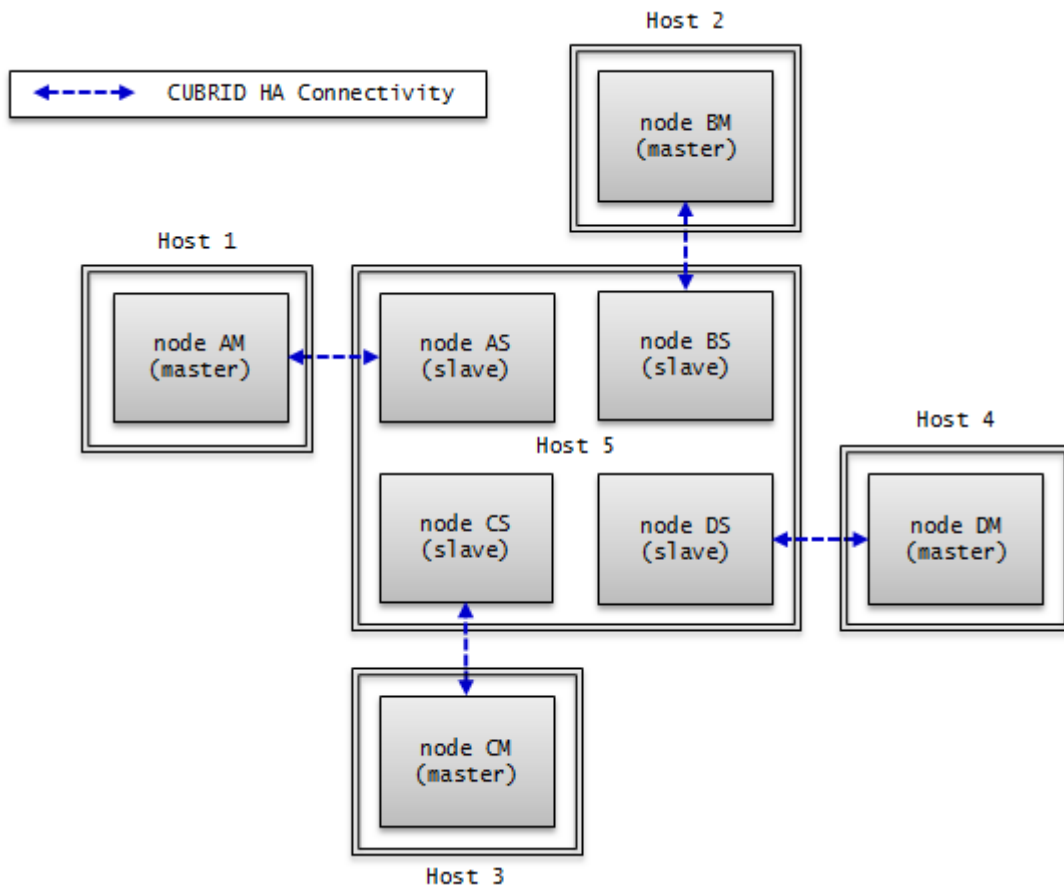
이와 같은 구성에서 트랜잭션 로그의 이동 경로를 중심으로 살펴보면 다음과 같다.



## 다중 스탠바이 서버 구성

한 개의 마스터 노드와 한 개의 슬레이브 노드로 구성되나, 여러 서비스의 슬레이브 노드를 하나의 물리적인 서버에 구성한다.

매우 작은 서비스에서 슬레이브 노드로 읽기 부하를 받지 않아도 되는 경우를 위한 것으로, CUBRID 서비스의 가용성만을 위한 구성이다. 따라서 failover 후 장애가 발생했던 마스터 노드가 복구되면 부하를 다시 마스터 노드로 옮겨 오도록 하여 슬레이브 노드들이 들어있는 서버의 부하를 최소화해야 한다.



### 노드 설정 예시

HA 기본 구성의 각 노드는 다음과 같이 설정한다.

- **node AM, node AS** : 두 노드는 동일하게 설정한다.
  - **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=10000
ha_node_list=cubridA@Host1:Host5
```

```
ha_db_list=testdbA1,testdbA2
ha_copy_sync_mode=sync:sync
```

- **node BM, node BS** : 두 노드는 동일하게 설정한다.
- **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=10001
ha_node_list=cubridB@Host2:Host5
ha_db_list=testdbB1,testdbB2
ha_copy_sync_mode=sync:sync
```

- **node CM, node CS** : 두 노드는 동일하게 설정한다.
- **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=10002
ha_node_list=cubridC@Host3:Host5
ha_db_list=testdbC1,testdbC2
ha_copy_sync_mode=sync:sync
```

- **node DM, node DS** : 두 노드는 동일하게 설정한다.
- **cubrid.conf** 파일의 **ha\_mode**를 **on**으로 설정한다.

```
ha_mode=on
```

- 다음은 **cubrid\_ha.conf** 파일의 설정 예이다.

```
ha_port_id=10003
ha_node_list=cubridD@Host4:Host5
ha_db_list=testdbD1,testdbD2
ha_copy_sync_mode=sync:sync
```

## HA 제약 사항

### 지원 플랫폼

현재 CUBRID HA 기능은 Linux 계열에서만 사용할 수 있다. CUBRID HA 그룹의 모든 노드들은 반드시 동일한 플랫폼으로 구성해야 한다.

### 테이블 기본키(primary key)

CUBRID HA는 마스터 노드의 서버에서 생성되는 기본키 기반의 복제 로그를 슬레이브 노드에 복제 후 반영하는 방식(transaction log shipping)으로 노드 간 데이터를 동기화하므로 기본키가 설정된 테이블에 대해서만 CUBRID HA 그룹 내의 노드 간 데이터 동기화가 가능하다.

CUBRID HA 그룹 내의 노드 간 특정 테이블의 데이터가 동기화되지 않는다면 해당 테이블에 적절한 기본키가 설정되어 있는지 확인해야 한다.

분할 테이블에서 **PROMOTE** 문에 의해 일부 분할이 승격된 테이블은 모든 데이터를 슬레이브에 복제하지만, 기본 키를 가지지 않게 되므로 이후 마스터에서 해당 테이블의 데이터를 수정해도 슬레이브에 반영되지 않음에 주의한다.

#### 참고

**USE\_SBR** 힌트를 통해 기본키가 설정되지 않은 테이블에 대한 데이터 복제도 지원한다. 자세한 내용은 [SQL 힌트](#) 을 참고한다.

### 저장 프로시저

CUBRID HA에서 **PL/CSQL 저장 프로시저**는 복제되지만 **Java 저장 프로시저**는 선언문만 복제되고, 실행 프로그램은 복제되지 않으므로 수동으로 HA의 모든 노드에 실행 프로그램을 복사해야 한다.

저장 프로시저 실행을 위해 CUBRID 프로시저 언어 서버의 구동이 필요하며 데이터베이스 서버 구동 시 자동으로 구동된다.

그러나 CUBRID 언어 서버 구동을 위해 **cubrid.conf** 환경 설정은 복제되지 않으므로 마스터 노드와 슬레이브 노드가 동일하게 **stored\_procedure** 가 **yes** (기본값) 로 설정되어 있는지 확인해야 한다.

### 메서드

CUBRID HA는 복제 로그를 기반으로 CUBRID HA 그룹 내의 노드 간 데이터를 동기화하므로 복제 로그를 생성하지 않는 메서드를 사용하면 CUBRID HA 그룹 내 노드 간 데이터 불일치가 발생할 수 있다.

따라서 CUBRID HA 환경에서는 메서드 사용(예: CALL login('dba', '') ON CLASS dbuser;)을 권장하지 않는다.

## stand-alone 모드

CUBRID의 stand-alone 모드에서 수행한 작업에 대해서는 복제 로그가 생성되지 않는다. 따라서 stand-alone 모드로 csql 등을 통해 작업 수행 시 CUBRID HA 그룹 내 노드 간 데이터 불일치가 발생할 수 있다.

## 시리얼 캐시(serial cache)

시리얼 캐시는 성능 향상을 위해 시리얼 정보를 조회하거나 갱신할 때 Heap에 접근하지 않고 복제 로그를 생성하지 않는다. 따라서 시리얼 캐시를 사용하는 경우 CUBRID HA 그룹 내 노드 간 시리얼의 현재 값이 일치하지 않는다.

## cubrid backupdb -r

이는 지정한 데이터베이스를 백업하는 명령으로 **-r** 옵션을 사용하면 백업을 수행한 후 복구에 필요하지 않은 로그를 삭제한다. 하지만 이 옵션으로 인해 로그가 사라지는 경우 CUBRID HA 그룹 내의 노드 간 데이터 불일치가 발생할 수 있으므로 **-r** 옵션을 사용하지 않아야 한다

## 온라인 백업

HA 환경에서 온라인 백업을 하려면 데이터베이스 이름 뒤에 @ *hostname*을 추가해야 한다. *hostname*은 \$CUBRID\_DATABASES/databases.txt에 지정된 이름이다. 보통 해당 서버의 로컬 DB를 온라인 백업하므로 "@localhost"를 지정하면 된다.

```
cubrid backupdb -C -D ./ -l 0 -z testdb@localhost
```

데이터베이스 운영 중에 수행하는 온라인 백업은 디스크 I/O의 부하가 예상되므로, 마스터 DB보다는 슬레이브 DB에서 백업할 것을 권장한다.

## INCR/DECR 함수

HA 구성의 슬레이브 노드에서 클릭 카운터 함수인 :**INCR()** / **DECR()** 함수를 사용하면 오류를 반환한다.

## LOB(BLOB/CLOB) 타입

CUBRID HA에서 **LOB** 칼럼 메타 데이터(Locator)는 복제되고, **LOB** 데이터는 복제되지 않는다. 따라서 **LOB** 타입 저장소가 로컬에 위치할 경우, 슬레이브 노드 또는 failover 이후 마스터 노드에서 해당 칼럼에 대한 작업을 허용하지 않는다.

## 참고

9.1 이전 버전의 CUBRID HA 환경에서 트리거를 사용할 경우 마스터 노드에서 이미 수행된 트리거를 슬레이브 노드에서 중복 수행하므로 CUBRID HA 그룹 내의 노드 간 데이터 불일치가 발생할 수 있다. 따라서 9.1 이전 버전의 CUBRID HA 환경에서는 트리거를 사용하지 않도록 한다.

## 참고

### UPDATE STATISTICS 문

10.0 부터는 UPDATE STATISTICS 문이 복제된다.

10.0 미만 버전에서는 UPDATE STATISTICS 문이 복제되지 않으므로 슬레이브/레플리카 노드에 별도로 수행해야 한다. 10.0 미만 버전의 슬레이브/레플리카 노드에서 “UPDATE STATISTICS” 구문을 적용하려면 CSQL에서 **--sysadm** 옵션과 **--write\_on\_slave** 옵션을 추가한 후 이 구문을 수행해야 한다.

## 운영 시나리오

### 읽기 쓰기 서비스 중 운영 시나리오

이 운영 시나리오는 서비스의 읽기 쓰기에 영향을 받지 않으므로, CUBRID 운영으로 인해 서비스에 미치는 영향이 매우 작다. 읽기 쓰기 서비스 중의 운영 시나리오는 failover가 일어나는 경우와 그렇지 않은 경우로 나눌 수 있다.

#### failover가 필요 없는 운영 시나리오

다음 작업은 CUBRID HA 그룹 내의 노드를 종료하고 다시 구동하지 않고 바로 수행할 수 있다.

| 대표적인 운영 작업                          | 시나리오                                       | 고려 사항                                                                            |
|-------------------------------------|--------------------------------------------|----------------------------------------------------------------------------------|
| 온라인 백업                              | 운영 중 마스터 노드와 슬레이브 노드에서 각각 운영 작업을 수행한다.     | 운영 작업으로 인해 마스터 노드의 트랜잭션이 지연될 수 있으므로 주의해야 한다.                                     |
| 스키마 변경(기본키 변경 작업 제외), 인덱스 변경, 권한 변경 | 마스터 노드에서만 운영 작업을 하면 자동으로 슬레이브 노드로 복제 반영한다. | 운영 작업이 마스터 노드에서 완료된 후 슬레이브 노드로 복제 로그가 복사되고 그 후부터 슬레이브 노드에 반영이 되므로 운영 작업 시간이 2배 소 |

| 대표적인 운영 작업   | 시나리오                                                                                                                                           | 고려 사항                                                                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              |                                                                                                                                                | 요 된다. 스키마 변경은 반드시 중간에 failover 없이 진행해야 한다. 스키마 변경을 제외한 인덱스 변경, 권한 변경은 운영 작업 소요 시간이 문제가 되는 경우, 각 노드를 정지한 후 독립 모드(예: <b>csql</b> 유틸리티의 <b>-S</b> 옵션)를 통해 수행할 수 있다. |
| 볼륨 추가        | HA 구성과 별개로 각 DB에서 운영 작업을 수행한다                                                                                                                  | 운영 작업으로 인해 마스터 노드의 트랜잭션이 지연될 수 있으므로 주의해야 한다. 운영 작업 소요 시간이 문제가 되는 경우 각 노드를 정지한 후 독립 모드(예: <b>cubrid addvoldb</b> 유틸리티의 <b>-S</b> 옵션)를 통해 수행할 수 있다.                |
| 장애 노드 서버 교체  | 장애 발생 후 실행 중인 CUBRID HA 그룹의 재시작 없이 교체한다.                                                                                                       | CUBRID HA 그룹 내 설정의 <code>ha_node_list</code> 에 장애 노드가 등록되어 있는 경우로, 교체 시 노드명 등이 변경되지 않아야 한다.                                                                      |
| 장애 브로커 서버 교체 | 장애 발생 후 실행 중인 브로커의 재시작 없이 교체한다.                                                                                                                | 클라이언트에서 교체된 브로커로의 연결은 URL 문자열에 설정된 <code>rcTime</code> 값에 의한다.                                                                                                   |
| DB 서버 증설     | 기존에 구성된 CUBRID HA 그룹의 재시작 없이 설정 변경( <code>ha_node_list</code> , <code>ha_replica_list</code> ) 후 <b>cubrid heartbeat reload</b> 를 각 노드에서 수행한다. | 변경된 설정 정보를 로딩하여 추가/삭제된 노드에 해당하는 <b>copylogdb/applylogdb</b> 프로세스를 시작 또는 정지한다.                                                                                    |

| 대표적인 운영 작업 | 시나리오                           | 고려 사항                                       |
|------------|--------------------------------|---------------------------------------------|
| 브로커 서버 증설  | 기존 브로커들의 재시작 없이 추가된 브로커를 구동한다. | 클라이언트가 추가된 브로커로 연결되기 위해서는 URL 문자열을 수정해야 한다. |

### failover가 필요한 운영 시나리오

다음 작업은 CUBRID HA 그룹 내의 노드를 종료하고 운영 작업을 완료한 후 구동해야 한다.

| 대표적인 운영 작업                | 시나리오                                                    | 고려 사항                                        |
|---------------------------|---------------------------------------------------------|----------------------------------------------|
| DB 서버 설정 변경               | <b>cubrid.conf</b> 의 설정이 변경되면 설정 변경된 노드를 재시작한다.         |                                              |
| 브로커 설정 변경, 브로커 추가, 브로커 삭제 | <b>cubrid_broker.conf</b> 의 설정이 변경되면 설정 변경된 브로커를 재시작한다. |                                              |
| DBMS 버전 패치                | HA 그룹 내 노드와 브로커들을 각각 버전 패치 후 재시작한다.                     | 버전 패치는 CUBRID의 내부 프로토콜, 볼륨 및 로그의 변경이 없는 것이다. |

### 읽기 서비스 중 운영 시나리오

이 운영 시나리오는 읽기 서비스만 가능하도록 하여 운영 작업을 수행한다. 서비스의 읽기 서비스만을 허용하거나 브로커의 모드 설정을 Read Only로 동적 변경해야 한다. 읽기 서비스 중의 운영 시나리오는 failover가 일어나는 경우와 그렇지 않은 경우로 나눌 수 있다.

### failover가 필요 없는 운영 시나리오

다음 작업은 CUBRID HA 그룹 내의 노드를 종료하고 다시 구동하지 않고 바로 수행할 수 있다.

| 대표적인 운영 작업 | 시나리오 | 고려 사항 |
|------------|------|-------|
|------------|------|-------|

스키마 변경(기본키 변경)



| 대표적인 운영 작업                          | 시나리오                                     | 고려 사항                                                                                                                                                                                                                                            |
|-------------------------------------|------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                     | 마스터 노드에서만 운영 작업하면 자동으로 슬레이브 노드로 복제 반영한다. | 기본키를 변경하려면 기본키를 삭제하고 다시 추가해야 한다. 따라서 기본키 기반의 복제 로그를 반영하는 HA 내부 구조 상 복제 반영이 일어나지 않을 수 있으므로, 반드시 읽기 서비스 중에 운영 작업을 수행해야 한다.                                                                                                                         |
| 스키마 변경(기본키 변경 작업 제외), 인덱스 변경, 권한 변경 | 마스터 노드에서만 운영 작업하면 자동으로 슬레이브 노드로 복제 반영한다. | 운영 작업이 마스터 노드에서 완료된 후 슬레이브 노드로 복제 로그가 복사되고 그 후부터 슬레이브 노드에 반영이 되므로 운영 작업 시간이 2배 소요 된다. 스키마 변경은 반드시 중간에 failover 없이 진행해야 한다. 스키마 변경을 제외한 인덱스 변경, 권한 변경은 운영 작업 소요 시간이 문제가 되는 경우, 각 노드를 정지한 후 독립 모드(예: <b>csql</b> 유틸리티의 <b>-S</b> 옵션)를 통해 수행할 수 있다. |

### failover가 필요한 운영 시나리오

다음 작업은 CUBRID HA 그룹 내의 노드를 종료하고 운영 작업을 완료한 후 구동해야 한다.

| 대표적인 운영 작업    | 시나리오                                           | 고려 사항                                                                                                                             |
|---------------|------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| DBMS 버전 업그레이드 | CUBRID HA 그룹 내 노드와 브로커들을 각각 버전 업그레이드 후 재시작 한다. | 버전 업그레이드는 CUBRID의 내부 프로토콜, 볼륨 및 로그의 변경이 있는 것이다. 업그레이드 중의 브로커 및 서버는 프로토콜, 볼륨 및 로그 등이 서로 맞지 않는 두 버전이 존재하게 되므로 업그레이드 전후의 클라이언트 및 브로커 |

| 대표적인 운영 작업                        | 시나리오                                | 고려 사항                                        |
|-----------------------------------|-------------------------------------|----------------------------------------------|
|                                   |                                     | 는 각각의 버전에 맞는 브로커 및 서버에 연결되도록 운영 작업을 수행해야 한다. |
| 대량의 데이터 작업 (INSERT/UPDATE/DELETE) | 작업할 노드를 정지하고 운영 작업을 수행한 후 노드를 구동한다. | 분할하여 작업할 수 없는 대량의 데이터 작업이 이에 해당한다.           |

## 서비스 정지 후 운영 시나리오

이 운영 시나리오는 CUBRID HA 그룹 내의 모든 노드들을 정지 후 운영 작업을 수행해야 한다.

| 대표적인 운영 작업          | 시나리오                                      | 고려 사항                                                                                      |
|---------------------|-------------------------------------------|--------------------------------------------------------------------------------------------|
| DB 서버의 호스트명 및 IP 변경 | CUBRID HA 그룹 내의 모든 노드를 정지하고 운영 작업 후 구동한다. | 호스트명 변경 시 각 브로커의 <b>databases.txt</b> 도 변경한 후 <b>cubrid broker reset</b> 으로 브로커의 연결을 리셋한다. |

## 레플리카 복제 지연 설정 시나리오

이 운영 시나리오는 마스터 노드에서 실수로 데이터를 삭제한 경우를 감지하여 특정 시점 이전으로 데이터를 되돌리고 싶을 때, 레플리카 노드를 통해 긴급히 이전 데이터까지 복구할 수 있게 하기 위해 지정한 시간 간격을 두고 복제를 진행하며 특정 시점에서 복제를 정지하도록 복제 정지 시간을 설정한다.

| 대표적인 운영 작업         | 시나리오                                           | 고려 사항                                                          |
|--------------------|------------------------------------------------|----------------------------------------------------------------|
| 레플리카 노드에서 복제 지연 설정 | 레플리카 노드에 복제되는 시간 간격을 지정하고, 특정 시간에 복제가 정지되게 한다. | cubrid_ha.conf의 ha_replica_delay와 ha_replica_time_bound를 지정한다. |

## 복제 구축

### restoreslave

**cubrid restoreslave** 는 백업본으로부터 데이터베이스를 복구하는 **cubrid restoredb** 와 동일하지만 슬레이브(slave)나 레플리카(replica)를 재구성할 때 편리한 기능이 포함되어 있다. **cubrid restoreslave** 를 사용하면 사용자가 **\_db\_ha\_apply\_info** 에 저장되는 복제 카탈로그 생성을 위해 백업 출력에서 복제 관련 정보를 수동으로 수집하지 않아도 된다. 이 명령은 백업 이미지와 활성 로그로부터 필요한 정보를 모두 자동으로 읽어서 관련 복제 카탈로그를 **\_db\_ha\_apply\_info** 에 추가한다. 사용자는 백업 이미지가 생성된 노드의 상태와 현재 마스터 노드의 호스트명, 이 두 가지 필수 옵션만 제공하면 된다. 자세한 내용은 **restoredb** 를 참고한다.

```
cubrid restoreslave [OPTION] database-name
```

#### **-s, --source-state=STATE**

백업 이미지가 생성된 노드의 상태를 지정해야 한다. STATE는 'master', 'slave' 또는 'replica'일 수 있다.

#### **-m, --master-host-name=NAME**

현재 마스터 노드의 호스트명을 입력한다.

#### **--list**

이 옵션은 데이터베이스의 백업 파일 정보를 표시한다; 복구 절차는 수행하지 않는다. 더 많은 정보는 **restoredb** 를 참고한다.

#### **-B, --backup-file-path=PATH**

이 옵션을 이용해서 백업 파일들이 위치할 디렉토리를 지정할 수 있다. 더 많은 정보는 **restoredb** 의 -B 옵션을 참고한다.

#### **-o, --output-file=FILE**

출력된 메시지들을 저장할 파일명을 지정할 수 있다. 더 많은 정보는 **restoredb** 의 -o 옵션을 참고한다.

#### **-u, --use-database-location-path**

이 옵션은 databases.txt에 지정된 데이터베이스 경로로 복구를 수행할 경우 지정한다. 더 많은 정보는 **restoredb** 의 -u 옵션을 참고한다.

#### **-k, --keys-file-path=PATH**

이 옵션은 복구 시 필요한 키 파일의 경로를 지정한다. 더 많은 정보는 **restoredb** 를 참고한다.

## 복제 구축 시나리오 예제

이 글에서는 HA 구성을 운영하는 도중 새로운 노드를 추가하거나 삭제하는 다양한 시나리오를 알아본다.

### 참고

- 기본 키가 있는 테이블만 복제된다는 점에 반드시 주의한다.
- 마스터 노드와 슬레이브 노드 (또는 레플리카 노드)의 볼륨 디렉터리들은 모두 일치해야 한다는 점에 주의한다.
- 하나의 노드는 다른 노드들에 대한 복제 로그 진행 정보를 모두 포함한다. 예를 들어, 마스터 *nodeA*와 슬레이브 *nodeB*, *nodeC*로 구성된 HA 구성을 가정하자. *nodeA*는 *nodeB*, *nodeC*에 대한 복제 진행 정보를 가지고 있으며, *nodeB*는 *nodeA*와 *nodeC*에 대한 정보를, *nodeC*는 *nodeA*와 *nodeB*에 대한 정보를 가지고 있다.
- 절체(failover)가 발생하지 않는다는 전제 하에 새로운 노드를 추가한다. 만약 복제 구축 도중 절체가 발생하면 다시 처음부터 복제 구축을 수행할 것을 권장한다.

### · 서비스 정지 후 슬레이브 추가

한 대의 장비로만 데이터베이스를 운영하다가 서비스 정지 후 슬레이브 노드를 새로 추가한다.

### · 서비스 운영 중 슬레이브 하나 더 추가

마스터 한 대, 슬레이브 한 대가 구축된 환경에서 기존의 슬레이브를 사용하여 새로운 슬레이브를 추가한다.

### · 서비스 운영 중 슬레이브 제거

마스터와 2개의 슬레이브가 구축된 환경에서 하나의 슬레이브를 제거한다.

### · 서비스 운영 중 레플리카 추가

마스터 한 대, 슬레이브 한 대가 구축된 환경에서 기존의 슬레이브를 사용하여 새로운 레플리카를 추가한다.

### · 서비스 운영 중 슬레이브 재구축

마스터, 슬레이브, 레플리카가 구축된 환경에서 마스터를 이용해 슬레이브를 재구축한다. 사용자에게 의해 `_db_ha_apply_info` 카탈로그 테이블의 정보를 변경해야 한다.

이상에 대해 좀더 자세히 알아보자.

HA 환경에서 비정상 노드만 재구축하고자 하는 경우 **복제 재구축 스크립트**를 참고한다.

## 참고

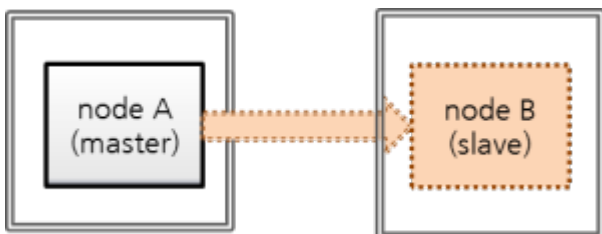
- 복제 구축 시 염두에 두어야 할 사항은 새로 구축하거나, 재구축하거나, 특정 노드를 제거하는 모든 경우, 해당 노드에 대한 복제 정보와 복제 로그를 변경 또는 삭제해야 한다는 점이다. 복제 정보는 `_db_ha_apply_info`에 저장되며, 상대방 노드의 복제 진행 상태 정보를 가지고 있다.

단, 레플리카의 경우 역할 변경(role change)이 발생하지 않으므로 상대방 노드가 레플리카 노드의 복제 정보와 복제 로그를 가지고 있을 필요가 없다. 반면, 레플리카는 마스터와 슬레이브 노드의 정보를 모두 가지고 있다.

- 서비스 운영 중 슬레이브 재구축**의 경우 사용자가 직접 `_db_ha_apply_info` 정보를 변경해야 한다.
- 서비스 정지 후 슬레이브 추가**의 경우 슬레이브를 새로 구축하므로 자동 생성되는 정보를 그대로 사용하면 된다.
- 서비스 운영 중 슬레이브 하나 더 추가, 서비스 운영 중 레플리카 추가**의 경우 기존의 슬레이브로부터 백업한 데이터베이스를 사용하는데, 기존의 슬레이브에는 마스터의 `_db_ha_apply_info` 정보가 포함되어 있으므로 그대로 사용할 수 있다.
- 서비스 운영 중 슬레이브 제거**의 경우 제거된 슬레이브에 대한 복제 정보를 사용자가 직접 삭제해야 한다. 하지만 이 정보가 남아 있다 하더라도 문제가 되진 않는다.

## 서비스 정지 후 슬레이브 추가

마스터 노드 한 대로만 운영하는 도중 서비스를 정지한 후 슬레이브를 추가하여 마스터와 슬레이브를 1:1로 구성하는 경우를 알아보자.



- nodeA: 마스터 노드 호스트 명
- nodeB: 새로 추가할 슬레이브 노드 호스트 명

이 시나리오에서 데이터베이스는 다음 명령으로 이미 생성되어 있다고 가정한다. `createdb` 수행 시 로컬 이름과 문자셋은 마스터 노드와 슬레이브 노드가 서로 동일해야 한다.

```
export CUBRID_DATABASES=/home/cubrid/DB
mkdir $CUBRID_DATABASES/testdb
mkdir $CUBRID_DATABASES/testdb/log
cd $CUBRID_DATABASES/testdb
cubrid createdb testdb -L $CUBRID_DATABASES/testdb/log en_US.utf8
```

백업 파일의 저장 위치를 별도의 옵션으로 지정하지 않으면 \$CUBRID\_DATABASES/testdb 디렉터리가 기본이 된다.

브로커 노드가 별도의 장비에 구성되어 있고 브로커를 추가하지 않는 경우, 응용 프로그램에서 URL의 변경을 필요로 하지 않는다.

### 참고

브로커와 데이터베이스의 장비가 분리되어 있지 않은 경우 마스터의 장비가 고장날 것에 대비해 슬레이브 장비에서 브로커 접속이 가능해야 한다. 이를 위해, 응용 프로그램의 접속 URL에 altHosts를 추가해야 하며, 브로커 이중화를 위해 databases.txt의 설정이 변경되어야 한다.

HA 환경에서 브로커 설정은 [cubrid\\_broker.conf](#)를 참고한다.

이 시나리오에서는 브로커 노드를 별도의 장비로 분리하지 않은 것으로 가정한다.

이상을 염두에 두고 다음의 순서로 작업한다.

#### 1. 마스터 노드와 슬레이브 노드의 HA 설정

- 마스터 노드와 슬레이브 노드가 동일하게 **\$CUBRID/conf/cubrid.conf** 설정

```
[service]
service=server,broker,manager

서비스 시작 시 구동될 데이터베이스 이름 추가
server=testdb

[common]
...

HA 구성 시 추가 (Logging parameters)
force_remove_log_archives=no

HA 구성 시 추가 (HA 모드)
ha_mode=on
```

- 마스터 노드와 슬레이브 노드가 동일하게 **\$CUBRID/conf/cubrid\_ha.conf** 설정

```
[common]
ha_port_id=59901

cubrid는 HA 시스템의 그룹 이름이고, nodeA와 nodeB는 호스트 이름임.
ha_node_list=cubrid@nodeA:nodeB

ha_db_list=testdb
ha_copy_sync_mode=sync:sync
ha_apply_max_mem_size=500
```

- 마스터 노드와 슬레이브 노드가 동일하게 **\$CUBRID\_DATABASES/databases.txt** 설정.

```
#db-name vol-path db-host log-
path lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /home/cubrid/
DB/testdb/log file:/home/cubrid/DB/testdb/lob
```

- HA 환경에서 브로커, 응용 프로그램 설정

브로커 설정과 응용 프로그램에서 URL에 altHosts를 추가하는 것에 대한 설명은 **cubrid\_broker.conf**를 참고한다.

- 마스터 노드와 동일하게 슬레이브 노드의 로컬 라이브러리 설정

```
[nodeA]$ scp $CUBRID/conf/cubrid_locales.txt cubrid_usr@nodeB:
$CUBRID/conf/.
[nodeB]$ make_locale.sh -t64
```

- 슬레이브 노드에 데이터베이스 디렉터리 생성

```
[nodeB]$ cd $CUBRID_DATABASES
[nodeB]$ mkdir testdb
```

- 슬레이브 노드에 로그 디렉터리 생성(마스터 노드와 같은 위치에 생성)

```
[nodeB]$ cd $CUBRID_DATABASES/testdb
[nodeB]$ mkdir log
```

## 2. 마스터 노드 서비스 중지, 마스터 데이터베이스 백업

```
[nodeA]$ cubrid service stop
```

마스터 노드의 데이터베이스를 백업하고, 슬레이브 노드에 백업 파일을 복사한다.

마스터 노드에서 백업 파일의 저장 위치는 별도의 지정이 없으면 *testdb* 의 로그 디렉터리가 되며, 슬레이브 노드에도 마스터 노드와 같은 위치에 백업 파일을 복사한다. 아래에서 *testdb\_bk0v000*은 백업 볼륨 파일, *testdb\_bkvinf*는 백업 볼륨 정보 파일이다.

```
[nodeA]$ cubrid backupdb -z -S testdb
Backup Volume Label: Level: 0, Unit: 0, Database testdb, Backup
Time: Thu Apr 19 16:05:18 2012
[nodeA]$ cd $CUBRID_DATABASES/testdb/log
[nodeA]$ scp testdb_bk* cubrid_usr@nodeB:$CUBRID_DATABASES/testdb/
log
cubrid_usr@nodeB's password:
testdb_bk0v000 100% 6157KB 6.0MB/s
00:00
testdb_bkvinf 100% 66 0.1KB/s
00:00
```

### 3. 슬레이브 노드에서 데이터베이스 복구.

이때, 마스터 노드와 슬레이브 노드의 볼륨 경로가 반드시 같아야 한다.

```
[nodeB]$ cubrid restoredb -B $CUBRID_DATABASES/testdb/log testdb
```

### 4. 마스터 노드 시작, 슬레이브 노드 시작

```
[nodeA]$ cubrid heartbeat start
```

마스터 노드가 정상 구동되었음을 확인한 후, 슬레이브 노드를 시작한다.

제일 아래에서 *nodeA*의 state가 *registered\_and\_to\_be\_active*에서 *registered\_and\_active*로 변경 되면 마스터 노드가 정상 구동된 것이다.

```
[nodeA]$ cubrid heartbeat status
@ cubrid heartbeat status

HA-Node Info (current nodeA, state master)
 Node nodeB (priority 2, state unknown)
 Node nodeA (priority 1, state master)

HA-Process Info (master 123, state master)

 Applylogdb testdb@localhost:/home/cubrid/DB/testdb_nodeB (pid
234, state registered)
 Copylogdb testdb@nodeB:/home/cubrid/DB/testdb_nodeB (pid 345,
state registered)
 Server testdb (pid 456, state registered_and_to_be_active)
```



```
[nodeB]$ cubrid heartbeat start
```

마스터 노드, 슬레이브 노드의 HA 구성이 정상 동작하는지 확인한다.

```
[nodeA]$ csql -u dba testdb@localhost -c"create table tbl(i int
primary key);insert into tbl values (1),(2),(3)"
```

```
[nodeB]$ csql -u dba testdb@localhost -c"select * from tbl"
```

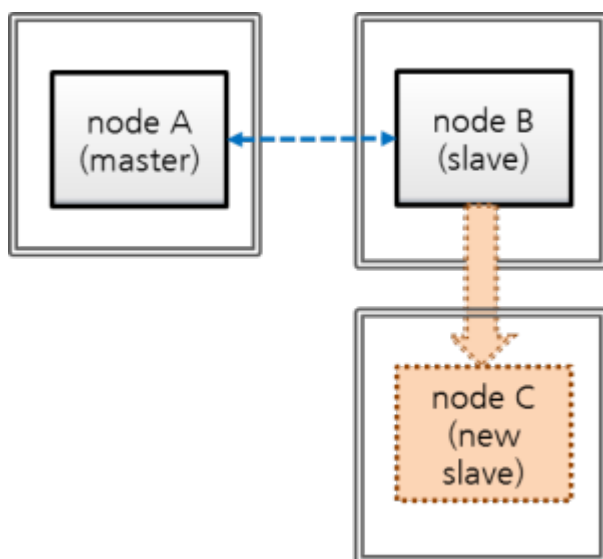
```

 i
=====
 1
 2
 3

```

## 서비스 운영 중 슬레이브 하나 더 추가

다음은 마스터:슬레이브로 HA 서비스 운영 중에 기존 슬레이브로부터 새 슬레이브를 추가하는 시나리오이다. “마스터:슬레이브”는 1:1에서 1:2가 된다.



- nodeA: 마스터 노드 호스트 명
- nodeB: 슬레이브 노드 호스트 명
- nodeC: 새로 추가할 슬레이브 노드 호스트 명

HA 서비스 운영 중 슬레이브를 새로 추가하려면 기존의 마스터 또는 슬레이브를 이용할 수 있는데, 보통 마스터에 비해 슬레이브가 상대적으로 디스크 I/O 부하가 적다고 보고, 여기에서는 기존 슬레이브를 이용하여 추가로 슬레이브를 구성해보자.

### 참고

#### 노드 추가 시 마스터 대신 슬레이브를 이용할 수 있는 이유

마스터 대신 슬레이브를 이용하는 것이 가능한 이유는, 마스터의 트랜잭션 로그(활성 로그 + 보관 로그)가 슬레이브에 그대로 복제되며 마스터 노드의 트랜잭션 로그와 슬레이브 노드의 복제 로그(복제된 트랜잭션 로그)가 형식이나 내용 면에서 동일하기 때문이다.

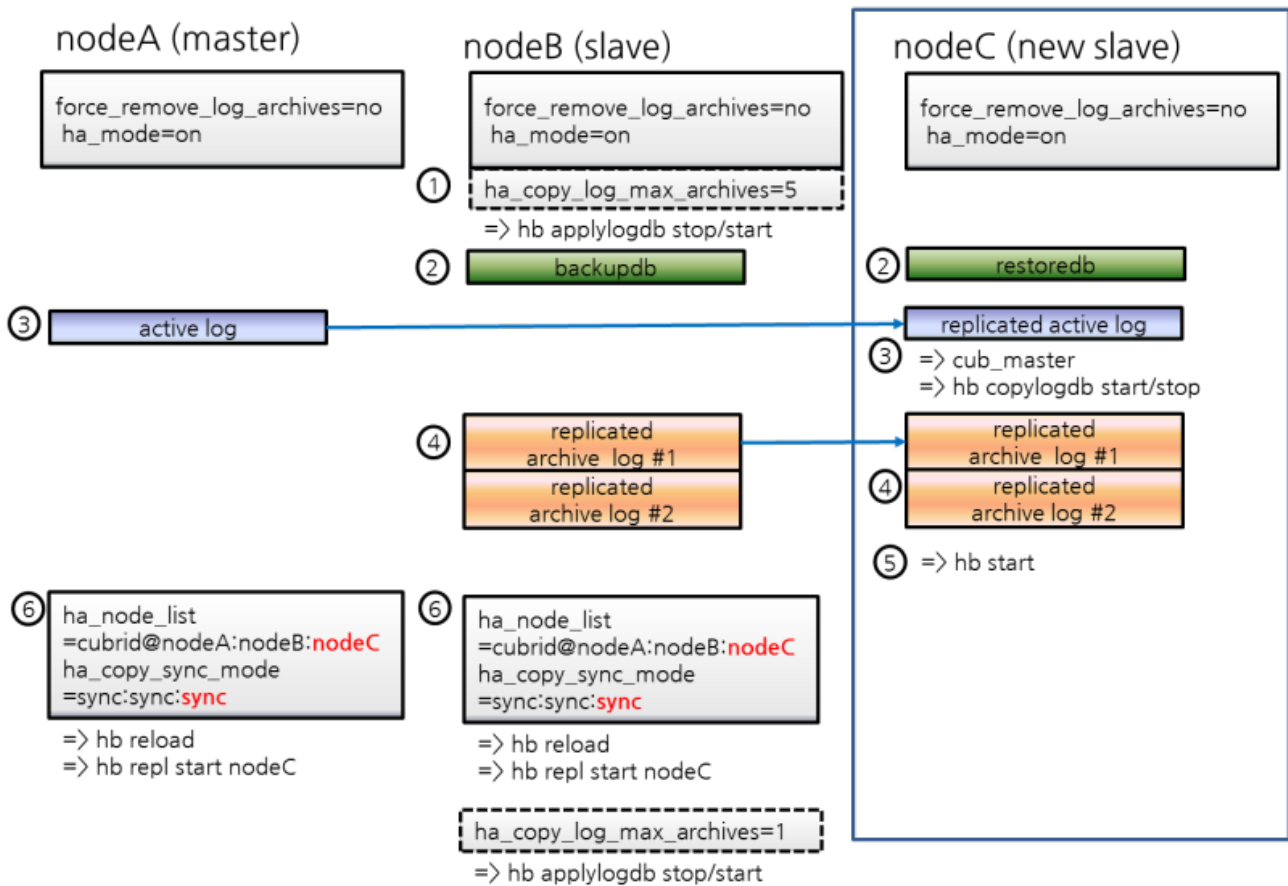
### 참고

#### 초기 데이터베이스 구축

데이터베이스는 다음 명령으로 이미 생성되어 운영되고 있다고 가정한다. createdb 수행 시 로컬 이름과 문자셋은 마스터 노드와 슬레이브 노드가 서로 동일해야 한다.

```
export CUBRID_DATABASES=/home/cubrid/DB
mkdir $CUBRID_DATABASES/testdb
mkdir $CUBRID_DATABASES/testdb/log
cd $CUBRID_DATABASES/testdb
cubrid createdb testdb -L $CUBRID_DATABASES/testdb/log en_US.utf8
```

이때 백업 파일은 저장 위치를 별도의 옵션으로 지정하지 않으면 databases.txt에 명시된 로그 디렉터리에 저장된다.



### 1. nodeC의 HA 설정

- nodeC는 nodeB와 동일하게 `$CUBRID/conf/cubrid.conf` 설정

HA 모드로 구동 중 복제되지 않은 로그가 삭제되지 않도록 하기 위해 반드시 `"force_remove_log_archives=no"`로 설정한다.

```
[service]
service=server,broker,manager

[common]
...

HA 구성 시 추가 (Logging parameters)
force_remove_log_archives=no

HA 구성 시 추가 (HA 모드)
ha_mode=on
```

- nodeC는 nodeB와 동일하게 `$CUBRID/conf/cubrid_ha.conf` 설정

단, `ha_node_list`에 nodeC를 추가하고, `ha_copy_sync_mode`에 `sync`를 하나 더 추가한다.

```
[common]
ha_port_id=59901

cubrid는 HA 시스템의 그룹 이름이고, nodeA, nodeB, nodeC는 호스트 이름.
ha_node_list=cubrid@nodeA:nodeB:nodeC

ha_db_list=testdb
ha_copy_sync_mode=sync:sync:sync
ha_apply_max_mem_size=500
```

- nodeA, nodeB와 동일하게 nodeC의 로컬 라이브러리 설정

```
[nodeB]$ scp $CUBRID/conf/cubrid_locales.txt cubrid_usr@nodeC:
$CUBRID/conf/.
[nodeC]$ make_locale.sh -t64
```

- nodeC의 **\$CUBRID\_DATABASES/databases.txt**에서 db-host에 nodeA, nodeB, nodeC 추가

| #db-name | vol-path                  | db-host                         | log-path |
|----------|---------------------------|---------------------------------|----------|
| testdb   | /home/cubrid/DB/testdb    | nodeA:nodeB:nodeC               | /        |
|          | home/cubrid/DB/testdb/log | file:/home/cubrid/DB/testdb/lob |          |

**databases.txt**의 **db-host** 부분은 브로커에서 DB 서버로의 접속 순서와 관련된 설정이므로 원하는대로 변경이 가능하다. 예를 들어, **db-host**에 nodeA:nodeB만 쓰거나, localhost만 써도 된다. 다만, 로컬에서 접속하는 경우(`csql -u dba testdb@localhost`)를 감안하여 **db-host**에 localhost 또는 로컬 호스트의 이름은 반드시 포함한다.

## 2. 백업 이후 복구 도중 nodeB의 복제 로그 삭제 방지를 위한 설정 적용

nodeB에서 백업한 이후 nodeC에 복구하는 도중에도 서비스가 계속된다면 nodeB의 복제 로그가 추가될 수 있는데, 설정에 따라 슬레이브의 복제 로그가 nodeC에 복제되기도 전에 삭제될 수 있다. 이를 방지하기 위해 다음과 같은 설정을 한다.

- nodeB에 DB 재구동 없이 **ha\_copy\_log\_max\_archives** 값을 크게 변경

nodeB에서 백업을 받은 시점부터, nodeC가 구동한 이후 데이터가 마스터와 동일해질 때까지 수행된 트랜잭션이 보관되어야, 백업 시점 이후의 트랜잭션을 nodeC에 반영할 수 있다.

따라서, 백업 이후 추가된 nodeB의 복제 로그는 유지되어야 한다.

여기서는 nodeC 구축 시간 동안 수행되는 트랜잭션을 충분히 보관할 수 있는 복제 로그 개수를 5라고 가정한다.

**\$CUBRID/conf/cubrid\_ha.conf** 편집

```
ha_copy_log_max_archives=5
```

*nodeB*의 *applylogdb* 프로세스에 **ha\_copy\_log\_max\_archives** 파라미터 변경 내역 반영

```
[nodeB]$ cubrid heartbeat applylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat applylogdb start testdb nodeA
```

#### 참고

마스터 노드인 *nodeA*만 사용하여 *nodeC*를 구축한다면 *nodeB*에서 복제 로그를 복사하는 작업은 불필요하므로, *nodeB*에서 **ha\_copy\_log\_max\_archives**를 변경하는 작업 역시 불필요하다.

대신 *nodeA*에 변경된 **log\_max\_archives**를 적용해야 한다.

*cubrid.conf*의 **log\_max\_archives** 파라미터는 “SET SYSTEM PARAMETERS” 구문만으로 서비스 중 변경이 가능하다.

```
[nodeA]$ csqll -u dba -c "SET SYSTEM PARAMETERS
'log_max_archives=5'" testdb@localhost
```

### 3. *nodeB*에서 백업, *nodeC*에 복구

#### ◦ *nodeB* 백업

```
[nodeB]$ cd $CUBRID_DATABASES/testdb/log
[nodeB]$ cubrid backupdb --sleep-msecs=10 -C -o output.txt
testdb@localhost
```

#### 참고

**--sleep-msecs**는 1MB의 백업 파일이 쓰여질 때마다 쉬는 시간을 설정하는 옵션으로, 단위는 밀리초이다. 백업할 장비의 디스크 I/O 부하가 심한 경우 이 값을 설정하는 것을 고려하되, 이 값이 클수록 백업 시간이 길어지므로 가급적이면 부하가 적은 시간대에 백업하고 이 값은 작게 설정할 것을 권장한다.

*nodeB*의 부하가 전혀 없는 상태라면 이 옵션을 생략해도 무방하다.

-o 옵션으로 명시한 파일에는 백업 결과 정보가 기록된다.

- 백업 파일을 *nodeC*에 복사

```
[nodeB]$ cd $CUBRID_DATABASES/testdb/log
[nodeB]$ scp -l 131072 testdb_bk* cubrid_usr@nodeC:
$CUBRID_DATABASES/testdb/log
```

#### 참고

scp 명령의 -l 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

- *nodeC*에서 복구를 수행

이때, 마스터 노드와 슬레이브 노드의 볼륨 경로가 반드시 같아야 한다.

```
[nodeC]$ cubrid restoredb -B $CUBRID_DATABASES/testdb/log testdb
```

#### 4. *nodeA*의 활성 로그(active log)를 *nodeC*에 복사

*nodeA*의 활성 로그에는 최근에 생성된 보관 로그의 번호 정보가 들어 있다. 활성 로그를 복사한 시점 이후에 생성되는 보관 로그가 *nodeC*의 HA 구동 후 copylogdb에 의해 자동으로 복사되려면 해당 정보가 필요하므로, 사용자는 활성 로그 복사 이전에 생성된 보관 로그만 수동으로 복사하면 된다.

- *nodeC*에서 HA 연결을 관리하는 cub\_master 구동

```
[nodeC]$ cub_master
```

- *nodeA*에 대한 copylogdb 프로세스 구동

```
[nodeC]$ cubrid heartbeat copylogdb start testdb nodeA
```

- *nodeA*의 활성 로그가 복사되었음을 확인

현 시점의 활성 로그를 복사해 온다는 것은 최근에 생성된 보관 로그 번호 정보를 획득한다는 의미이다. 이후에 생성되는 보관 로그는 [5. *nodeC*에서 HA 서비스를 구동] 과정 이후 자동으로 복사된다.

```
[nodeC]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeC]$ ls
testdb_lgar_t testdb_lgat testdb_lgat__lock
```

- *nodeC*에서 *nodeA*에 대한 copylogdb 프로세스 정지

```
[nodeC]$ cubrid heartbeat copylogdb stop testdb nodeA
```

- *nodeC*의 cub\_master 정지

```
[nodeC]$ cubrid service stop
```

## 5. *nodeC*에 데이터베이스 복구 이후 필요한 로그 파일 복사

- *nodeB*의 복제된 보관 로그(replicated archive log) 전부를 *nodeC*에 복사

```
[nodeC]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeC]$ scp -l 131072 cubrid@nodeB:$CUBRID_DATABASES/
testdb_nodeA/testdb_lgar0* .
```

이 과정에서는 복제에 필요한 복제 보관 로그가 전부 존재해야 한다는 점에 주의한다.

### 참고

scp 명령의 -l 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

### 참고

파일 이름이 숫자로 끝나는 파일만 복사한다. 여기에서는 복제 로그 파일 이름의 번호가 모두 0으로 시작하기 때문에 testdb\_lgar0\*을 복사했다.

### 참고

필요한 복제 로그가 이미 삭제된 경우 대처 방안

**ha\_copy\_log\_max\_archives**의 값이 충분히 크지 않은 경우 백업 시각과 활성 로그 생성 시각 사이에 생성된 복제 로그가 삭제될 수 있는데, 이 경우 아래 [5. nodeC에서 HA 서비스를 구동] 과정에서 오류가 발생할 수 있다.

오류가 발생한 경우, 필요한 로그가 마스터에 존재한다면 추가로 복사해서 nodeC의 HA를 재구동한다. 예를 들어, 마스터에는 보관 로그 #1, #2, #3 이 있고 슬레이브에는 복제된 보관 로그 #2, #3만 존재하며 #1이 추가로 필요하다면 마스터에서 #1을 복사해도 된다. 그러나, 마스터의 보관 로그 파일이 더 많이 남는 경우는 마스터의 **log\_max\_archives** 값이 충분히 큰 경우에만 발생할 수 있는 상황이다.

만약 마스터의 보관 로그에도 원하는 번호의 보관 로그가 없다면 **ha\_copy\_log\_max\_archives**의 값을 충분히 크게 한 후 백업 및 복구를 처음부터 다시 진행한다.

## 참고

### 필요로 하는 복제 로그 삭제 여부 확인

필요로 하는 복제 로그가 이미 삭제되었는지 여부는 testdb\_lginf 파일을 사용하여 확인할 수 있다.

1. 슬레이브에서 백업한 데이터베이스를 복원한 경우라면, 복원된 데이터베이스의 \_db\_ha\_apply\_info 카탈로그 테이블에서 required\_lsa\_pageid를 통해 마스터 데이터베이스의 어느 페이지부터 필요한지 확인할 수 있다.
2. 마스터에서 \$CUBRID\_DATABASES/testdb/log 디렉터리에 있는 testdb\_lginf 파일의 내용을 가장 아래에서부터 확인해 나간다.

```
Time: 03/16/15 17:44:23.767 - COMMENT: CUBRID/LogInfo for
database /home/cubrid/DB/databases/testdb/testdb
Time: 03/16/15 17:44:23.767 - ACTIVE: /home/cubrid/DB/
databases/testdb/log/testdb_lgat 1280 pages
Time: 03/16/15 17:54:40.892 - ARCHIVE: 0 /home/cubrid/DB/
databases/testdb/log/testdb_lgar000 0 1277
Time: 03/16/15 17:57:29.451 - COMMENT: Log archive /home/
cubrid/DB/databases/testdb/log/testdb_lgar000 is not needed
any longer unless a database media crash occurs.
Time: 03/16/15 18:03:08.166 - ARCHIVE: 1 /home/cubrid/DB/
databases/testdb/log/testdb_lgar001 1278 2555
Time: 03/16/15 18:03:08.167 - COMMENT: Log archive /home/
cubrid/DB/databases/testdb/log/testdb_lgar000, which contains
log pages before 2556, is not needed any longer by any HA
utilities.
Time: 03/16/15 18:03:29.378 - COMMENT: Log archive /home/
```



```
cubrid/DB/databases/testdb/log/testdb_lgar001 is not needed
any longer unless a database media crash occurs.
```

testdb\_lginf 파일의 내용이 위와 같이 기록되어 있을 때, “Time: 03/16/15 17:54:40.892 - ARCHIVE: 0 /home/cubrid/DB/databases/testdb/log/testdb\_lgar000 0 1277”의 뒤 두 개의 숫자는 해당 파일이 저장하고 있는 페이지의 시작 ID와 끝 ID이다.

예를 들어, \_db\_ha\_apply\_info를 통해 확인한 백업 당시 페이지 ID가 2300번이라면 “testdb\_lgar001 1278 2555”을 통해 1278번과 2555번 사이의 페이지 ID임을 확인할 수 있으므로, 복원 시 마스터의 보관 로그 (또는 슬레이브의 복제된 보관 로그)는 1번부터 필요하다는 것을 알 수 있다.

## 6. nodeC에서 HA 서비스를 구동

```
[nodeC]$ cubrid heartbeat start
```

## 7. nodeA, nodeB의 HA 설정 변경

- nodeB의 ha\_copy\_log\_max\_archive 설정 원복

```
[nodeB]$ vi cubrid_ha.conf
ha_copy_log_max_archives=1
```

- nodeB의 applylogdb 프로세스에 ha\_copy\_log\_max\_archives 파라미터 변경 내역 반영

```
[nodeB]$ cubrid heartbeat applylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat applylogdb start testdb nodeA
```

- nodeA, nodeB의 cubrid\_ha.conf에서 ha\_node\_list에 nodeC를 추가하고, ha\_copy\_sync\_mode에 sync를 하나 더 추가

```
$ cd $CUBRID/conf
$ vi cubrid_ha.conf

ha_node_list=cubrid@nodeA:nodeB:nodeC
ha_copy_sync_mode=sync:sync:sync
```

- 변경된 ha\_node\_list를 적용

```
[nodeA]$ cubrid heartbeat reload
[nodeB]$ cubrid heartbeat reload
```

- *nodeC*에 대한 *copylogdb*, *applylogdb* 구동

```
[nodeA]$ cubrid heartbeat repl start nodeC
[nodeB]$ cubrid heartbeat repl start nodeC
```

## 8. 브로커 추가 및 응용 프로그램 URL의 altHosts에 추가된 브로커의 호스트 이름 추가

필요에 따라 브로커를 추가하고, 추가된 브로커에 접속하기 위해 응용 프로그램의 URL 속성인 altHosts에 추가된 브로커의 호스트 이름을 추가한다.

브로커의 추가를 고려하지 않고 있다면 이 작업을 하지 않아도 된다.

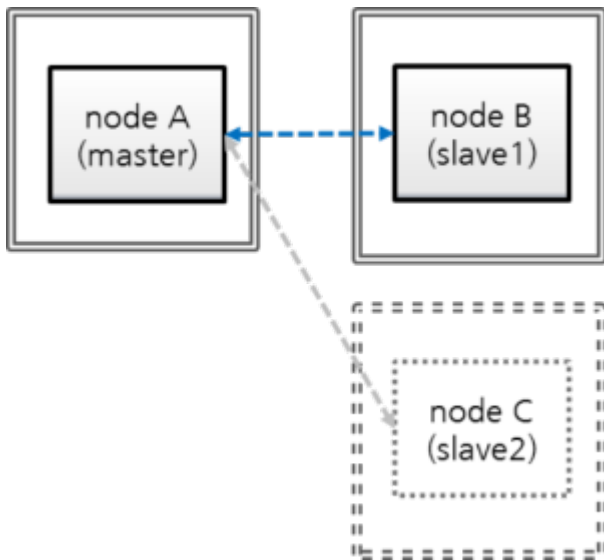
- *nodeA*, *nodeB* 모두 **\$CUBRID\_DATABASES/databases.txt**에서 db-host에 *nodeC* 추가

| #db-name | vol-path               | db-host                         | log-path                   |
|----------|------------------------|---------------------------------|----------------------------|
| testdb   | /home/cubrid/DB/testdb | nodeA:nodeB:nodeC               | /home/cubrid/DB/testdb/log |
|          |                        | file:/home/cubrid/DB/testdb/lob |                            |

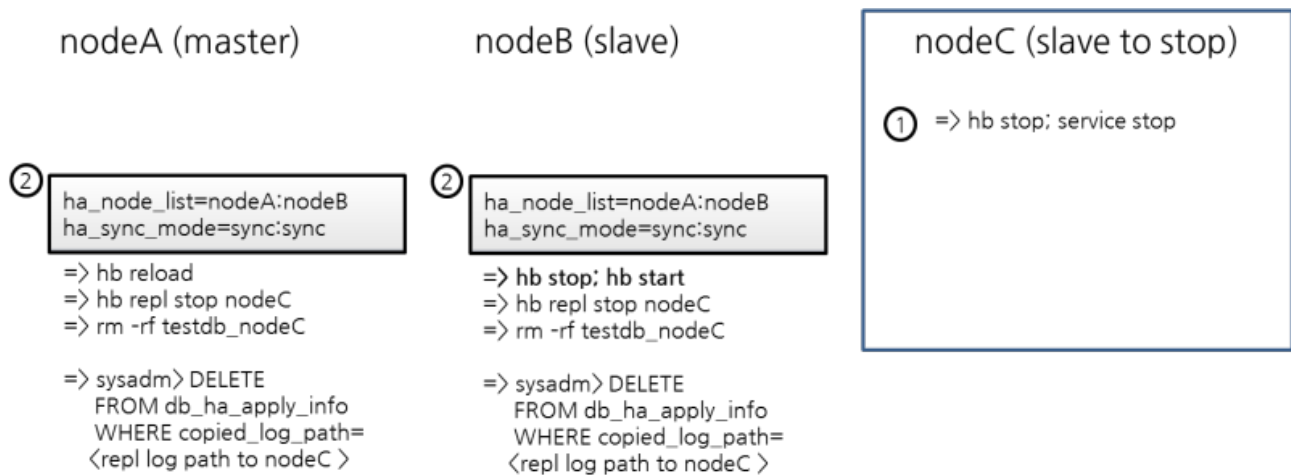
**databases.txt**의 **db-host** 부분은 브로커에서 DB 서버로의 접속 순서와 관련된 설정이므로 원하는 대로 변경이 가능하다. 예를 들어, **db-host**에 *nodeA:nodeB*만 쓰거나, *localhost*만 써도 된다. 다만, 로컬에서 접속하는 경우(*csql -u dba testdb@localhost*)를 감안하여 **db-host**에 *localhost* 또는 로컬 호스트 이름은 반드시 포함한다.

## 서비스 운영 중 슬레이브 제거

“마스터:슬레이브=1:2”로 구성된 환경에서 하나의 슬레이브를 제거해보자.



- nodeA: 마스터 노드 호스트 명
- nodeB: 슬레이브 노드 호스트 명
- nodeC: 제거할 슬레이브 노드 호스트 명



### 1. nodeC 정지

```

$ cubrid heartbeat stop # heartbeat와 관련된 프로세스들
 (applylogdb, copylogdb, cub_server) 정지
$ cubrid service stop # cub_master, cub_broker, cub_manager 등
 의 구동 정지

```

### 2. nodeA, nodeB의 설정에서 nodeC 제거

- nodeA, nodeB의 cubrid\_ha.conf에서 ha\_node\_list의 nodeC 제거,  
ha\_copy\_sync\_mode의 sync 하나 제거

```
$ vi cubrid_ha.conf

ha_node_list=cubrid@nodeA:nodeB
ha_copy_sync_mode=sync:sync
```

- 변경된 `ha_node_list`를 적용

마스터인 `nodeA`에는 “reload” 명령을 사용한다.

```
[nodeA]$ cubrid heartbeat reload
```

슬레이브인 `nodeB`에는 “stop/start” 명령을 사용한다.

```
[nodeB]$ cubrid heartbeat stop
[nodeB]$ cubrid heartbeat start
```

- `nodeC`에 대한 `copylogdb`, `applylogdb` 프로세스 정지

```
[nodeA]$ cubrid heartbeat repl stop nodeC
[nodeB]$ cubrid heartbeat repl stop nodeC
```

- `nodeA`, `nodeB`에서 `nodeC`에 대한 복제 로그 제거

```
$ cd $CUBRID_DATABASES
$ rm -rf testdb_nodeC
```

- `nodeA`, `nodeB`에서 `nodeC`에 대한 복제 정보 제거

`csql` 실행 시 **--sysadm** 옵션과 **--write-on-standby** 옵션을 주어야 슬레이브에서 `DELETE` 연산을 수행할 수 있다.

```
$ csql -u dba --sysadm --write-on-standby testdb@localhost

sysadm> DELETE FROM _db_ha_apply_info WHERE
copied_log_path='/home/cubrid/DB/databases/testdb_nodeC';
```

`copied_log_path`는 복제 로그 파일이 저장된 경로를 나타낸다.

## 1. 브로커 설정에서 `nodeC` 제거

- `nodeA`, `nodeB`의 `databases.txt`에서 `db-host`에 `nodeC`가 포함되어 있는 경우, `nodeC`를 제거

```
$ cd $CUBRID_DATABASES
$ vi databases.txt

#db-name vol-path db-host log-
path lob-base-path
testdb /home/cubrid/DB/testdb nodeA:nodeB /
home/cubrid/DB/testdb/log file:/home/cubrid/DB/testdb/lob
```

- *nodeA*, *nodeB*의 브로커 재구동

```
$ cubrid broker restart
```

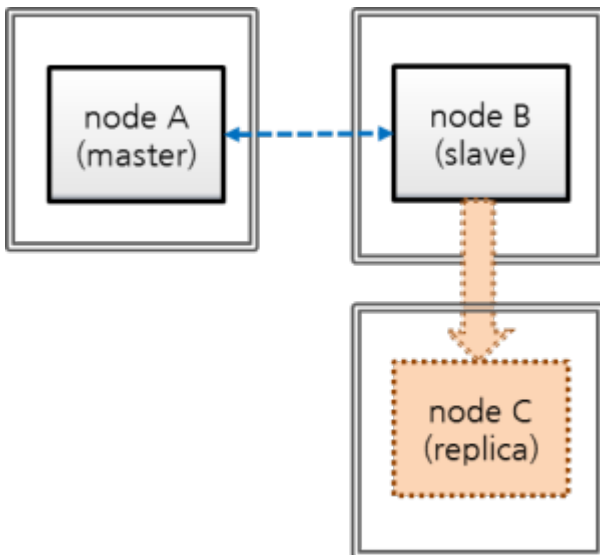
### 참고

*nodeC*가 종료된 상태이므로 응용 프로그램이 *nodeC* 데이터베이스에 접속되지는 않을 것이며, 따라서 브로커 재구동을 서두를 필요는 없다. 브로커 재구동 시 해당 브로커에 연결된 응용 프로그램의 연결이 끊어질 수 있음을 감안해야 한다. 또한, 동일한 설정의 브로커가 다중으로 존재하는 경우 하나의 브로커가 다른 브로커를 대신할 수 있으므로 브로커를 하나씩 재구동하도록 한다.

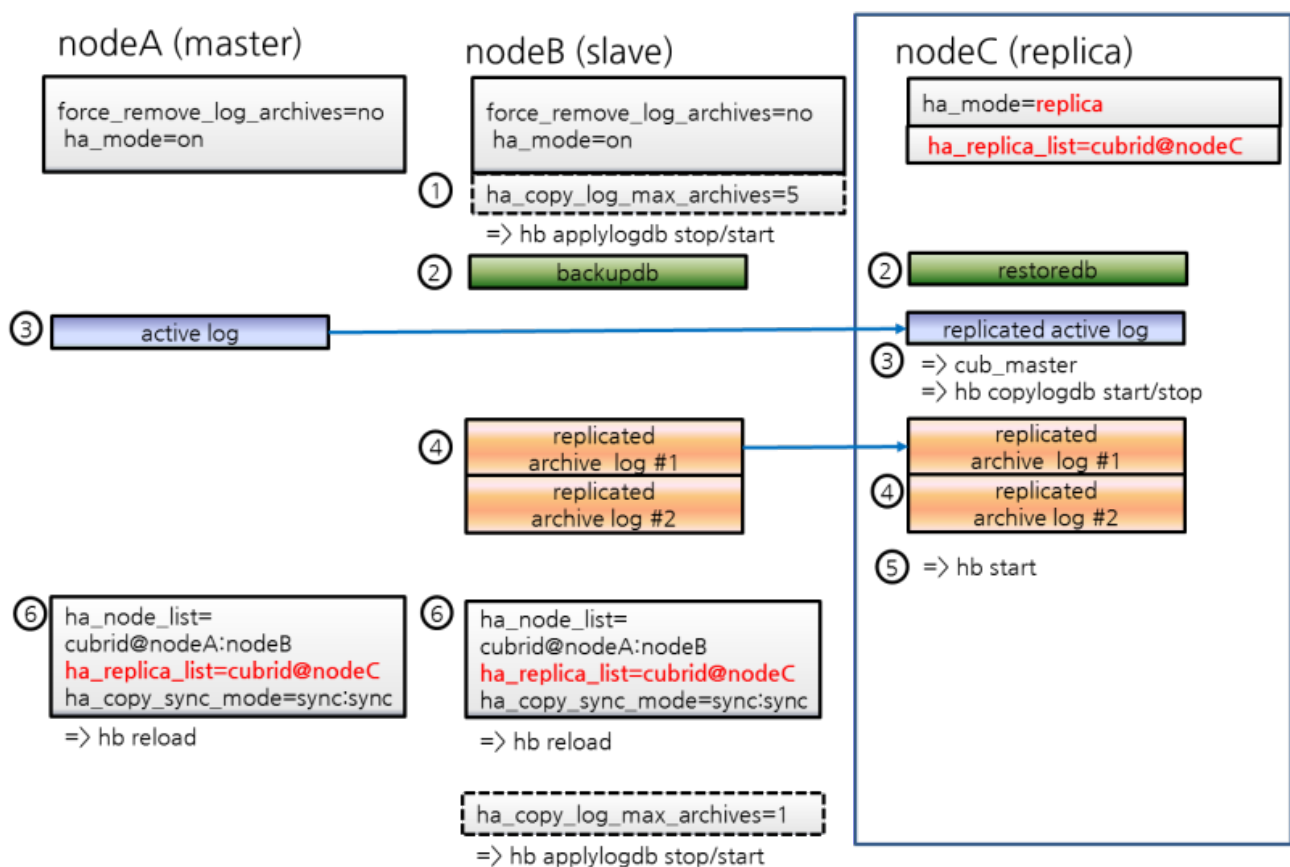
예를 들어, 위의 설정에서는 *nodeA*와 *nodeB*에 브로커가 동일하게 설정되어 있으므로, *nodeB*를 재구동한 이후 *nodeA*를 재구동하도록 한다.

## 서비스 운영 중 레플리카 추가

“마스터:슬레이브=1:1” 구성에서 슬레이브를 이용해 레플리카를 추가해 보자. 초기 데이터베이스 설정은 **서비스 운영 중 슬레이브 하나 더 추가의 초기 데이터베이스 구축**에서 설정한 것과 동일하다.



- nodeA: 마스터 노드 호스트 명
- nodeB: 슬레이브 노드 호스트 명
- nodeC: 새로 추가할 레플리카 노드 호스트 명



### 1. nodeC의 HA 설정

- nodeC는 `ha_mode`를 제외하고는 `nodeB`와 동일하게 `$CUBRID/conf/cubrid.conf` 설정

ha\_mode=replica로 설정한다.

```
[service]
service=server,broker,manager

[common]

HA 구성 시 추가 (HA 모드)
ha_mode=replica
```

### 참고

레플리카의 트랜잭션 로그는 복제에 사용되지 않으므로, force\_remove\_log\_archives의 설정이 무엇이든 항상 yes로 동작한다.

- nodeC의 **\$CUBRID/conf/cubrid\_ha.conf** 설정

ha\_replica\_list에 nodeC를 추가한다.

```
[common]
ha_port_id=59901

cubrid는 HA 시스템의 그룹 이름이고, nodeA, nodeB, nodeC는 호스트 이름임.
ha_node_list=cubrid@nodeA:nodeB

ha_replica_list=cubrid@nodeC

ha_db_list=testdb
ha_copy_sync_mode=sync:sync
ha_apply_max_mem_size=500
```

- nodeA, nodeB와 동일하게 nodeC의 로컬 라이브러리 설정

```
[nodeB]$ scp $CUBRID/conf/cubrid_locales.txt cubrid_usr@nodeC:
$CUBRID/conf/.
[nodeC]$ make_locale.sh -t64
```

- nodeC의 **\$CUBRID\_DATABASES/databases.txt**에서 db-host에 nodeC 지정

| #db-name | vol-path | db-host       | log- |
|----------|----------|---------------|------|
| path     |          | lob-base-path |      |

```
testdb /home/cubrid/DB/testdb nodeC /home/cubrid/DB/
testdb/log file:/home/cubrid/DB/testdb/lob
```

**databases.txt**의 **db-host** 부분은 브로커에서 DB 서버로의 접속 순서와 관련된 설정이므로 원하는대로 변경이 가능하다. 예를 들어, **db-host**에 nodeA:nodeB만 쓰거나, localhost만 써도 된다. 다만, 로컬에서 접속하는 경우(csql -u dba testdb@localhost)를 감안하여 **db-host**에 localhost 또는 로컬 호스트의 이름은 반드시 포함한다.

## 2. 백업 이후 복구 도중 nodeB의 복제 로그 삭제 방지를 위한 설정 적용

nodeB에서 백업한 이후 nodeC에 복구하는 도중에도 서비스가 계속된다면, 복구 이후에도 nodeB의 복제 로그가 추가될 수 있는데, 설정에 따라 nodeC에 필요한 복제 로그 일부가 삭제될 수 있다. 이를 방지하기 위해 다음과 같은 설정을 한다.

- nodeB에 DB 재구동 없이 **ha\_copy\_log\_max\_archives** 값을 크게 변경

nodeB의 백업 데이터를 복원한 nodeC의 데이터가 마스터(nodeA)와 동일해질 때까지의 트랜잭션이 보관되어야, 백업 시점 이후의 트랜잭션을 nodeC에 반영할 수 있다.

따라서, 백업 이후 추가된 nodeB의 복제 로그는 유지되어야 한다.

여기서는 nodeC 구축 시간 동안 수행되는 트랜잭션을 충분히 보관할 수 있는 복제 로그 개수를 5라고 가정한다.

### \$CUBRID/conf/cubrid\_ha.conf 편집

```
ha_copy_log_max_archives=5
```

nodeB의 applylogdb 프로세스에 ha\_copy\_log\_max\_archives 파라미터 변경 내역 반영

```
[nodeB]$ cubrid heartbeat applylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat applylogdb start testdb nodeA
```

### 참고

마스터 노드인 nodeA만 사용하여 nodeC를 구축한다면 nodeB에서 복제 로그를 복사하는 작업은 불필요하므로, nodeB에서 **ha\_copy\_log\_max\_archives**를 변경하는 작업 역시 불필요하다.

대신 nodeA에 변경된 **log\_max\_archives**를 적용하여야 한다.

cubrid.conf의 **log\_max\_archives** 파라미터는 “SET SYSTEM PARAMETERS” 구문만으로 서비스 중 변경이 가능하다.



```
[nodeA]$ csq1 -u dba -c "SET SYSTEM PARAMETERS
'log_max_archives=5'" testdb@localhost
```

### 3. nodeB의 백업 및 nodeC의 복구

#### ◦ nodeB 백업

```
[nodeB]$ cd $CUBRID_DATABASES/testdb/log
[nodeB]$ cubrid backupdb --sleep-msecs=10 -C -o output.txt
testdb@localhost
```

#### 참고

서비스 운영 중 슬레이브 하나 더 추가의 `-sleep-msecs`에 대한 설명을 참고한다.

#### ◦ 백업 파일을 nodeC에 복사

```
[nodeB]$ cd $CUBRID_DATABASES/testdb/log
[nodeB]$ scp -l 131072 testdb_bk* cubrid_usr@nodeC:
$CUBRID_DATABASES/testdb/log
```

.. note::

scp 명령의 `-l` 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

#### ◦ nodeC에서 복구를 수행

이때, 마스터 노드와 슬레이브 노드의 볼륨 경로가 반드시 같아야 한다.

```
[nodeC]$ cubrid restoredb -B $CUBRID_DATABASES/testdb/log testdb
```

### 4. nodeA의 활성 로그(active log)를 nodeC에 복사

nodeA의 활성 로그에는 최근에 생성된 보관 로그의 번호 정보가 들어 있다. 활성 로그를 복사한 시점 이후에 생성되는 보관 로그가 nodeC의 HA 구동 후 copylogdb에 의해 자동으로 복사되려면

이 정보가 필요하므로, 사용자는 활성 로그 복사 이전에 생성된 보관 로그만 수동으로 복사하면 된다.

- *nodeC*에서 HA 연결을 관리하는 *cub\_master* 구동

```
[nodeC]$ cub_master
```

- *nodeA*에 대한 *copylogdb* 프로세스 구동

```
[nodeC]$ cubrid heartbeat copylogdb start testdb nodeA
```

- *nodeA*의 활성 로그가 복사되었음을 확인

현 시점의 활성 로그를 복사해 온다는 것은 현 시점까지 생성된 보관 로그 번호 정보를 획득한다는 의미이다. 이후에 생성되는 보관 로그는 [5. *nodeC*에서 HA 서비스를 구동] 과정 이후 자동으로 복사된다.

```
[nodeC]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeC]$ ls
testdb_lgar_t testdb_lgat testdb_lgat__lock
```

- *nodeC*에서 *nodeA*에 대한 *copylogdb* 프로세스 정지

```
[nodeC]$ cubrid heartbeat copylogdb stop testdb nodeA
```

- *nodeC*의 *cub\_master* 정지

```
[nodeC]$ cubrid service stop
```

## 5. *nodeC*에 데이터베이스 복구 이후 필요한 로그 파일 복사

- *nodeB*의 복제된 보관 로그(replicated archive log) 전부를 *nodeC*에 복사

```
[nodeC]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeC]$ scp -l 131072 cubrid@nodeB:$CUBRID_DATABASES/
testdb_nodeA/testdb_lgar0* .
```

이 과정에서는 복제에 필요한 복제 보관 로그가 전부 존재해야 한다는 점에 주의한다.

### 참고

scp 명령의 -l 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

#### **i** 참고

파일 이름이 숫자로 끝나는 파일만 복사한다. 여기에서는 복제 로그 파일 이름의 번호가 모두 0으로 시작하기 때문에 testdb\_lgar0\*을 복사했다.

#### **i** 참고

**ha\_copy\_log\_max\_archives**의 값이 충분히 크지 않은 경우 백업 시각과 활성 로그 생성 시각 사이에 생성된 복제 로그가 삭제될 수 있는데, 이 경우 아래 [5. nodeC에서 HA 서비스를 구동] 과정에서 오류가 발생할 수 있다.

오류가 발생하면 위에서 설명한 **필요한 복제 로그의 삭제 여부 확인 및 대처 방안**을 참고하여 조치하도록 한다.

### 6. nodeC에서 HA 서비스를 구동

```
[nodeC]$ cubrid heartbeat start
```

### 7. nodeA, nodeB의 HA 설정 변경

- nodeB의 ha\_copy\_log\_max\_archive 설정 원복

```
[nodeB]$ vi cubrid_ha.conf
ha_copy_log_max_archives=1
```

- nodeB의 applylogdb 프로세스에 ha\_copy\_log\_max\_archives 파라미터 변경 내역 반영

```
[nodeB]$ cubrid heartbeat applylogdb stop testdb nodeA
[nodeB]$ cubrid heartbeat applylogdb start testdb nodeA
```

- nodeA, nodeB의 cubrid\_ha.conf에서 ha\_replica\_list에 nodeC를 추가

```
$ cd $CUBRID/conf
$ vi cubrid_ha.conf

ha_replica_list=cubrid@nodeC
```

- 변경된 ha\_replica\_list를 적용

```
[nodeA]$ cubrid heartbeat reload
[nodeB]$ cubrid heartbeat reload
```

nodeC는 레플리카 노드이므로, nodeA, nodeB에서 nodeC에 대한 applylogdb, copylogdb의 구동은 불필요하다.

## 8. 브로커 추가 및 응용 프로그램 URL의 altHosts에 추가된 브로커의 호스트 이름 추가

필요에 따라 브로커를 추가하고, 추가된 브로커에 접속하기 위해 응용 프로그램의 연결 속성에 추가된 브로커의 호스트 이름을 추가한다.

브로커의 추가를 고려하지 않고 있다면 이 작업을 하지 않아도 된다.

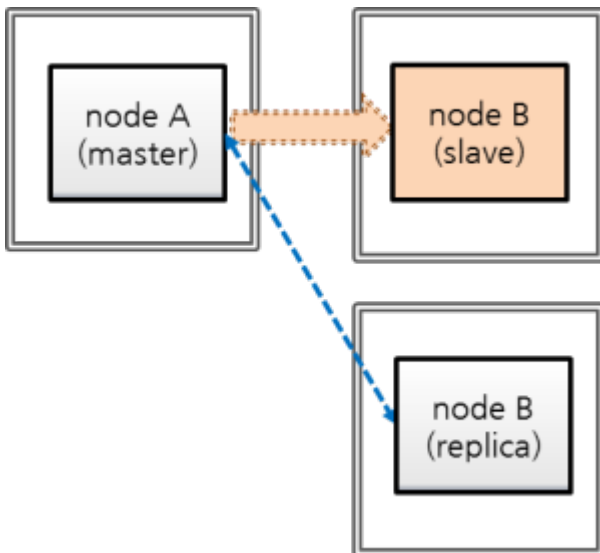
- nodeA, nodeB 모두 \$CUBRID\_DATABASES/databases.txt에서 db-host에 nodeC 추가

| #db-name | vol-path               | db-host                         | log-path                   |
|----------|------------------------|---------------------------------|----------------------------|
| testdb   | /home/cubrid/DB/testdb | nodeA:nodeB:nodeC               | /home/cubrid/DB/testdb/log |
|          |                        | file:/home/cubrid/DB/testdb/lob |                            |

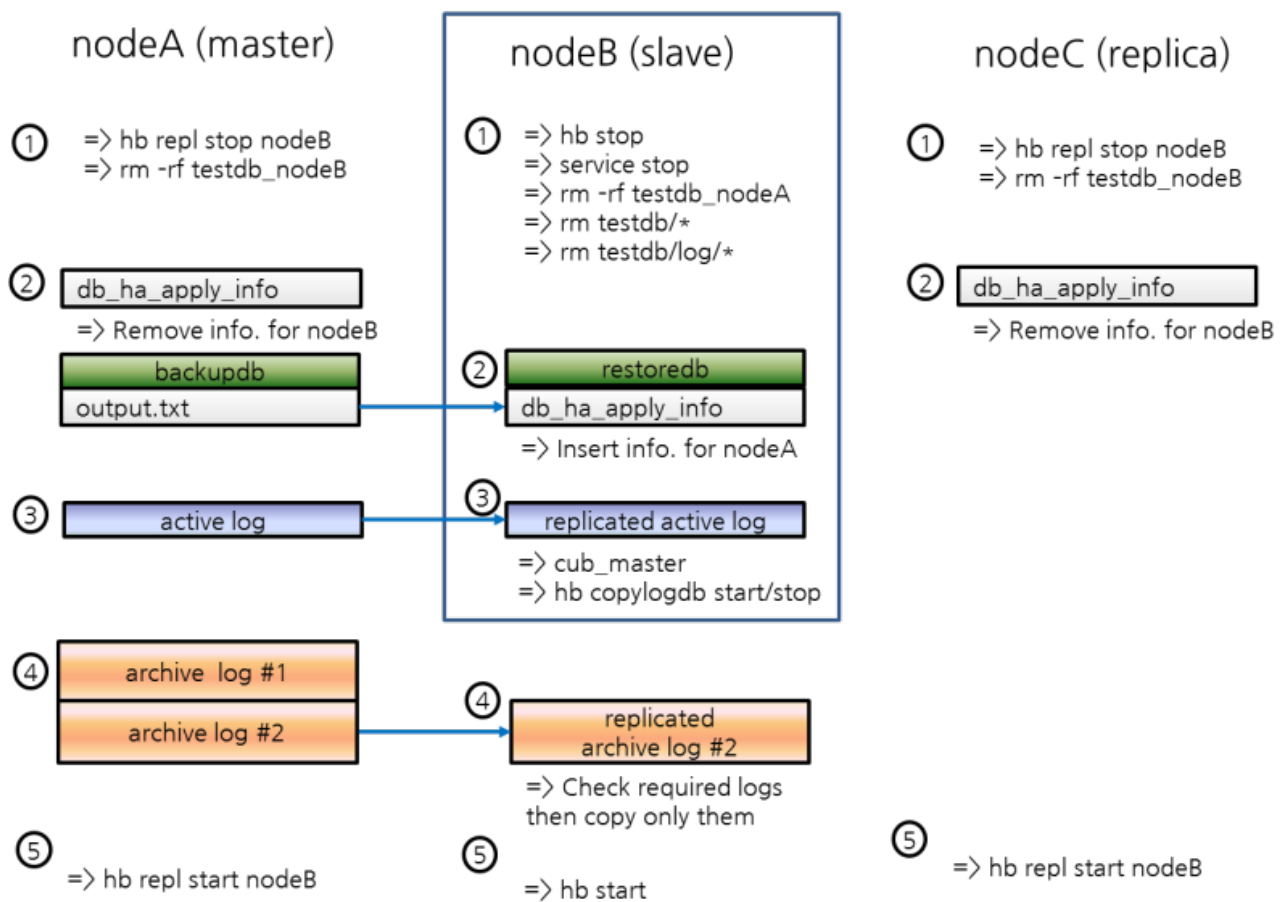
**databases.txt**의 **db-host** 부분은 브로커에서 DB 서버로의 접속 순서와 관련된 설정이므로 원하는 대로 변경이 가능하다. 예를 들어, **db-host**에 nodeA:nodeB만 쓰거나, localhost만 써도 된다. 다만, 로컬에서 접속하는 경우(csql -u dba testdb@localhost)를 감안하여 **db-host**에 localhost 또는 로컬 호스트 이름은 반드시 포함한다.

## 서비스 운영 중 슬레이브 재구축

“마스터:슬레이브:레플리카 = 1:1:1”로 구축된 환경에서 슬레이브의 데이터에 이상이 발생했다고 가정하고, 마스터로부터 데이터를 완전히 새로 구축해 보자. 초기 데이터베이스 설정은 **서비스 운영 중 슬레이브 하나 더 추가의 초기 데이터베이스 구축**에서 설정한 것과 동일하다.



- nodeA: 마스터 노드 호스트 명
- nodeB: 재구축할 슬레이브 노드 호스트 명
- nodeC: 레플리카 노드 호스트 명



#### 1. nodeB 정지, nodeB 데이터 삭제

*nodeB*를 정지한 후, *nodeB*의 데이터베이스 볼륨, *nodeA*와 *nodeC*에 있는 *nodeB*의 복제 로그를 삭제한다.

- *nodeB* 정지

```
[nodeB]$ cubrid heartbeat stop
[nodeB]$ cubrid service stop
```

- *nodeB*의 데이터베이스 볼륨, 복제 로그 삭제

```
[nodeB]$ cd $CUBRID_DATABASES
[nodeB]$ rm testdb/*
[nodeB]$ rm testdb/log/*

[nodeB]$ rm -rf testdb_nodeA
[nodeB]$ rm $CUBRID/var/APPLYLOGDB/testdb
```

- *nodeA*, *nodeC*에서 *nodeB*의 로그 복제 정지

```
[nodeA]$ cubrid heartbeat repl stop nodeB
[nodeC]$ cubrid heartbeat repl stop nodeB
```

- *nodeA*, *nodeC*에서 *nodeB*에 대한 복제 로그 삭제

```
[nodeA]$ rm -rf $CUBRID_DATABASES/testdb_nodeB
[nodeC]$ rm -rf $CUBRID_DATABASES/testdb_nodeB
```

## 2. *nodeA*의 백업 및 *nodeB*의 복구, HA 카탈로그 테이블에 정보 추가

*nodeA*, *nodeC*에서 *nodeB*에 대한 `_db_ha_apply_info` 정보를 삭제한다.

```
[nodeA]$ csql --sysadm -u dba testdb@localhost
csql> DELETE FROM _db_ha_apply_info WHERE
copied_log_path='/home/cubrid/DB/databases/testdb_nodeB';

[nodeC]$ csql --sysadm --write-on-standby -u dba
testdb@localhost
csql> DELETE FROM _db_ha_apply_info WHERE
copied_log_path='/home/cubrid/DB/databases/testdb_nodeB';
```

- *nodeA* 백업

```
[nodeA]$ cd $CUBRID_DATABASES/testdb/log
[nodeA]$ cubrid backupdb --sleep-msecs=10 -C -o output.txt
testdb@localhost
```

#### **i** 참고

서비스 운영 중 슬레이브 하나 더 추가의 `-sleep-msecs`에 대한 설명을 참고한다.

- 백업 파일을 *nodeB*에 복사

```
[nodeA]$
[nodeA]$ scp -l 131072 testdb_bk* cubrid_usr@nodeB:
$CUBRID_DATABASES/testdb/log
```

#### **i** 참고

scp 명령의 `-l` 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

- *nodeB*에서 복구를 수행

이때, 마스터 노드와 슬레이브 노드의 볼륨 절대 경로가 반드시 같아야 한다.

```
[nodeB]$ cubrid restoredb -B $CUBRID_DATABASES/testdb/log testdb
```

- *nodeB*의 `_db_ha_apply_info`에 *nodeA*에 대한 복제 정보를 추가

*nodeA*에 있는 백업 결과를 저장한 `output.txt` 파일에서 `_db_ha_apply_info`를 업데이트할 정보를 얻는다. `output.txt`는 “cubrid backupdb” 명령을 수행한 디렉터리에 저장된다.

```
[nodeA]$ vi $CUBRID_DATABASES/testdb/log/output.txt

[Database(testdb) Full Backup start]

- num-threads: 2
- compression method: NONE
- sleep 10 millisecond per 1M read.
```

```
- backup start time: Fri Mar 20 18:18:53 2015
- number of permanent volumes: 2
- HA apply info: testdb 1426495463 12922 16192
- backup progress status
...
```

아래 스크립트를 만든 후 이를 수행한다. 위의 출력 정보 중 “HA apply info”에서 첫 번째 숫자인 1426495463은 \$db\_creation에, 두 번째 숫자인 12922는 \$pageid에, 세 번째 숫자인 16192는 \$offset에 넣고, db\_name에는 데이터베이스 이름인 testdb, master\_host에는 마스터 노드의 호스트 이름인 nodeA를 넣는다.

```
[nodeB]$ vi update.sh

#!/bin/sh

db_creation=1426495463
page_id=12922
offset=16192
db_name=testdb
master_host=nodeA

repl_log_home_abs=$CUBRID_DATABASES

repl_log_path=$repl_log_home_abs/${db_name}_${master_host}

local_db_creation=`awk 'BEGIN { print
strftime("%m/%d/%Y %H:%M:%S", '$db_creation') }'`
csql_cmd="\
INSERT INTO \
 _db_ha_apply_info \
VALUES \
(\
 '$db_name', \
 datetime '$local_db_creation', \
 '$repl_log_path', \
 $page_id, $offset, \
 $page_id, $offset, \
 $page_id, $offset, \
 $page_id, $offset, \
 $page_id, $offset, \
 $page_id, $offset, \
 NULL, \
 NULL, \
 NULL, \
 0, \
```



```

0, \
0, \
0, \
0, \
0, \
0, \
NULL \
)"

```

```

Insert nodeA's HA info.
csql --sysadm -u dba -c "$csql_cmd" -S testdb

```

```
[nodeB]$ sh update.sh
```

입력이 제대로 되었는지 확인한다.

```

[nodeB]$ csql -u dba -S testdb

csql> ;line on
csql> SELECT * FROM _db_ha_apply_info;

```

### 3. nodeA의 활성 로그(active log)를 nodeB에 복사

nodeA의 활성 로그에는 최근에 생성된 보관 로그의 번호 정보가 들어 있다. 활성 로그를 복사한 시점 이후에 생성되는 보관 로그가 nodeB의 HA 구동 후 copylogdb에 의해 자동으로 복사되려면 해당 정보가 필요하므로, 사용자는 활성 로그 복사 이전에 생성된 보관 로그만 수동으로 복사하면 된다.

- nodeB에서 HA 연결을 관리하는 cub\_master 구동

```
[nodeB]$ cub_master
```

- nodeB에서 nodeA에 대한 copylogdb 프로세스 구동

```
[nodeB]$ cubrid heartbeat copylogdb start testdb nodeA
```

- nodeB에서 nodeA의 활성 로그가 복사되었음을 확인

현 시점의 활성 로그를 복사해 온다는 것은 현 시점까지 생성된 보관 로그 번호 정보를 획득한다는 의미이다. 이후에 생성되는 보관 로그는 [5. nodeB에서 HA 서비스를 구동] 과정 이후 자동으로 복사된다.

```
[nodeB]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeB]$ ls
testdb_lgar_t testdb_lgat testdb_lgat__lock
```

- *nodeB*에서 *nodeA*에 대한 copylogdb 프로세스 정지

```
[nodeB]$ cubrid heartbeat copylogdb stop testdb nodeA
```

- *nodeB*의 cub\_master 정지

```
[nodeB]$ cubrid service stop
```

#### 4. *nodeB*에 데이터베이스 복구 이후 필요한 로그 파일 복사

- *nodeA*의 보관 로그 전부를 *nodeB*에 복사

```
[nodeB]$ cd $CUBRID_DATABASES/testdb_nodeA
[nodeB]$ scp -l 131072 cubrid@nodeA:$CUBRID_DATABASES/testdb/log/
testdb_lgar0* .
```

이 과정에서는 복제에 필요한 보관 로그가 전부 존재해야 한다는 점에 주의한다.

#### 참고

scp 명령의 -l 옵션은 복사량을 조절하는 옵션으로, 노드 간 파일 복사 시 I/O 부하를 감안하여 이 옵션을 적절히 부여하도록 한다. 단위는 Kbits이며, 131072는 16MB이다.

#### 참고

파일 이름이 숫자로 끝나는 파일만 복사한다. 위의 예에서는 보관 로그가 모두 0으로 시작하기 때문에 testdb\_lgar0\*을 복사했다.

#### 참고

**필요한 로그가 이미 삭제된 경우**

**log\_max\_archives**의 값이 충분히 크지 않은 경우 백업 시각과 활성 로그 생성 시각 사이에 생성된 보관 로그가 삭제될 수 있는데, 이 경우 아래 [5. nodeB에서 HA 서비스를 구동 ...] 과정에서 오류가 발생할 수 있다.

이 경우 **log\_max\_archives**의 값을 충분히 크게 한 후 백업 및 복구를 처음부터 다시 진행한다.

5. nodeB에서 HA 서비스를 구동하고, nodeA, nodeC에서 로그 복제 및 반영 프로세스 재구동

```
[nodeB]$ cubrid heartbeat start
```

```
[nodeA]$ cubrid heartbeat applylogdb start testdb nodeB
[nodeA]$ cubrid heartbeat copylogdb start testdb nodeB
```

```
[nodeC]$ cubrid heartbeat applylogdb start testdb nodeB
[nodeC]$ cubrid heartbeat copylogdb start testdb nodeB
```

## 복제 불일치 감지

### 복제 불일치 감지 방법

마스터 노드와 슬레이브 노드(또는 레플리카 노드)의 데이터가 일치하지 않는 복제 노드 간 데이터 불일치 현상은 다음과 같은 과정을 통해 어느 정도 감지할 수 있다. 또한 **checksumdb** 유틸리티를 사용해 복제 불일치를 감지할 수도 있다. 그러나 마스터 노드와 슬레이브 노드(또는 레플리카 노드)의 데이터를 서로 직접 비교해보는 방법보다 더 정확한 확인 방법은 없다. 복제 불일치 상태라는 판단이 될 경우, 마스터 노드의 데이터베이스를 슬레이브 노드(또는 레플리카 노드)에 새로 구축해야 한다.( **복제 재구축 스크립트** 참고).

- **cubrid statdump** 명령을 수행하여 **Time\_ha\_replication\_delay** 시간을 확인한다. 이 값이 클 수록 복제 지연 정도가 클 수 있다는 것을 의미하며, 지연된 시간만큼 복제 불일치가 존재할 가능성이 커진다.
- 슬레이브 노드(또는 레플리카 노드)에서 **cubrid applyinfo** 를 실행하여 “Fail count” 값을 확인한다. “Fail count”가 0이면, 복제에 실패한 트랜잭션이 없다고 볼 수 있다(**applyinfo** 참고).

```
[nodeB]$ cubrid applyinfo -L /home/cubrid/DB/testdb_nodeA -r nodeA -a testdb
```

```
*** Applied Info. ***
Committed page : 1913 | 2904
Insert count : 645
```

```
Update count : 0
Delete count : 0
Schema count : 60
Commit count : 15
Fail count : 0
...
```

- 슬레이브 노드(또는 레플리카 노드)에서 복제 로그의 복사 지연 여부를 확인하기 위해, **cubrid applyinfo** 를 실행하여 “Copied Active Info.”의 “Append LSA” 값과 “Active Info.”의 “Append LSA” 값을 비교한다. 이 값이 큰 차이를 보인다면, 복제 로그가 슬레이브 노드(또는 레플리카 노드)에 복사되는데 지연이 있다는 의미이다([applyinfo](#) 참고).

```
[nodeB]$ cubrid applyinfo -L /home/cubrid/DB/testdb_nodeA -r nodeA -a testdb
```

```
...
```

```
*** Copied Active Info. ***
```

```
DB name : testdb
DB creation time : 11:28:00.000 AM 12/17/2010
(1292552880)
Vol creation time : 11:28:10.000 AM 12/17/2010
(1292552890)
EOF LSA : 1913 | 2976
Append LSA : 1913 | 2976
HA server state : active
```

```
*** Active Info. ***
```

```
DB name : testdb
DB creation time : 11:28:00.000 AM 12/17/2010
(1292552880)
Vol creation time : 11:28:10.000 AM 12/17/2010
(1292552890)
EOF LSA : 1913 | 2976
Append LSA : 1913 | 2976
HA server state : active
```

- 복제 로그 복사 지연이 의심되는 경우 네트워크 회선 속도가 느려졌는지, 디스크 여유 공간이 충분한지, 디스크 I/O에는 이상이 없는지 등을 확인한다.
- 슬레이브 노드(또는 레플리카 노드)에서 복제 로그의 반영 지연 여부를 확인하기 위해, **cubrid applyinfo** 를 실행하여 “Applied Info.” 의 “Committed page” 값과 “Copied Active Info.”의 “EOF LSA” 값을 비교한다. 이 값이 큰 차이를 보인다면, 복제 로그가 슬레이브(또는 레플리카) 데이터베이스에 반영되는데 지연이 있다는 의미이다([applyinfo](#) 참고).

```
[nodeB]$ cubrid applyinfo -L /home/cubrid/DB/testdb_nodeA -r nodeA -a testdb
```

```

*** Applied Info. ***
Committed page : 1913 | 2904
Insert count : 645
Update count : 0
Delete count : 0
Schema count : 60
Commit count : 15
Fail count : 0

*** Copied Active Info. ***
DB name : testdb
DB creation time : 11:28:00.000 AM 12/17/2010
(1292552880)
Vol creation time : 11:28:10.000 AM 12/17/2010
(1292552890)
EOF LSA : 1913 | 2976
Append LSA : 1913 | 2976
HA server state : active
...

```

- 복제 로그 반영 지연이 심한 경우 수행 시간이 긴 트랜잭션을 의심해 볼 수 있는데, 해당 트랜잭션의 수행이 정상이라면 복제 지연 역시 정상적으로 발생할 수 있다. 정상 여부를 판단하기 위해 **cubrid applyinfo** 를 지속적으로 수행하면서 applylogdb가 복제 로그를 슬레이브 노드(또는 레플리카 노드)에 계속 반영하고 있는지 확인해야 한다.
- copylogdb, applylogdb 프로세스가 생성한 오류 로그의 메시지를 확인한다(오류 메시지 참고).
- 마스터 데이터베이스 테이블의 레코드 개수, 슬레이브(또는 레플리카) 데이터베이스 테이블의 레코드 개수를 비교한다.

## checksumdb

**checksumdb** 를 통해 간단하게 복제 무결성을 확인할 수 있다. 기본적으로 이 유틸리티는 마스터 노드의 각 테이블을 청크(chunk)로 분할한 후 CRC32 값을 계산한다. 계산 값이 아닌 계산 방법이 CUBRID HA를 통해 복제된다. 결과적으로 마스터 노드와 슬레이브 노드(또는 레플리카 노드)에서 계산된 CRC32 값을 비교함으로써 **checksumdb** 는 복제 무결성 여부를 확인할 수 있다. **checksumdb** 는 성능 저하를 최소화하도록 설계되었으나 마스터의 성능에 영향을 줄 수 있어 사용시 유의해야 한다.

```
cubrid checksumdb [options] <database-name>@<hostname>
```

- *<hostname>* : 체크섬(checksum) 계산을 시작할 때 마스터 노드의 호스트명을 지정해야 한다. 계산이 완료된 후 결과를 가져올 때 확인할 노드의 호스트명을 지정한다.

### -c, --chunk-size=NUMBER

CRC32 계산에 사용할 행 수를 지정한다. (기본값: 500행, 최소값: 100행)

**-s, --sleep=NUMBER**

체크를 계산하는 도중 checksumdb가 쉬는 시간을 설정한다.(기본값: 100ms)

**-i, --include-class-file=FILE**

-i 옵션을 지정해 복제 불일치를 확인할 테이블을 지정한다. 테이블을 지정하지 않으면 전체 테이블을 확인한다. 파일 내용의 테이블명 구분자로 사용할 수 있는 기호는 빈 문자열, 탭, 개행문자 및 쉼표이다.

**-e, --exclude-class-file=FILE**

-e 옵션을 지정하여 복제 불일치 확인에서 제외할 테이블을 지정한다. -i와 -e 중 하나만 사용할 수 있다.

**-t, --timeout=NUMBER**

이 옵션으로 계산 시간 제한을 지정한다. (기본값: 1000ms) 이 시간 제한에 도달하면 계산이 취소되고 잠시 후에 다시 시작된다.

**-n, --table-name=STRING**

체크섬 결과를 저장할 테이블명을 지정한다. 테이블명 입력 시 “소유자명.테이블명” 형식을 사용해야 하며, 현재는 소유자명으로 dba만 지정할 수 있다. (기본값: dba.db\_ha\_checksum)

**-r, --report-only**

이 옵션을 통해 checksum 계산이 완료된 후에 결과를 얻을 수 있다.

**--resume**

checksum 계산이 중지되었을 경우, 이 옵션을 이용해서 다시 실행할 수 있다.

**--schema-only**

이 옵션을 이용해서 CRC32 계산을 하지 않고 각 테이블의 스키마만을 비교할 수 있다.

**--cont-on-error**

이 옵션이 없으면 에러 발생시 checksumdb가 중지된다.

다음은 checksumdb를 실행하는 예제이다.

```
cubrid checksumdb -c 100 -s 10 testdb@master
```

복제 불일치가 발견되지 않았을 경우

```
$ cubrid checksumdb -r testdb@slave
=====
target DB: testdb@slave (state: standby)
report time: 2016-01-14 16:33:30
checksum table name: db_ha_checksum, db_ha_checksum_schema
=====

different table schema
```

```

NONE

table name diff chunk id chunk lower bound

NONE

table name total # of chunks # of diff chunks total/
avg/min/max time

t1 7 0 88 /
12 / 5 / 14 (ms)
t2 7 0 96 /
13 / 11 / 15 (ms)

```

테이블 *t1* 에서 복제 불일치가 감지되었을 경우

```

$ cubrid checksumdb -r testdb@slave
=====
target DB: testdb@slave (state: standby)
report time: 2016-01-14 16:35:57
checksum table name: db_ha_checksum, db_ha_checksum_schema
=====

different table schema

NONE

table name diff chunk id chunk lower bound

t1 0 (id>=1)
t1 1 (id>=100)
t1 4 (id>=397)

table name total # of chunks # of diff chunks total/
avg/min/max time

t1 7 3 86 /
12 / 5 / 14 (ms)
t2 7 0 93 /
13 / 11 / 15 (ms)

```

테이블 *t1* 에서 스키마 불일치가 감지되었을 경우

```
$ cubrid checksumdb -r testdb@slave
=====
target DB: testdb@slave (state: standby)
report time: 2016-01-14 16:40:56
checksum table name: db_ha_checksum, db_ha_checksum_schema
=====

different table schema

<table name>
t1
<current schema - collected at 04:40:53.947 PM 01/14/2016>
CREATE TABLE [t1] ([id] INTEGER NOT NULL, [col1] CHARACTER
VARYING(20), [col2] INTEGER, [col3] DATETIME, [col4] INTEGER,
CONSTRAINT [pk_t1_id] PRIMARY KEY ([id])) COLLATE iso88591_bin
<schema from master>
CREATE TABLE [t1] ([id] INTEGER NOT NULL, [col1] CHARACTER
VARYING(20), [col2] INTEGER, [col3] DATETIME, CONSTRAINT [pk_t1_id]
PRIMARY KEY ([id])) COLLATE iso88591_bin

* Due to schema inconsistency, the checksum difference of the above
table(s) may not be reported.

table name diff chunk id chunk lower bound

NONE

table name total # of chunks # of diff chunks total/
avg/min/max time

t1 7 0 95 /
13 / 11 / 16 (ms)
t2 7 0 94 /
13 / 11 / 15 (ms)
```

## HA 오류 메시지

다음은 복제 불일치 발생의 원인이 될 수 있는 오류에 대한 오류 메시지들을 정리한 것이다.



## CAS 프로세스(cub\_cas)

CAS 프로세스의 오류 메시지는

**\$CUBRID/log/broker/error\_log/**

**<broker\_name>\_<app\_server\_num>.err**에 기록된다. 아래는 HA 환경에서 접속 시 발생하는 오류 메시지를 별도로 정리한 것이다.

## 브로커와 DB 서버 간 handshake 오류 메시지

| 오류 코드 | 오류 메시지                                                                              | severity | 설명                                                        | 조치 사항 |
|-------|-------------------------------------------------------------------------------------|----------|-----------------------------------------------------------|-------|
| -1139 | Handshake error (peer host ?): incompatible read/write mode. (client: ?, server: ?) | error    | 브로커 ACCESS_MODE 와 서버의 상태 (active/standby) 불일치 (브로커 모드 참고) |       |
| -1140 | Handshake error (peer host ?): HA replication delayed.                              | error    | <b>ha_delay_limit</b> 을 설정한 서버에서 복제 지연 발생                 |       |
| -1141 | Handshake error (peer host ?): replica-only client to non-replica server.           | error    | 레플리카만 접속 가능한 브로커(CAS)가 레플리카가 아닌 서버에 접속 시도 (브로커 모드 참고)     |       |
| -1142 | Handshake error (peer host ?): remote access to server not allowed.                 | error    | HA maintenance 모드인 서버에 원격 접속 시도 (서버 참고)                   |       |

| 오류 코드 | 오류 메시지                                                      | severity | 설명           | 조치 사항 |
|-------|-------------------------------------------------------------|----------|--------------|-------|
| -1143 | Handshake error (peer host ?): unidentified server version. | error    | 서버 버전 알 수 없음 |       |

### 연결 오류 메시지

| 오류 코드 | 오류 메시지                                                       | severity | 설명                   | 조치 사항 |
|-------|--------------------------------------------------------------|----------|----------------------|-------|
| -353  | Cannot make connection to master server on host ?.           | error    | cub_master 프로세스 down |       |
| -1144 | Timed out attempting to connect to ?.<br>(timeout: ? sec(s)) | error    | 장비 down              |       |

### 복제 로그 복사 프로세스(copylogdb)

복제 로그 복사 프로세스의 오류 메시지는 **\$CUBRID/log/<db-name>@<remote-node-name>\_copylogdb.err**에 기록된다. 복제 로그 복사 프로세스에서 남을 수 있는 오류 메시지의 severity는 fatal, error, notification이며 기본 severity는 error이다. 따라서 notification 오류 메시지를 남기려면 **cubrid.conf**의 **error\_log\_level** 값을 변경해야 한다. 이에 대한 자세한 설명은 [오류 메시지 관련 파라미터](#)를 참고한다.

### 초기화 오류 메시지

복제 로그 복사 프로세스의 초기화 단계에서 남을 수 있는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                                                                        | severity | 설명                                                         | 조치 사항                                                                                            |
|-------|-----------------------------------------------------------------------------------------------|----------|------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| -10   | Unable to mount disk volume ?.                                                                | error    | 복제 로그 파일 열기 실패                                             | 복제 로그 존재 여부를 확인한다. 복제 로그의 위치는 <b>기본 환경 설정</b> 을 참고한다.                                            |
| -78   | Internal error: an I/O error occurred while reading logical log page ? (physical page ?) of ? | fatal    | 복제 로그 읽기 실패                                                | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다.                                                           |
| -81   | Internal error: logical log page ? may be corrupted.                                          | fatal    | 복제 로그 복사 프로세스가 연결된 데이터베이스 서버 프로세스로부터 복사한 복사한 복제 로그 페이지의 오류 | 복제 로그 복사 프로세스가 연결된 데이터베이스 서버 프로세스의 오류 로그를 확인한다. 오류 로그를 확인한다. 이 오류 로그는 \$CUBRID/log/server에 위치한다. |
| -1039 | log writer: log writer started. mode: ?                                                       | error    | 복제 로그 복사 프로세스가 초기화 성공하여 정상 시작                              | 이 오류 메시지는 복제 로그 복사 프로세스의 시작 정보를 나타내기 위해 기록되는 것이므로 조치 사항은 없다. 복제 로그 복사 프로세스가 시작된 이후 이 오류 메시지      |

| 오류 코드 | 오류 메시지 | severity | 설명 | 조치 사항                                          |
|-------|--------|----------|----|------------------------------------------------|
|       |        |          |    | 가 나오기 전까지의 오류 메시지는 정상 상황에서 발생할 수 있는 것이므로 무시한다. |

### 복제 로그 요청 및 수신 오류 메시지

복제 로그 복사 프로세스는 연결된 데이터베이스 서버 프로세스에 복제 로그를 요청하고 적절한 복제 로그를 수신한다. 이때 발생하는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                       | severity | 설명                                         | 조치 사항                                                                                                     |
|-------|----------------------------------------------|----------|--------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| -89   | Log ? does not belong to the given database. | error    | 기존에 복제되었던 로그와 현재 복제하려는 로그가 다른 로그가 다름       | 복제 로그 복사 프로세스가 연결한 데이터베이스 서버/호스트 정보를 확인한다. 연결하려는 데이터베이스 서버/호스트 정보를 변경해야 하는 경우 기존 복제 로그를 삭제하여 초기화하고 재시작한다. |
| -186  | Error receiving data from server.            | error    | 복제 로그 복사 프로세스가 연결된 데이터베이스 서버로부터 잘못된 정보를 수신 | 내부적으로 복구된다.                                                                                               |
| -199  |                                              | error    | 복제 로그 복사 프로세스가 연결                          | 내부적으로 복구된다.                                                                                               |

| 오류 코드 | 오류 메시지                       | severity | 설명                   | 조치 사항 |
|-------|------------------------------|----------|----------------------|-------|
|       | Server no longer responding. |          | 된 데이터베이스 서버로부터 연결 종료 |       |

### 복제 로그 쓰기 오류 메시지

복제 로그 복사 프로세스는 연결된 데이터베이스 서버 프로세스로부터 수신한 복제 로그를 **cubrid\_ha.conf** 에서 지정한 위치(**ha\_copy\_log\_base**)에 복사한다. 이때 발생하는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                                                                                                                                        | severity | 설명                         | 조치 사항                      |
|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|----------------------------|----------------------------|
| -10   | Unable to mount disk volume ?.                                                                                                                                | error    | 복제 로그 파일 열기 실패             | 복제 로그 유무를 확인한다.            |
| -79   | Internal error: an I/O error occurred writing logical log page ? (physical page ?) of ?.                                                                      | fatal    | 복제 로그 쓰기 실패                | 내부적으로 복구된다.                |
| -80   | An error occurred due to insufficient space in ating system device while writing logical log page ?(physical page ?) of ?. Up to ? bytes in size are allowed. | fatal    | 파일 시스템 공간 부족으로 복제 로그 쓰기 실패 | 디스크 파티션 내 여유 공간이 있는지 확인한다. |

## 복제 로그 아카이브 오류 메시지

복제 로그 복사 프로세스는 연결된 데이터베이스 서버 프로세스로부터 받은 복제 로그를 일정한 주기마다 아카이브(archive)하여 보관하게 된다. 이때 발생하는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                                                                         | severity     | 설명                 | 조치 사항                                  |
|-------|------------------------------------------------------------------------------------------------|--------------|--------------------|----------------------------------------|
| -78   | Internal error: an I/O error occurred while reading logical log page ? (physical page ?) of ?. | fatal        | 아카이브 중 복제 로그 읽기 실패 | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -79   | Internal error: an I/O error occurred while writing logical log page ? (physical page ?) of ?. | fatal        | 아카이브 로그 쓰기 실패      | 내부적으로 복구된다.                            |
| -81   | Internal error: logical log page ? may be corrupted.                                           | fatal        | 아카이브 중 복제 로그 오류 발견 | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -98   | Unable to create archive log ? to archive pages from ? to ?.                                   | fatal        | 아카이브 로그 파일 생성 실패   | 디스크 파티션 내 여유 공간이 있는지 확인한다.             |
| -974  | Archive log ? is created to archive pages from ? to ?.                                         | notification | 아카이브 로그 파일 정보      | 이 오류 메시지는 생성된 아카이브 로그 정보를 위해 기록되       |

| 오류 코드 | 오류 메시지 | severity | 설명 | 조치 사항             |
|-------|--------|----------|----|-------------------|
|       |        |          |    | 는 것이므로 조치 사항은 없다. |

### 종료 및 재시작 오류 메시지

복제 로그 복사 프로세스가 종료 및 재시작 시에 발생하는 오류 메시지는 다음과 같다.

| 오류 코드 | 오류 메시지                                             | severity | 설명                            | 조치 사항       |
|-------|----------------------------------------------------|----------|-------------------------------|-------------|
| -1037 | log writer: log writer shut itself down by signal. | error    | 지정된 시그널에 의해 copylogdb 프로세스 종료 | 내부적으로 복구된다. |

### 복제 로그 반영 프로세스(applylogdb)

복제 로그 반영 프로세스의 오류 메시지는 **\$CUBRID/log/db-name@local-node-name\_applylogdb\_db-name\_remote-node-name.err** 에 기록된다. 복제 로그 반영 프로세스에서 남을 수 있는 오류 메시지의 severity는 fatal, error, notification이며 기본 severity는 error이다. 따라서 notification 오류 메시지를 남기려면 **cubrid.conf** 의 **error\_log\_level** 값을 변경해야 한다. 이에 대한 자세한 설명은 [오류 메시지 관련 파라미터](#) 를 참고한다.

### 초기화 오류 메시지

복제 로그 반영 프로세스의 초기화 단계에서 남을 수 있는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                     | severity | 설명                                   | 조치 사항                                       |
|-------|--------------------------------------------|----------|--------------------------------------|---------------------------------------------|
| -10   | Unable to mount disk volume ?.             | error    | 동일한 복제 로그를 반영하려는 applylogdb가 이미 실행 중 | 동일한 복제 로그를 반영하려는 applylogdb 프로세스가 있는지 확인한다. |
| -1038 | log applier: log applier started. required | error    | applylogdb 초기화 성공 후 정상 시작 나타내        | 이 오류 메시지는 복제 로그 반영                          |

| 오류 코드 | 오류 메시지                                                                | severity | 설명                        | 조치 사항          |
|-------|-----------------------------------------------------------------------|----------|---------------------------|----------------|
|       | LSA: ? ?. last committed<br>LSA: ? ?. last committed repl<br>LSA: ? ? |          | 기 위해 기록되는 것이므로 조치 사항은 없다. | 영 프로세스의 시작 정보를 |

### 로그 분석 오류 메시지

복제 로그 반영 프로세스는 복제 로그 복사 프로세스에 의해 복사된 복제 로그를 읽어 분석하고 이를 반영한다. 복제 로그를 분석할 때 발생하는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                                     | severity | 설명                                          | 조치 사항                                          |
|-------|------------------------------------------------------------|----------|---------------------------------------------|------------------------------------------------|
| -13   | An I/O error occurred while reading page ? of volume ?.    | error    | 복제 반영할 로그 페이지 읽기 실패                         | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다.         |
| -17   | Internal error: fetching deallocated pageid ? of volume ?. | fatal    | 복제 로그에 포함되지 않은 로그 페이지를 읽기 시도                | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다.         |
| -81   | Internal error: logical log page ? may be corrupted.       | fatal    | 기존 복제 반영 중이던 로그와 현재 로그가 불일치 또는 복제 로그 레코드 오류 | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다.         |
| -82   | Unable to mount log disk volume/file ?.                    | error    | 복제 로그 파일이 존재하지 않음                           | 복제 로그 존재 유무를 확인한다. cubrid applyinfo 유틸리티를 통해 복 |



| 오류 코드 | 오류 메시지                                                                                                                                           | severity | 설명                      | 조치 사항                                  |
|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|----------|-------------------------|----------------------------------------|
|       |                                                                                                                                                  |          |                         | 제 로그를 확인한다.                            |
| -97   | Internal error: unable to find log page ? in log archives.                                                                                       | error    | 로그 페이지가 복제 로그에 존재하지 않음  | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -897  | Decompression failed.                                                                                                                            | error    | 로그 레코드 압축 해제 실패         | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -1028 | log applier: unexpected EOF record in archive log. LSA: ? ?.                                                                                     | error    | 아카이브 로그에 잘못된 로그 레코드가 포함 | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -1029 | log applier: invalid replication log page/offset. page HDR: ? ?, final: ? ?, append LSA: ? ?, EOF LSA: ? ?, ha file status: ?, is end-of-log: ?. | error    | 잘못된 로그 레코드가 포함          | cubrid applyinfo 유틸리티를 통해 복제 로그를 확인한다. |
| -1030 | log applier: invalid replication record. LSA: ? ?,                                                                                               | error    | 로그 레코드 헤더 오류            | cubrid applyinfo 유틸리티를 통해 복            |

| 오류 코드 | 오류 메시지                                                                       | severity | 설명 | 조치 사항           |
|-------|------------------------------------------------------------------------------|----------|----|-----------------|
|       | forw LSA: ? ?,<br>backw LSA: ? ?,<br>Trid: ?, prev tran<br>LSA: ? ?, type: ? |          |    | 제 로그를 확인<br>한다. |

### 복제 로그 반영 오류 메시지

복제 로그 반영 프로세스는 복제 로그 복사 프로세스에 의해 복사된 복제 로그를 읽어 분석하고 이를 반영한다. 분석한 복제 로그를 반영할 때 발생하는 오류 메시지는 아래와 같다.

| 오류 코드 | 오류 메시지                                                                      | severity | 설명                                                | 조치 사항       |
|-------|-----------------------------------------------------------------------------|----------|---------------------------------------------------|-------------|
| -72   | The transaction (index ?, ?@? ?) has been cancelled by system.              | error    | 데드락 등에 의해 복제 반영 실패                                | 내부적으로 복구된다. |
| -111  | Your transaction has been cancelled due to server failure or a mode change. | error    | 복제를 반영하려는 데이터베이스 서버 프로세스 종료 또는 모드 변경에 의해 복제 반영 실패 | 내부적으로 복구된다. |
| -191  | Cannot connect to server ? on ?.                                            | error    | 복제를 반영하려는 데이터베이스 서버 프로세스와의 연결 종료                  | 내부적으로 복구된다. |
| -195  | Server communication error: ?.                                              | error    | 복제를 반영하려는 데이터베이스 서버 프로세                           | 내부적으로 복구된다. |

| 오류 코드 | 오류 메시지                                                                                             | severity | 설명                               | 조치 사항                                  |
|-------|----------------------------------------------------------------------------------------------------|----------|----------------------------------|----------------------------------------|
|       |                                                                                                    |          | 스와의 연결 종료                        |                                        |
| -224  | The database has not been resumed.                                                                 | error    | 복제를 반영하려는 데이터베이스 서버 프로세스와의 연결 종료 | 내부적으로 복구된다.                            |
| -1027 | log applier: Failed to change the reflection status from ? to ?.                                   | error    | 복제 반영 상태 변경 실패                   | 내부적으로 복구된다.                            |
| -1031 | log applier: Failed to reflect the Schema replication log. class: ?, schema: ?, internal error: ?. | error    | SCHEMA 복제 반영 실패                  | 복제 불일치 여부를 확인하고 불일치 시 HA 복제 재구성을 실행한다. |
| -1032 | log applier: Failed to reflect the Insert replication log. class: ?, key: ?, internal error: ?.    | error    | INSERT 복제 반영 실패                  | 복제 불일치 여부를 확인하고 불일치 시 HA 복제 재구성을 실행한다. |
| -1033 | log applier: Failed to reflect the Update                                                          | error    | UPDATE 복제 반영 실패                  | 복제 불일치 여부를 확인하고 불일치 시 HA 복제            |

| 오류 코드 | 오류 메시지                                                                                                            | severity     | 설명                                  | 조치 사항                                     |
|-------|-------------------------------------------------------------------------------------------------------------------|--------------|-------------------------------------|-------------------------------------------|
|       | replication log.<br>class: ?, key: ?,<br>internal error: ?.                                                       |              |                                     | 제 재구성을 실행한다.                              |
| -1034 | log applier:<br>Failed to<br>reflect the<br>Delete<br>replication log.<br>class: ?, key: ?,<br>internal error: ?. | error        | DELETE 복제 반영 실패                     | 복제 불일치 여부를 확인하고 불일치 시 HA 복제 재구성을 실행한다.    |
| -1040 | HA generic: ?.                                                                                                    | notification | 아카이브 로그의 마지막 레코드를 반영하거나 복제 반영 상태 변경 | 이 에러 메시지는 일반적인 정보를 위해 기록되는 로그로 조치 사항은 없다. |

### 종료 및 재시작 오류 메시지

복제 로그 반영 프로세스가 종료 및 재시작 시에 발생하는 오류 메시지는 다음과 같다.

| 오류 코드 | 오류 메시지                                                                                                                                                                                                    | severity | 설명                                 | 조치 사항       |
|-------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|------------------------------------|-------------|
| -1035 | log applier:<br>The memory<br>size (? MB) of<br>the log applier<br>is larger than<br>the maximum<br>memory size (?<br>MB), or is<br>doubled the<br>starting<br>memory size (?<br>MB) or more.<br>required | error    | 최대 메모리 크기 제한에 의해 복제 로그 반영 프로세스 재시작 | 내부적으로 복구된다. |

| 오류 코드 | 오류 메시지                                            | severity | 설명                           | 조치 사항       |
|-------|---------------------------------------------------|----------|------------------------------|-------------|
|       | LSA: ? ?. last committed LSA: ? ?.                |          |                              |             |
| -1036 | log applier: log applier is terminated by signal. | error    | 지정된 시그널에 의해 복제 로그 반영 프로세스 종료 | 내부적으로 복구된다. |

## Fail-over 원인 분석

### Fail-over, Fail-back 메시지

HA 환경에서 fail-over와 fail-back과 관련된 메시지는 각각 [Failover], [Failback] 접두어를 포함한다. 이 메시지는 fail-over, fail-back이 결정될 때 남겨지는 진단 로그 메시지와, fail-over 및 fail-back이 취소되거나, 성공적으로 완료될 때 기록되는 결과 로그 메시지로 나눌 수 있다.

진단 로그 메시지는 fail-over 또는 fail-back이 시작된 원인을 기록하며, [Diagnosis] 접두어를 포함하고, 결과 로그 메시지는 fail-over, fail-back이 취소된 경우 [Cancelled] 접두어와 함께 원인을 기록하며, fail-over, fail-back이 성공적으로 완료된 경우 [Success] 접두어와 함께 결과를 기록한다.

Fail-over, Fail-back 메시지를 통해 Fail-over의 원인을 분석할 수 있다. 아래는 fail-over가 발생하는 상황과 그에 따라 기록될 수 있는 fail-over, fail-back 메시지의 목록이다. Fail-over 메시지는 슬레이브 노드에 기록되고, Fail-back 메시지는 마스터 노드에 기록된다.

## 마스터 노드의 고립

### 마스터 노드의 고립

마스터 노드가 모든 슬레이브 노드로부터 heartbeat 메시지를 수신하지 못하여 네트워크 고립이 되었다고 판단될 때, 마스터 노드의 fail-back과 슬레이브 노드의 fail-over가 결정된다. **ha\_ping\_hosts**에 등록된 호스트를 통해 실제로 마스터 노드의 네트워크가 파티션 되었는지를 확인하고, 실제 마스터 노드의 네트워크 파티션이 확인된 경우에만 fail-back이 성공적으로 완료된다.

슬레이브 노드 또한 **ha\_ping\_hosts**를 통해 ping 체크를 한 결과, 현재 네트워크 파티션이 없는 경우에만 fail-over가 성공적으로 완료된다. 아래는 마스터 노드가 고립된 상황에서 로깅될 수 있는 fail-over, fail-back 메시지의 목록이다.

| 진단 메시지                                                                                                                                                         | 설명                                                                                       |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| [Failback] [Diagnosis] The master node has failed to receive heartbeat messages from all other slave nodes, resulting in a network partition.                  | 마스터 노드가 모든 슬레이브 노드로 부터 heartbeat 메시지를 수신하지 못하여 fail-back이 결정됨                            |
| [Failover] [Diagnosis] The current node has failed to receive heartbeat messages from the master node (master_node_name).                                      | 마스터 노드로 부터 heartbeat 메시지를 수신하지 못하여 슬레이브 노드의 fail-over가 결정됨                               |
| 결과 메시지                                                                                                                                                         | 설명                                                                                       |
| [Failback] [Cancelled] Ping check succeeded for the hosts registered in ha_ping_hosts, determining that it is not a network partition.                         | 마스터 노드가 ping 체크에 성공하여 네트워크 파티션이 없다고 판단하고, fail-back이 취소됨                                 |
| [Failback] [Cancelled] No hosts are registered in ha_ping_hosts, or all registered hosts are invalid, making it impossible to determine the network partition. | ha_ping_hosts에 등록된 호스트가 없거나 유효하지 않아서, 마스터 노드의 네트워크 파티션 여부를 판단하는 것이 불가능하기에 fail-back이 취소됨 |
| [Failback] [Success] Current node has been successfully demoted to slave.                                                                                      | Fail-back이 성공적으로 완료되어, 마스터 노드가 슬레이브 노드로 변경됨                                              |
| [Failover] [Cancelled] Ping check has been failed to all hosts registered in ha_ping_hosts, indicating a network partition.                                    | 슬레이브 노드가 ping check 실패로 네트워크 파티션이 있다고 판단하고, 마스터 역할을 수행할 수 없기 때문에 fail-over가 취소됨          |
| [Failover] [Success] Current node has been successfully promoted to master.                                                                                    | Fail-over가 성공적으로 완료되어, 슬레이브 노드가 마스터 노드로 변경됨                                              |

## 마스터 노드의 기능 상실

### 마스터 노드의 기능 상실

마스터 노드가 **cub\_server** 재구동의 실패나, 디스크 이상 등의 이유로 인해 마스터 노드로서의 기능을 상실할 때, 마스터 노드의 fail-back과 슬레이브 노드의 fail-over가 결정된다. 디스크 이상이라고 판단된 경우, 서버 에러 로그에 “Disk failure has been occurred: prev\_eof\_lsa(4096, 128), eof\_lsa(4096, 128)”와 같은 에러 메시지를 남긴다.

슬레이브 노드는 **ha\_ping\_hosts** 를 통해 ping 체크를 한 결과, 현재 네트워크 파티션이 없는 경우에만 fail-over가 성공적으로 완료된다. 아래는 마스터 노드의 기능 상실 상황에서 로깅될 수 있는 fail-over, fail-back 메시지의 목록이다.

| 진단 메시지                                                                                                                           | 설명                                                                                 |
|----------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| [Failback] [Diagnosis] The master node failed to restart the server process.                                                     | 마스터 노드가 비정상 종료된 cub_server 프로세스를 재구동하는데 실패하여 fail-back이 결정됨                        |
| [Failback] [Diagnosis] The master node has lost its role due to server process problem, such as disk failure.                    | 마스터 노드가 disk 이상 등의 사유로 마스터 노드로서의 기능을 상실하여 fail-back이 결정됨                           |
| [Failover] [Diagnosis] The master node (master_node_name) has lost its role due to server process problem, such as disk failure. | 슬레이브 노드는 마스터 노드가 disk 이상 등의 사유로 마스터 노드로서의 기능을 상실하였다고 판단하고, 슬레이브 노드의 fail-over를 결정함 |
| 결과 메시지                                                                                                                           | 설명                                                                                 |
| [Failback] [Success] Current node has been successfully demoted to slave.                                                        | Fail-back이 성공적으로 완료되어, 마스터 노드가 슬레이브 노드로 변경됨                                        |
| [Failover] [Cancelled] Ping check has been failed to all hosts registered in ha_ping_hosts, indicating a network partition.      | 슬레이브 노드가 ping check에 실패하여 네트워크 파티션이 있다고 판단하고, fail-over가 취소됨                       |
| [Failover] [Success] Current node has been successfully promoted to master.                                                      | Fail-over가 성공적으로 완료되어, 슬레이브 노드가 마스터 노드로 변경됨                                        |

## 복수의 마스터 노드 탐지 (split-brain)

### 복수의 마스터 노드 탐지 (split-brain)

HA 환경에서 복제 구성된 두 개 이상의 서버 모두 마스터 역할을 맡게 되는 비정상적인 상황이 발생하는 것을 split-brain 라고 한다. 이러한 상황에서 마스터 노드 역할을 수행하기 위한 하나의 노드를 제외한 다른 마스터 노드는 split-brain 상황을 해소하기 위해 슬레이브 노드로 fail-back 된다. Split-brain 상황에 남을 수 있는 메시지는 아래와 같다.

| 진단 메시지                                                                                            | 설명                                                             |
|---------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| [Failback] [Diagnosis] Multiple master nodes (master_node_name1, master_node_name2) are detected. | 복수의 마스터 노드가 감지되어서 split-brain 을 해소하기 위해 마스터 노드의 fail-back이 결정됨 |
| 결과 메시지                                                                                            | 설명                                                             |
| [Failback] [Success] Current node has been successfully demoted to slave.                         | Fail-back이 성공적으로 완료되어, 마스터 노드가 슬레이브 노드로 변경됨                    |

## 복제 재구축 스크립트

CUBRID HA 환경에서의 복제 재구축은 다중 슬레이브 노드의 다중 장애 상황이나 일반적인 오류 상황으로 인해 CUBRID HA 그룹 내의 데이터가 동일하지 않은 경우에 필요하다. **cubrid applyinfo** 유틸리티는 복제 진행 상태를 확인할 수는 있지만 이를 통해 복제 불일치 여부를 직접 판단할 수는 없으므로, 복제 불일치 여부를 정확하게 판단하려면 마스터 노드와 슬레이브 노드의 데이터를 직접 확인해야 한다.

마스터-슬레이브로 구성된 환경에서 슬레이브 노드의 이상으로 인해 슬레이브 노드만 재구축하는 경우, 그 절차는 다음과 같다.

1. 마스터 데이터베이스 백업
2. 슬레이브에서 데이터베이스 복구
3. 백업 시점을 슬레이브의 HA 메타 테이블(\_db\_ha\_apply\_info)에 저장
4. 슬레이브에서 HA 서비스 구동 (cubrid hb start)



복제 재구축을 위해서는 마스터 노드, 슬레이브 노드, 레플리카 노드에서 아래 환경이 동일해야 한다.

- CUBRID 버전
- 환경 변수(\$CUBRID, \$CUBRID\_DATABASES, \$LD\_LIBRARY\_PATH, \$PATH, \$CUBRID\_TMP, \$TMPDIR)
- 데이터베이스 볼륨, 로그 및 복제 로그 경로
- 리눅스 서버의 사용자 아이디 및 비밀번호
- **ha\_mode**, **ha\_copy\_sync\_mode**, **ha\_ping\_hosts** 를 제외한 모든 HA 관련 파라미터

다음의 경우에 한하여 **ha\_make\_slavedb.sh** 스크립트를 실행하여 복제 재구축이 가능하다.

- *from-master-to-slave*
- *from-slave-to-replica*
- *from-replica-to-replica*
- *from-replica-to-slave*

그 이외의 경우에는 수동으로 구축해야 하며, 수동으로 구축하는 시나리오는 **복제 구축**을 참고한다.

재구축이 아닌 신규 구축인 경우 cubrid.conf, cubrid\_ha.conf, databases.txt의 파일들을 마스터 노드와 동일하게 설정하면 된다.

다음 설명에서는 복제 재구축에 사용되는 **ha\_make\_slavedb.sh** 스크립트를 사용할 수 있는 사례들에 대해 알아볼 것이다.

참고로, 다중 슬레이브 노드를 구축하고자 하는 경우 **ha\_make\_slavedb.sh** 스크립트를 사용할 수 없다.

## ha\_make\_slavedb.sh 스크립트

**ha\_make\_slavedb.sh** 스크립트를 이용하여 복제 재구축을 수행할 수 있다. 이 스크립트는 **\$CUBRID/share/scripts/ha**에 위치하며, 복제 재구축에 들어가기 전에 다음의 항목을 사용자 환경에 맞게 설정해야 한다. 이 스크립트는 2008 R2.2 Patch 9 버전부터 지원하지만 2008 R4.1 Patch 2 미만 버전과는 일부 설정 방법이 다르며, 이 문서에서는 2008 R4.1 Patch 2 이상 버전에서의 설정 방법에 대해 설명한다.

- **target\_host** : 복제 재구축을 위한 원본 노드(주로 마스터 노드)의 호스트명으로, **/etc/hosts**에 등록되어 있어야 한다. 슬레이브 노드는 마스터 노드 또는 레플리카 노드를 원본으로 하여 복제 재구축이 가능하며, 레플리카 노드는 슬레이브 노드 또는 또 다른 레플리카 노드를 원본으로 하여 복제 재구축이 가능하다.

- **repl\_log\_home** : 마스터 노드의 복제 로그의 홈 디렉터리를 설정한다. 일반적으로 **\$CUBRID\_DATABASES** 와 동일하다. 반드시 절대 경로를 입력해야 하며, 심볼릭 링크를 사용하면 안 된다. 경로 뒤에 슬래시(/)를 붙이면 안 된다.

다음은 필요에 따라 선택적으로 설정하는 항목이다.

- **db\_name** : 복제 재구축할 데이터베이스 이름을 설정한다. 설정하지 않으면 **\$CUBRID/conf/cubrid-ha.conf** 내 **ha\_db\_list** 의 가장 처음에 위치한 이름을 사용한다.
- **backup\_dest\_path** : 복제 원본 노드에서 **backupdb** 수행 시 백업 볼륨을 생성할 경로를 설정한다.
- **backup\_option** : 복제 원본 노드에서 **backupdb** 수행 시 필요한 옵션을 설정한다.
- **restore\_option** : 복제 대상 노드에서 **restoredb** 수행 시 필요한 옵션을 설정한다.
- **scp\_option** : 복제 원본 노드의 백업 볼륨을 복제 대상 노드로 복사해 오기 위한 **scp** 옵션을 설정할 수 있는 항목으로 기본값은 복제 원본 노드의 네트워크 부하를 주지 않기 위해 **-l 131072** 옵션을 사용한다(전송 속도를 16M로 제한).
- **ssh\_port** : 스크립트에서 사용되는 ssh와 scp를 위한 **port** 번호를 설정할 수 있는 항목으로 기본값은 **22** 로 설정된다. 이 옵션은 스크립트에서 이용되는 **expect** 에도 동일하게 적용된다.

스크립트의 설정이 끝나면 **ha\_make\_slavedb.sh** 스크립트를 복제 대상 노드에서 수행한다. 스크립트 수행 시 여러 단계에 의해 복제 재구축이 이루어지며 각 단계의 진행을 위해서 사용자가 적절한 값을 입력해야 한다. 다음은 입력할 수 있는 값에 대한 설명이다.

- **yes** : 계속 진행한다.
- **no** : 현재 단계를 포함하여 이후 과정을 진행하지 않는다.
- **skip** : 현재 단계를 수행하지 않고 다음 단계를 진행한다. 이 입력 값은 이전 스크립트 수행에 실패하여 재시도할 때 다시 수행할 필요가 없는 단계를 무시하기 위해 사용한다.

#### ⚠ 경고

- **ha\_make\_slavedb.sh** 스크립트는 **expect**와 **ssh**를 이용하여 원격 노드에 접속 명령을 수행하므로 **expect** 명령이 설치(**root** 계정에서 **yum install expect**)되어 있어야 하고, 원격 **ssh** 접속이 가능해야 한다.

#### i 참고

- 복제 재구축에는 백업 볼륨이 필요

복제 재구축을 수행하려면 원본 노드에 있는 데이터베이스 볼륨의 물리적 이미지를 복제 대상 노드의 데이터베이스에 복사해야 한다. 그런데 **cubrid unloaddb** 는 논리적인 이미지를 백업하므로 **cubrid unloaddb** 와 **cubrid loaddb**를 이용해서는 복제 재구축을 할 수 없다. **cubrid backupdb** 는 물리적 이미지를 백업하므로 이를 이용한 복제 재구축이 가능하며, **ha\_make\_slavedb.sh** 스크립트는 **cubrid backupdb** 를 이용하여 복제 재구축을 수행한다.

#### · 복제 재구축 도중 원본 노드의 온라인 백업 및 재구축되는 노드로의 복구

**ha\_make\_slavedb.sh** 스크립트는 수행 도중 원본 노드에 대해 온라인 백업을 수행하고, 재구축되는 노드에 복구를 수행한다. 백업을 시작한 이후 추가로 진행된 트랜잭션들을 복구되는 노드에 반영하기 위해 **ha\_make\_slavedb.sh** 스크립트는 “마스터” 노드의 보관 로그를 복사해서 사용한다(슬레이브에서 레플리카를 구축할 때, 레플리카에서 레플리카를 구축할 때, 그리고 레플리카에서 슬레이브를 구축할 때 모두 “마스터”의 보관 로그를 사용).

따라서 온라인 백업이 진행되는 동안 원본 노드에 추가되는 보관 로그가 삭제되지 않도록 마스터 노드에서 **cubrid.conf**의 **force\_remove\_log\_max\_archives**, **log\_max\_archives**를 적절히 설정해야 한다. 자세한 내용은 아래의 구축 예들을 참고한다.

#### · 복제 재구축 스크립트 수행 중 오류 발생

복제 재구축 스크립트는 수행 도중 오류가 발생해도 이전 상황으로 자동 롤백되지 않는다. 이는 복제 재구축 스크립트를 수행하기 전에도 복제 대상 노드가 이미 정상적으로 서비스하기 힘든 상황이기 때문이다. 복제 재구축 스크립트를 수행하기 전 상황으로 돌아가려면, 복제 재구축 스크립트를 수행하기 전에 복제 원본 노드와 복제 대상 노드의 내부 카탈로그인 **\_db\_ha\_apply\_info** 정보와 기존의 복제 로그를 백업해야 한다.

## CUBRID 보안

이 장에서는 사용자 데이터베이스를 보호하기 위해서 CUBRID에서 제공하는 패킷 암호화, 서버 접근 제어, 권한 관리 및 TDE(Transparent Data Encryption)의 보안 기능에 대해 설명한다.

### 패킷 암호화

#### 패킷 암호화 필요성

클라이언트와 서버 간에 암호화 연결을 사용하지 않을 경우, 네트워크에 접근 가능한 해커가 모든 트래픽을 감시하고 클라이언트와 서버 간에 주고받는 데이터를 탈취하여 악용할 수 있다. 이렇게 제3자(중간자)가 정보를 탈취하여 악용하는 것을 중간자 (MITM, Man in the Middle) 공격이라 한다. 이 중간자 공격은 클라이언트와 서버간 연결에 보안 인증 절차를 추가하여 방지할 수 있다.

## 패킷 암호화 방법

큐브리드는 클라이언트와 서버 간에 전송되는 데이터를 암호화 하기 위해 SSL/TLS (Secure Socket Layer/Transport Layer Security) 프로토콜을 사용한다. 큐브리드 서버는 암호화를 위해 OpenSSL을 사용하였으며, 클라이언트는 JDBC, CCI Driver를 이용하여 암호화 연결을 할 수 있다. 클라이언트와 서버 간에 지원하는 암호화 프로토콜은 SSLv3, TLSv1, TLSv1.1, TLSv1.2 이다.

### 참고

#### SSL/TLS

SSL 란 네트워크를 통해 통신하는 클라이언트와 서버 간의 인증 및 데이터 암호화를 제공하는 암호화 프로토콜로 Netscape에 의해 처음 개발 되었다. Netscape는 1996년 보안 결함을 개선한 3.0 버전을 릴리즈하였다. SSL 3.0 버전은 TLS 1.0의 기초가 되고, 1999년 1월 IETF에서 [RFC2246](#) 표준 규약으로 정의되었으며, 마지막 갱신은 [RFC5246](#) 이다. TLS는 SSL 3.0 을 기반으로 정의되었기 때문에 SSL 3.0과 거의 유사하다.

SSL/TLS 은 서버 인증(Server Authentication), 클라이언트 인증(Client Authentication) 그리고 데이터 암호화(Data Encryption) 기능을 제공한다. 인증(Authentication)은 상대방이 맞는지 확인하는 절차를 의미하며, 암호화는 데이터를 탈취 하더라도 내용을 열람할 수 없게 하는 것을 의미한다.

## 패킷 암호화를 위한 서버 설정

### 암호화 모드 및 비암호화 모드 설정

CUBRID는 암호화 모드와 비암호화 모드를 모두 지원하며, 기본값은 비암호화 모드이다. 암호화 모드로 변경 하려면 cubrid\_broker.conf 파일의 SSL 파라미터 값을 설정해야 하며, 설정 변경 후에는 반드시 브로커를 재 시작해야 한다. 자세한 설정 방법은 [브로커 설정](#)을 참조한다.

### 인증서 (Certificate) 와 개인키 (Private Key)

SSL은 대칭키(Symmetric Key) 방식으로 송수신 데이터를 암호화한다. 즉, 클라이언트와 서버는 동일한 세션 키를 공유하여 데이터를 암호화한다. 이 세션 키는 클라이언트가 서버에 연결할 때마다 새롭게 생성 되며, 이 키를 안전하게 교환하기 위해 비대칭키(Asymmetric Key) 방식의 암호화를 사용한다.. 이를 위해 서버는 공개키와 개인키를 사용하게 된다.

서버의 공개키는 인증서에 포함되어 있으며, 이 인증서와 개인키는 `$CUBRID/conf` 디렉터리에 저장되어 있다. 각각의 파일명은 다음과 같다.

- 인증서 파일: cas\_ssl\_cert.crt
- 개인키 파일: cas\_ssl\_cert.key

이 인증서는 OpenSSL의 명령어 도구를 이용해 생성된 Self-signed 인증서이다.

사용자는 필요에 따라 IdenTrust, DigiCert 등 공인 인증 기관(CA)에서 발급받은 인증서로 대체할 수 있으며, OpenSSL을 이용해 새로 생성한 인증서를 사용하는 것도 가능하다.

다음은 OpenSSL 명령어 도구를 사용하여 개인키와 인증서를 생성하는 예시입니다:

```
$ openssl genrsa -out my_cert.key
2048 # 2048 bit 크기의
RSA 개인키 생성
$ openssl req -new -key my_cert.key -out
my_cert.csr # 인증요청서 CSR
(Certificate Signing Request)
$ openssl x509 -req -days 365 -in my_cert.csr -signkey my_cert.key -
out my_cert.crt # 1년 유효한 인증서 생성
```

위에서 생성된 my\_cert.key 와 my\_cert.crt 를 각각 \$CUBRID/conf/cas\_ssl\_cert.key와 \$CUBRID/conf/cas\_ssl\_cert.crt로 대체하면 된다.

## 지원하는 드라이버

큐브리드는 다양한 드라이버를 제공하고 있으나, 현재 패킷 암호화 연결을 지원하는 드라이버는 JDBC, CCI 이다.

### 암호화 연결 방법

클라이언트는 드라이버 연결 설정 중 db-url의 useSSL property를 사용하여 서버와 암호화 연결을 할 수 있다. 자세한 사용 방법은 JDBC 드라이버의 [연결 설정](#) 또는 CCI 드라이버의 [cci\\_connect\\_with\\_url](#)을 참고한다.

## 서버 접근제어

큐브리드는 브로커와 데이터베이스 서버의 2계층에서 인가된 사용자 또는 응용프로그램만 접근이 가능한 제어 기능을 지원한다. 브로커의 접근 제어는 웹과 웹 어플리케이션 서버(Web Application Server)등과 같은 응용프로그램들의 접근을 제어하는데 주로 사용하고, 데이터베이스 서버의 접근 제어는 브로커 및 csql 인터프리터등의 접근을 제어하는데 주로 사용한다.

자세한 내용은 [브로커 서버 접속 제한](#) 과 [데이터베이스 서버 접속 제한](#) 을 참조한다.

## 권한 관리

큐브리드는 사용자(그룹)를 생성할 수 있고, 사용자가 생성한 테이블에 대해 다른 사용자(그룹)의 접근 여부를 제어할 수 있는 기능을 제공한다.

자신이 만든 테이블에 다른 사용자(그룹)의 접근을 허용하려면 **GRANT** 구문을 사용하여 해당 사용자(그룹)에게 적절한 권한을 제공해야 한다. 접근 권한을 해지하기 위해서는 **REVOKE** 구문을 사용하여 해당 사용자(그룹)으로부터 권한을 회수할 수 있다. PUBLIC 사용자가 생성한 (가상) 테이블은 권한 제공 절차 없이 모든 사용자에게 접근이 허용된다.

자세한 내용은 **사용자 관리** 를 참조한다.

## TDE (Transparent Data Encryption)

### CUBRID TDE 개념

큐브리드는 **Transparent Data Encryption (이하 TDE)** 를 지원한다. TDE란 사용자의 관점에서 투명하게 (Transparent) 데이터를 암호화하는 것을 의미한다. 이를 통해 사용자는 애플리케이션의 변경을 거의 하지 않고 디스크에 저장되는 데이터를 암호화할 수 있다.

큐브리드 TDE는 암호화로 인한 성능 저하를 최소화하기 위하여 엔진 레벨에서 암복호화를 제공한다. 사용자가 암호화된 테이블을 생성하면 디스크에 저장되는 관련 유저 데이터 (data at rest)는 모두 자동으로 암호화된다. 큐브리드는 TDE를 제공함으로써 사용자가 다양한 현장에서 요구되는 보안 규정 및 지침을 준수할 수 있게 한다.

### 테이블 암호화

큐브리드에서는 테이블을 암호화 대상으로 설정한다. TDE 기능을 사용하기 위해서는 다음과 같이 **ENCRYPT** 옵션을 사용하여 테이블을 생성한다. 자세한 내용은 **테이블 암호화 (TDE)** 를 참고한다.

```
CREATE TABLE tde_tbl (att1 INT, att2 VARCHAR(20)) ENCRYPT=AES;
```

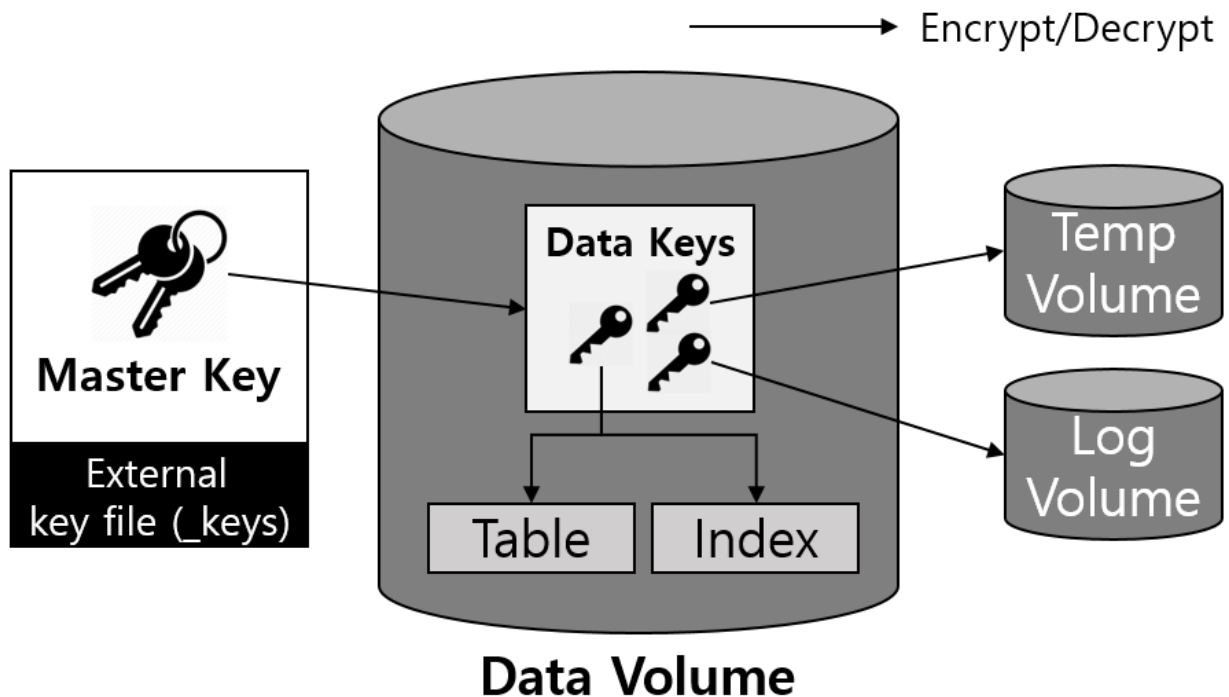
암호화된 테이블이 생성되면 테이블에 관련된 모든 데이터는 디스크에 쓰일 때 자동으로 암호화되고, 메모리로 읽어올 때 복호화된다. 관련 데이터는 단순히 테이블뿐만 아니라, 테이블에 생성되는 인덱스, 테이블과 관련된 질의 수행 중 생성되는 임시 데이터, 데이터 변경 시 생성되는 로그, DWB, 백업 등 모든 연관 데이터를 포함한다. 자세한 내용은 **암호화 대상** 과 **제약 사항** 을 참고한다.

### 키 관리

큐브리드는 데이터를 암호화하기 위하여 대칭키 알고리즘을 사용한다. 암복호화에 사용되는 키는 효율성을 위하여 마스터 키와 데이터 키로 구성된 2 계층으로 관리된다. 사용자가 관리하는 마스터 키는 별도의 파일에 저장되며, 큐브리드는 이를 관리할 수 있는 도구를 제공한다.

## 2 계층 키 관리

큐브리드 TDE는 다음과 같이 마스터 키와 데이터 키로 이루어진 2 계층으로 암호화 키를 관리한다.



- **마스터 키**: 데이터 키를 암호화할 때 사용되는 키로, 사용자에게 공개되어 관리되는 키
- **데이터 키**: 실제 테이블 및 로그 등의 유저 데이터를 암호화할 때 사용되는 키로, 큐브리드 엔진이 사용하는 키

데이터 키는 데이터 볼륨 안에서 별도로 관리되며 디스크에 저장될 때에는 마스터 키를 사용해 항상 안전하게 암호화된다. 마스터 키는 별도의 파일에 저장되며 사용자의 보안 정책에 따라 안전하게 관리되어야 한다.

2 계층으로 키를 관리하는 것은 키 변경 작업을 효율적으로 수행할 수 있게 해준다. 만약 실제 데이터를 암호화하는 키만 존재한다면, 키를 변경할 경우에 모든 데이터를 다시 읽어 들여 복호화 및 재암호화하는 과정을 거쳐야 하기 때문에 작업 시간이 오래걸리고 그 과정동안 데이터베이스의 전체적인 성능 저하를 가져올 수 있다.

### ⚠ 경고

#### 마스터 키 유실

마스터 키가 유실될 경우 TDE를 사용하여 암호화된 데이터는 다시 읽어 들일 수도, 변경할 수도 없다.

## 파일 기반의 마스터 키 관리

마스터 키는 사용자가 개별 보안 요구사항에 맞게 다양한 방법으로 키를 관리할 수 있도록 별도의 키 파일로 따로 저장되어 관리된다. 키 파일에는 마스터 키의 정보가 모두 들어가 있기 때문에 유출될 경우 보안상의 문제가 발생할 수 있고, 분실할 경우에는 암호화된 정보를 읽어 들일 수 없으므로 (TDE 기능을 사용할 수 없을 때의 동작) 관리에 주의가 필요하다.

키 파일은 기본적으로 createdb 를 사용해 데이터베이스 생성 시에 데이터 볼륨이 생성되는 위치에 **<database-name>\_keys** 이름으로 함께 생성된다. 이후 별도의 조작을 하지 않을 경우 해당 키 파일이 자동으로 사용된다. 참조할 키 파일의 위치는 시스템 파라미터를 통해서 변경 가능하다. 자세한 내용은 **디스크 관련 파라미터** 를 참고한다.

키 파일은 다수의 마스터 키를 포함할 수 있다 (최대 128개). 다수의 마스터 키 중 하나의 키를 등록하여 데이터베이스를 암호화하는 데 사용한다. 기본적으로 키 파일이 생성될 때 하나의 마스터 키를 포함하며 사용자는 TDE 유틸리티 (TDE 유틸리티)를 사용하여 키를 추가, 제거, 변경, 조회할 수 있다. 키 제거 시에는 해당 키가 키 파일에 존재해야 하며, 현재 데이터베이스에 등록된 키는 제거할 수 없다. 데이터베이스에 등록된 키를 변경 시에는 이전에 사용하던 키와 등록하려는 키가 모두 키 파일 안에 존재해야 한다. 키 조회를 통해 키의 개수, 생성 시간 등을 확인할 수 있고 현재 데이터베이스에 등록된 키와 등록 시간 등을 확인할 수 있다.

```
$ cubrid tde --show-keys testdb
Key File: /home/usr/CUBRID/databases/testdb/testdb_keys

The current key set on testdb:
Key Index: 2
Created on Fri Nov 27 11:14:54 2020
Set on Fri Nov 27 11:15:30 2020

Keys Information:
Key Index: 0 created on Fri Nov 27 11:11:27 2020
Key Index: 1 created on Fri Nov 27 11:14:47 2020
Key Index: 2 created on Fri Nov 27 11:14:54 2020
Key Index: 3 created on Fri Nov 27 11:14:55 2020

The number of keys: 4
```

### 참고

#### 기존의 키 파일을 사용하는 데이터베이스 생성

보안정책 등의 이유로 기존에 관리하던 키 파일을 사용하여 새로운 데이터베이스를 생성하고 싶다면, 데이터베이스 생성 전에 데이터베이스를 생성하려는 경로에 해당 키 파일을 먼저 복사 혹은 이동시켜두면 된다. 이때, 키 파일의 이름은 **<database\_name>\_keys** 으로 변경해야 한다. 만



약 **tde\_keys\_file\_path** 시스템 파라미터를 사용 중이라면 파라미터로 설정된 경로로 키 파일을 복사한다.

## 암호화 대상

### 영구 데이터 암호화

암호화된 테이블의 데이터와 모든 인덱스의 데이터가 암호화된다. 테이블 암호화에 대한 자세한 내용은 **테이블 암호화 (TDE)** 을 참고한다.

### 임시 데이터 암호화

테이블과 같은 영구 데이터 외에도 암호화 테이블과 관련된 질의 수행 중 생성되는 임시 데이터 또한 암호화된다. 예를 들어 `SELECT * FROM tde_tbl ORDER BY att1` 과 같은 질의를 수행하거나 `tde_tbl` 에 인덱스를 생성하는 과정 등에서 생성된 임시 데이터는 모두 암호화되어 디스크에 저장된다. 임시 데이터에 대한 자세한 내용은 **일시적 볼륨(Temporary Volume)** 을 참고한다.

### 로그 데이터 암호화

암호화된 테이블이 변경되면서 발생하는 REDO, UNDO 로그 레코드에 암호화가 필요한 데이터가 포함될 수 있으므로, 암호화 테이블과 관련된 모든 로그 정보를 암호화한다. 암호화는 활성 로그와 보관 로그에 모두 적용된다. 로그 볼륨에 대한 자세한 내용은 **데이터베이스 볼륨** 을 참고한다.

### DWB 암호화

영구 데이터는 데이터 볼륨에 쓰이기 전에 이중 쓰기 버퍼 (Double Write Buffer, DWB)에 일시적으로 쓰는데, 이때에도 암호화된 테이블에 대한 데이터를 포함될 수 있으므로 암호화된다. DWB에 대한 자세한 내용은 **데이터베이스 볼륨** 을 참고한다.

### 백업 볼륨 암호화

데이터 볼륨과 로그 볼륨에 암호화된 테이블이 있는 경우 백업 볼륨에도 동일하게 암호화된 상태로 저장된다. 백업에 대한 자세한 내용은 **backupdb** 를 참고한다.

### 백업 키 파일

백업 볼륨에는 기본적으로 키 파일이 포함된다. 키 파일을 포함한 백업 볼륨이 유출될 경우, 볼륨 내의 데이터들은 암호화되어 있지만, 마스터 키도 함께 유출되므로 보안상의 문제가 있을 수 있다. 이

를 방지하기 위해서는 **--separate-keys** 를 통해 키를 분리하여 백업할 수 있다. 하지만, 이렇게 키 파일을 분리한 경우에는 복구를 위해 키 파일을 분실하지 않게 관리해야 한다. 분리된 백업 키 파일은 백업 볼륨과 같은 경로에 생성되며 **<database\_name>\_bk<backup\_level>\_keys** 라는 이름을 가진다.

```
$ cubrid backupdb -S --separate-keys testdb
Backup Volume Label: Level: 0, Unit: 0, Database testdb, Backup
Time: Mon Nov 30 14:34:49 2020
$ ls
lob testdb testdb_bk0_keys testdb_bk0v000 testdb_bkvinf
testdb_keys
testdb_lgar_t testdb_lgat testdb_lginf testdb_vinf
```

### 백업 복구 시 키 파일 선택

백업 시에 분리된 키 파일은 백업 복구 (restoredb)를 수행할 때 **--keys-file-path** 옵션을 통해 복구에 사용할 키 파일로 지정해 줄 수 있다. 지정한 경로에 올바른 키 파일이 존재하지 않을 경우 백업 복구는 실패한다.

**--keys-file-path** 옵션이 주어지지 않을 경우에는 백업 복구 (restoredb)에서 다음의 우선 순위에 따라 사용할 키 파일을 탐색한다. 올바른 키 파일을 찾을 수 없다면 복구에 실패한다.

### 키 파일 분류

- 서버 키 파일: 일반적으로 서버 실행 시 참조하는 키 파일로 tde\_keys\_file\_path 시스템 파라미터로 설정되어 있거나 기본 경로에 있는 키 파일.
- 백업 키 파일: 백업 시에 생성된 키 파일로 백업 볼륨 내에 포함되어 있거나, **--separate-keys**로 분리된 키 파일.

### 백업 복구 시 키 파일 탐색 순서

1. 백업 볼륨이 포함하고 있는 백업 키 파일
2. 백업 시에 **--separate-keys** 옵션으로 생성된 백업 키 파일 (e.g. testdb\_bk0\_keys). 이 키 파일은 백업 볼륨과 같은 위치에 존재해야 한다.
3. 시스템 파라미터 **tde-keys-file-path** 로 지정된 경로에 있는 서버 키 파일
4. 데이터 볼륨과 같은 위치에 있는 서버 키 파일 (e.g. testdb\_keys)

#### 참고

(1)에서 백업 볼륨이 키 파일을 포함하고 있을 경우, 백업 볼륨 압축해제 과정에서 **--separate-keys** 를 통해 생성한 것과 같은 이름으로 백업 키 파일이 생성된다.

올바른 키 파일을 찾지 못하더라도 백업 볼륨에 암호화된 데이터가 전혀 없다면 복구에 성공할 수 있다. 하지만, 키 파일이 존재하지 않으므로 이후에 TDE 기능을 사용할 수 없다.

#### 참고

##### 충분 백업의 경우

충분 백업되어 여러 백업 볼륨을 사용하여 백업 복구를 수행할 경우, **--level** 옵션으로 지정하는 레벨의 백업 키 파일을 사용한다. **--level** 옵션을 지정해주지 않을 경우에는 가장 높은 레벨의 백업 키 파일을 사용한다. 사용하는 키 파일만 존재하면 복구가 가능하며, 해당 키 파일은 반드시 존재해야 한다.

#### 참고

##### 백업 키 파일을 분실한 경우

기본적으로 백업 키 파일을 분실한 경우에는 백업 복구를 수행할 수 없다. 하지만 키를 변경하지 않은 경우, 이전 볼륨의 백업 키 파일을 **--keys-file-path** 옵션으로 지정하여 복구가 가능하다. 또한, 기본 경로에 이전 볼륨의 백업 키가 존재한다면 백업 복구에 사용할 수 있다.

#### 참고

##### 백업 복구 완료 후 등록 키가 변경되는 경우

백업 복구 완료 후 데이터베이스에 등록된 키가 데이터베이스 키 파일에 존재하지 않을 경우, 백업 키 파일이 서버 키 파일로 복사되고 해당 키 파일의 첫 번째 키가 데이터베이스를 암호화하는 마스터 키로 임의로 지정된다. 이는 백업복구 완료 시의 데이터베이스에 등록되어있는 마스터 키가 어떠한 키 파일에도 존재하지 않을 수 있기 때문이다.

## 암호화 알고리즘

큐브리드에서 제공하는 대칭키 암호화 알고리즘은 다음과 같다.

### TDE 암호화 알고리즘

| 알고리즘                          | 키 크기     | 옵션 이름 |
|-------------------------------|----------|-------|
| Advanced Encryption Algorithm | 256 bits | AES   |
| ARIA                          | 256 bits | ARIA  |

Advanced Encryption Algorithm (AES)는 미국 표준 기술 연구소 (NIST)에 의해 제정된 암호화 방식으로 세계적으로 널리 사용되고 있는 알고리즘이다. 안정성이 높고 많은 플랫폼에서 하드웨어 가속 등의 최적화를 지원하여 암호화 성능 저하가 적다. ARIA는 대한민국의 국가 표준 암호화 알고리즘 중 하나로 경량 환경 및 하드웨어 구현에 최적화된 알고리즘이다.

### 참고

#### 기본 암호화 알고리즘

TDE 암호화 테이블 생성 시 알고리즘을 지정하지 않으면 AES를 기본적으로 사용한다. 만약, 기본 암호화 알고리즘을 변경하고 싶을 경우에는 시스템 파라미터 **tde\_default\_algorithm**을 통해 변경할 수 있다. 설정된 기본 암호화 알고리즘은 테이블 외에도 로그나 임시 데이터를 암호화할 때에도 사용된다. 테이블 생성 시 암호화 알고리즘 지정에 대한 자세한 설명은 [테이블 암호화 \(TDE\)](#)을 참고한다.

## 암호화 테이블 확인

테이블의 암호화 여부는 세 가지 방법으로 확인할 수 있다.

### SHOW CREATE TABLE 구문 이용

```
csql> show create table tde_tbl1;

=== <Result of SELECT Command in Line 1> ===

TABLE CREATE TABLE
=====
'tde_tbl1' 'CREATE TABLE [tde_tbl1] ([a] INTEGER)
REUSE_OID, COLLATE iso88591_bin ENCRYPT=AES'

1 row selected. (0.144627 sec) Committed.
```

```
1 command(s) successfully processed.
```

db\_class 시스템 카탈로그에 질의

시스템 카탈로그인 **db\_class** 혹은 **\_db\_class** 의 **tde\_algorithm** 속성을 통해 각 테이블의 암호화 여부 및 암호화 알고리즘을 파악할 수 있다. 시스템 카탈로그에 대한 자세한 내용은 [시스템 카탈로그](#)를 참고한다.

```
csql> select class_name, tde_algorithm from db_class where
class_name like '%tde%';
```

```
=== <Result of SELECT Command in Line 1> ===
```

| class_name    | tde_algorithm |
|---------------|---------------|
| 'tde_tbl1'    | 'AES'         |
| 'tde_tbl2'    | 'ARIA'        |
| 'not_tde_tbl' | 'NONE'        |

```
3 rows selected. (0.057243 sec) Committed.
```

```
1 command(s) successfully processed.
```

cubrid diagdb 유틸리티를 통한 확인

**cubrid diagdb** 유틸리티의 -d1(dump file tables) 옵션에서 암호화된 테이블, 인덱스 파일들의 파일 헤더정보 중 **tde\_algorithm** 을 참조하여 확인할 수 있다. 자세한 내용은 [diagdb](#) 를 참고한다.

```
$ cubrid diagdb -d1 testdb
```

```
...
```

```
Dumping file 0|3520
```

```
file header:
```

```
 vfid = 0|3520
```

```
 permanent
```

```
 regular
```

```
 tde_algorithm: AES
```

```
 page: total = 64, user = 1, table = 1, free = 62
```

```
 sector: total = 1, partial = 1, full = 0, empty = 0
```

```
...
```

## HA 환경에서의 TDE

HA 구성 시 각각의 노드마다 독립적으로 TDE가 적용된다. 이는 각 노드마다 키 파일 및 TDE 관련 시스템 파라미터들이 독립적으로 관리 될 수 있음을 의미한다.

단, 복제된 테이블의 TDE 정보는 공유되므로 마스터 노드의 TDE 테이블을 변경하려 할 때, 슬레이브 노드의 TDE 모듈이 로드되어 있지 않다면 복제가 멈추게 된다. 이때는 TDE 테이블뿐만 아니라 그 이후의 변경에 대한 복제도 수행하지 못한다. 이후 슬레이브 노드의 TDE 구성을 올바르게 한 후 재시작할 경우 중단된 부분부터 복제가 재개된다.

## TDE 기능을 사용할 수 없을 때의 동작

다음과 같은 경우에는 TDE 기능을 사용할 수 없고, TDE 모듈이 올바르게 로드되지 않았다고 오류를 발생한다.

- 올바른 키 파일을 찾을 수 없을 때
- 키 파일 내에 데이터베이스에 등록된 키를 찾을 수 없을 때

TDE 모듈이 로드되지 못한 경우에도 서버는 온전히 구동되며, 사용자는 암호화되지 않은 테이블들에 대해서는 접근할 수 있다. 이는 해당 테이블에 대한 SELECT, INSERT 등 모든 DML 및 DDL을 수행할 수 없음을 의미한다. 단, 큐브리드는 TDE 모듈이 로드되지 못하였다 하더라도 서버는 온전하게 실행되며, 암호화되지 않은 테이블들에 대해서는 접근할 수 있도록 한다.

TDE 모듈을 로드하지 못한 상태에서 로그 데이터가 암호화되어 있고 해당 로그가 리커버리, HA, VACUUM 등에 의해 접근될 경우에는 시스템이 올바르게 수행될 수 없으므로 서버 전체가 동작을 정지한다.

## 제약 사항

앞서 설명된 내용 이외에 다음과 같은 제한이 있다.

1. HA 구성 시 복제 로그는 암호화되지 않는다.
2. ALTER TABLE 구문을 통한 암호화 알고리즘 변경 및 해제를 제공하지 않는다. 기존 암호화된 테이블의 암호화 알고리즘을 변경 또는 해제하고 싶은 경우, 새로운 테이블로 데이터를 이동시키는 작업이 필요하다.
3. SQL 로그에 찍히는 데이터는 암호화되지 않는다. SQL 로그에 관한 자세한 내용은 [SQL 로그 관리](#)를 참고한다.

# CUBRID SHARD

## Database Sharding

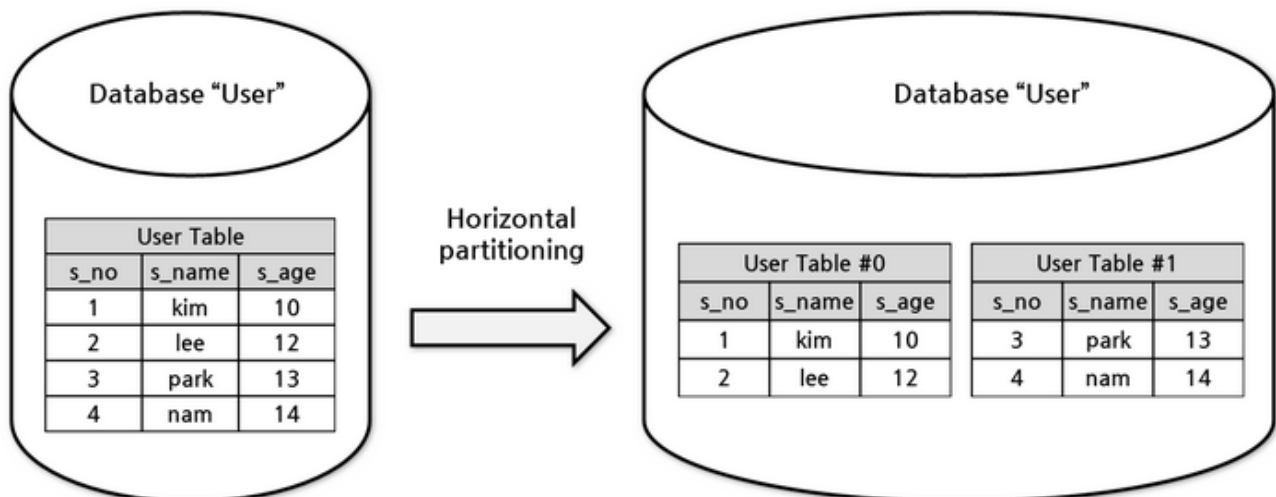
### 수평 분할

**수평 분할(Horizontal Partitioning)**이란, 동일한 스키마를 가진 데이터를 행(row) 단위로 나누어 두 개 이상의 테이블에 분산 저장하는 방식을 말한다.

예를 들어, 하나의 'User Table'을 나이 기준으로 분할하여 13세 미만의 유저는 'User Table #0', 13세 이상의 유저는 'User Table #1'에 저장하도록 설계할 수 있다.

수평 분할을 적용하면 각 테이블의 데이터 및 인덱스 크기가 작아져 쿼리 성능이 향상되며, 작업의 동시성도 높아져 전체적인 성능 개선을 기대할 수 있다.

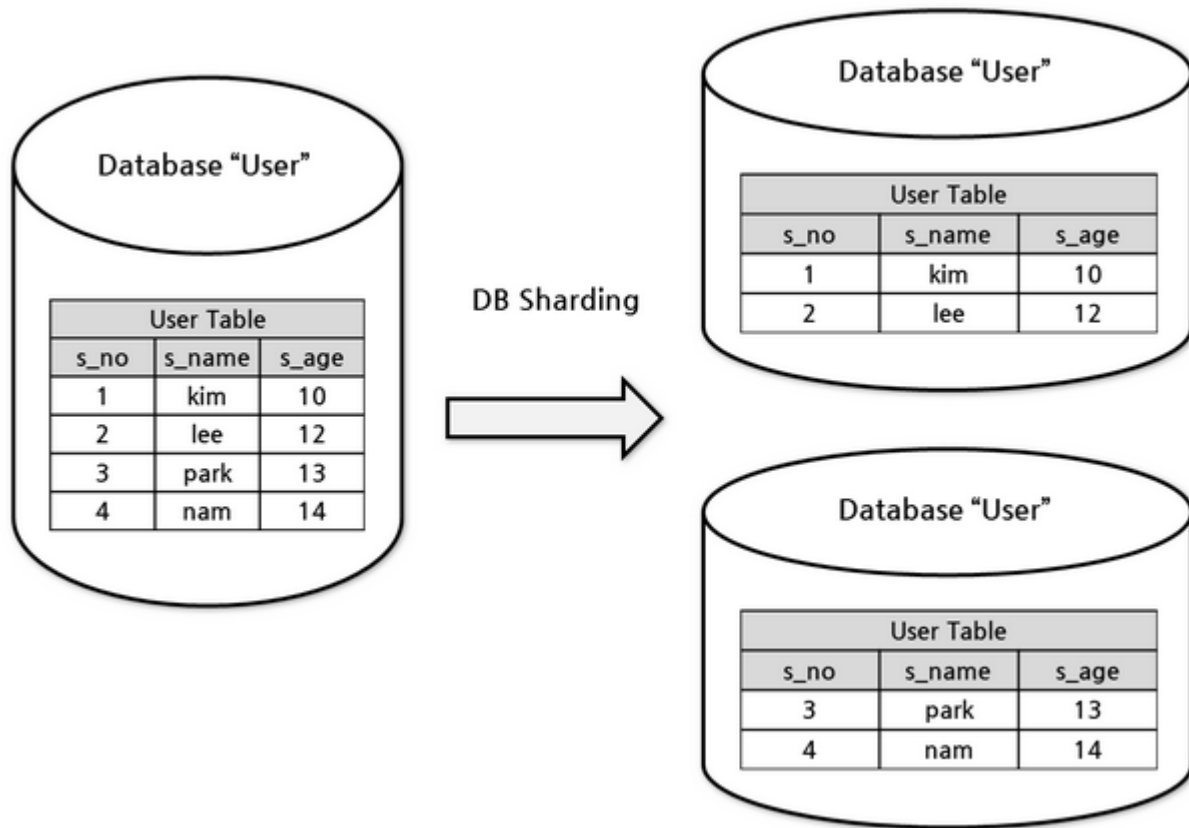
수평 분할은 일반적으로 하나의 데이터베이스 내에서 이루어지며, 대용량 데이터를 효과적으로 관리하고 시스템 부하를 분산시키는 데 유용하다.



### database sharding

database sharding은 물리적으로 다른 데이터베이스에 데이터를 수평 분할(horizontal partitioning) 방식으로 분산 저장하고 조회하는 방법을 말한다. 예를 들어 'User Table'이 여러 데이터베이스에 있을 때 13세 미만의 유저를 0번 데이터베이스에 13세 이상의 유저를 1번 데이터베이스에 저장되도록 하는 방식이다. database sharding은 성능상 이유뿐 아니라 하나의 데이터베이스 인스턴스에 넣을 수 없는 큰 데이터를 분산하여 처리하기 위해 사용된다.

분할된 각 데이터베이스를 shard 또는 database shard라고 부른다.



## CUBRID SHARD

CUBRID SHARD는 database sharding을 위한 미들웨어로, 다음과 같은 특징을 갖는다.

- 기존 응용의 변경을 최소화하기 위한 미들웨어 형태로서, 흔히 사용되는 JDBC(Java Database Connectivity)나 CUBRID C API인 CCI(CUBRID C Interface)를 이용하여 투명하게 sharding된 데이터에 접근할 수 있다.
- 힌트를 이용하여 실제 질의 수행할 shard를 선택하는 방식으로, 기존 사용하던 질의에 힌트를 추가하여 사용할 수 있다.
- 일부 트랜잭션의 특성을 보장한다.

## CUBRID SHARD 기본 용어

다음은 앞으로 CUBRID SHARD를 설명하기 위해 사용되는 용어들로, 각 의미는 다음과 같다.

- **shard DB** : 분할된 테이블과 데이터를 포함하고 있으며, 실제로 사용자 요청을 처리하는 데이터베이스
- **shard metadata** : CUBRID SHARD의 동작을 위한 설정 정보. 요청된 질의를 분석하여 실제 질의를 수행할 shard DB를 선택하기 위한 정보 및 shard DB와의 데이터베이스 세션을 생성하기 위한 정보를 포함한다.

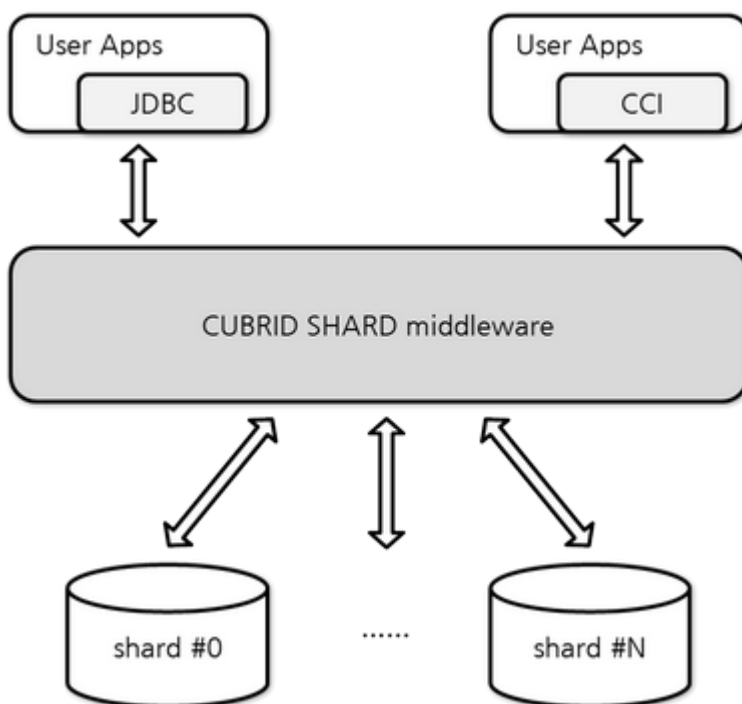


- **shard key (column)** : sharding된 테이블에서 shard를 선택하기 위한 식별자로 사용되는 칼럼
- **shard key data** : 질의 중 shard를 식별하기 위한 힌트에 해당하는 shard key의 값
- **shard ID** : shard DB를 식별하기 위한 식별자
- **proxy** : 사용자 질의에 포함된 힌트를 해석하고, 해석된 힌트와 shard metadata를 이용하여 실제 질의 처리할 shard DB로 요청을 전달하는 역할을 하는 CUBRID 미들웨어 프로세스

## CUBRID SHARD 주요 기능

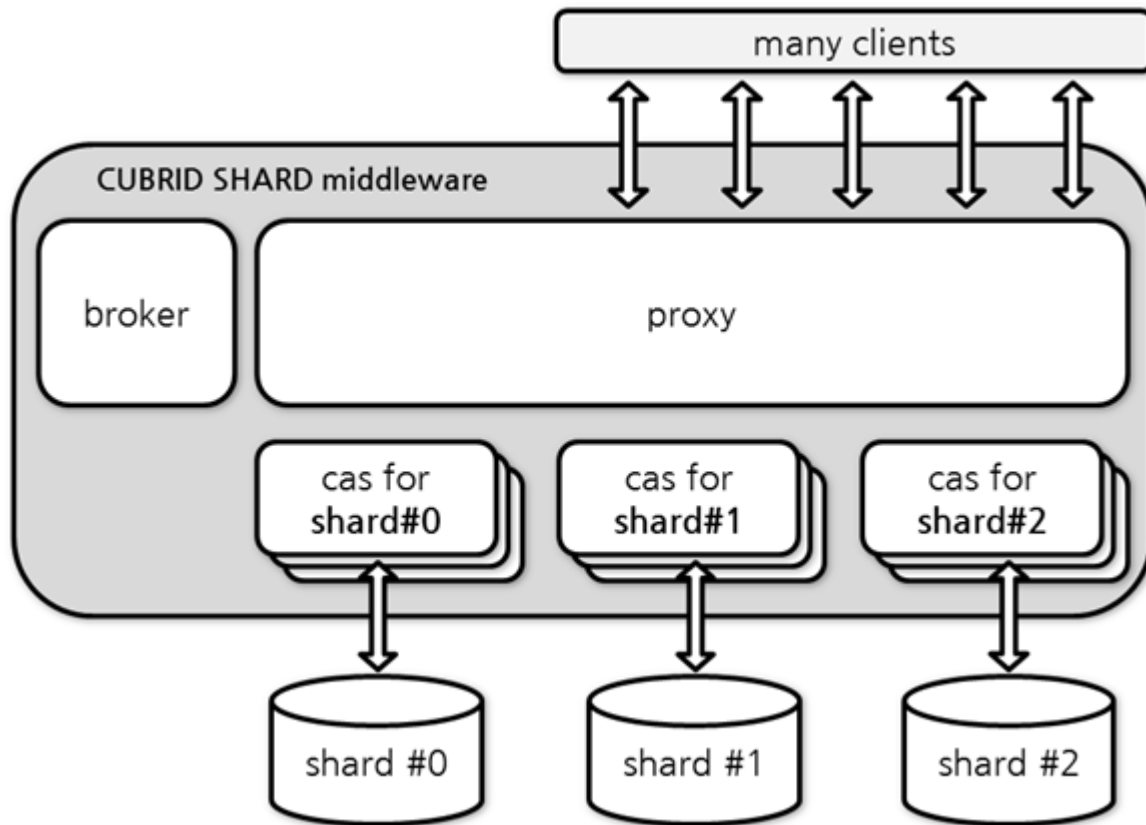
### 미들웨어 구조

CUBRID SHARD는 응용 프로그램과 물리적 또는 논리적으로 분할된 shard의 중간에 위치하는 미들웨어(middleware)로서, 동시에 다수의 응용 프로그램과의 연결을 유지하며, 응용의 요청이 있는 경우 적절한 shard로 전달하여 처리하고 결과를 응용에 반환하는 기능을 수행한다.



일반적으로 사용되는 JDBC(Java Database Connectivity)나 CUBRID C 인터페이스인 CCI(CUBRID C Interface)를 이용하여 CUBRID SHARD로 연결하여 응용의 요청을 처리할 수 있으며, 별도의 드라이버나 프레임워크가 필요 없기 때문에 기존 응용의 변경을 최소화할 수 있다.

CUBRID SHARD middleware는 broker/proxy/CAS 세 개의 프로세스로 구성되며, 각 프로세스의 간략한 기능은 다음과 같다.



#### · broker

- JDBC/CCI 등 드라이버로부터의 최초 연결 요청을 수신하고, 수신된 연결 요청을 부하 분산 정책에 따라 proxy로 전달
- proxy 프로세스와 CAS 프로세스의 상태 감시 및 복구

#### · proxy

- 드라이버로부터의 사용자 요청을 CAS로 전달하고, 처리한 결과를 응용에 반환
- 드라이버 및 CAS와의 연결 상태 관리 및 트랜잭션 처리

#### · CAS

- 분할된 shard DB와 연결을 생성하고, 그 연결을 이용하여 proxy로부터 수신한 사용자 요청(질의)를 처리
- 트랜잭션 처리

## shard SQL 힌트를 이용한 shard DB 선택

### shard SQL 힌트

CUBRID SHARD는 SQL 힌트 구문에 포함된 힌트와 설정 정보를 이용하여, 응용으로부터 요청된 질의를 실제로 처리할 shard DB를 선택한다. 사용 가능한 SQL 힌트의 종류는 다음과 같다.

| SQL 힌트                                  | 설명                                                                     |
|-----------------------------------------|------------------------------------------------------------------------|
| <code>/*+ shard_key */</code>           | shard key 칼럼에 해당하는 바인드 변수 또는 리터럴 값의 위치를 지정하기 위한 힌트                     |
| <code>/*+ shard_val( value )*/</code>   | 질의 내에 shard key에 해당하는 칼럼이 존재하지 않는 경우 힌트 내에 shard key를 명시적으로 지정하기 위한 힌트 |
| <code>/*+ shard_id( shard_id )*/</code> | 사용자가 특정 shard DB를 지정하여 질의를 처리하고자 할 때 사용하는 힌트                           |

설명을 위해 간략하게 용어를 정리하면 다음과 같다. 용어에 대한 보다 자세한 설명은 [CUBRID SHARD 기본 용어](#)를 참고한다.

- **shard key** : shard DB를 식별할 수 있는 칼럼. 일반적으로 shard DB 내의 모든 혹은 대부분의 테이블에 존재하는 칼럼으로서, DB 내에서 유일한 값을 갖는다.
- **shard id** : shard를 논리적으로 구분할 수 있는 식별자. 예를 들어, 하나의 DB가 4개의 shard DB로 분할되면 4개의 shard id가 존재한다.

힌트와 설정 정보를 이용한 자세한 질의 처리 절차는 [shard SQL 힌트를 이용하여 질의가 수행되는 일반적인 절차](#)를 참고한다.

#### 참고

- 하나의 질의 안에 두 개 이상의 shard 힌트가 존재할 경우 서로 같은 shard를 가리키면 정상 처리하고, 다른 shard를 가리키면 오류 처리한다.

```
SELECT * FROM student WHERE shard_key = /*+ shard_key */ 250 OR
shard_key = /*+ shard_key */ 22;
```

위와 같은 경우 250과 22가 같은 shard를 가리키면 정상 처리, 다른 shard를 가리키면 오류 처리한다.

- 여러 개의 값을 바인딩하는 배열로 질의를 일괄 처리하는 드라이버 함수(예: JDBC의 `PreparedStatement.executeBatch`, CCI의 `cci_execute_array`)에서 여러 개의 질의 중 하나라도 다른 shard에 접근하는 질의가 있으면 모두 오류 처리한다.
- shard 환경에서 한번에 여러 문장을 실행하는 함수(예: JDBC의 `Statement.executeBatch`, CCI의 `cci_execute_batch`)는 추후 지원할 예정이다.

### shard\_key 힌트

**shard\_key** 힌트는 바인드 변수나 리터럴 값의 위치를 지정하기 위한 힌트로서, 반드시 바인드 변수나 리터럴 값의 앞에 위치해야 한다.

예) 바인드 변수 위치 지정. 실행 시 바인딩되는 `student_no` 값에 해당하는 shard DB에서 질의를 수행.

```
SELECT name FROM student WHERE student_no = /** shard_key */ ?;
```

예) 리터럴 값 위치 지정. 실행 시 리터럴 값인 `student_no`가 123에 해당하는 shard DB에서 질의를 수행

```
SELECT name FROM student WHERE student_no = /** shard_key */ 123;
```

### shard\_val 힌트

**shard\_val** 힌트는 질의 내에 shard DB를 식별할 수 있는 shard key 칼럼이 존재하지 않는 경우 사용하며, 실제 질의 처리 시 무시되는 shard key 칼럼을 **shard\_val** 힌트의 값으로 설정한다. **shard\_val** 힌트는 SQL 구문의 어느 곳에도 위치할 수 있다.

예) shard key가 `student_no`이나 질의 내에 포함되지 않은 경우. shard key인 `student_no`가 123에 해당하는 shard DB에서 질의를 수행

```
SELECT age FROM student WHERE name =? /** shard_val(123) */;
```

### shard\_id 힌트

**shard\_id** 힌트는 shard key 칼럼의 값과 무관하게 사용자가 특정 shard를 지정하여 질의를 수행하고자 할 때 사용한다. **shard\_id** 힌트는 SQL 구문의 어느 곳에도 위치할 수 있다.

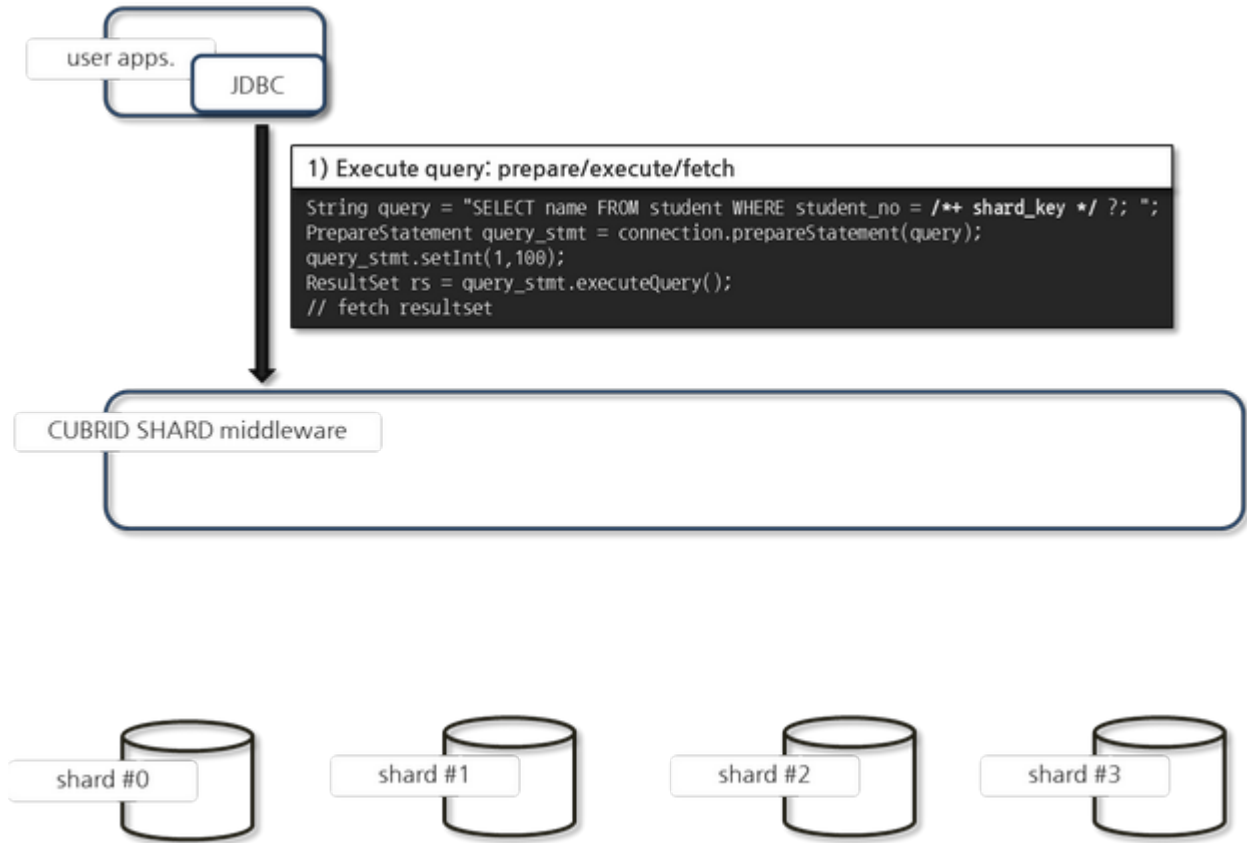
예) shard DB #3 에서 질의를 수행해야 하는 경우. shard DB #3에서 age가 17보다 큰 학생을 조회

```
SELECT * FROM student WHERE age > 17 /** shard_id(3) */;
```

## shard SQL 힌트를 이용하여 질의가 수행되는 일반적인 절차

### 1. 질의 수행

다음은 사용자 질의 요청이 수행되는 과정이다.



- 응용 프로그램은 JDBC 인터페이스를 통해 CUBRID SHARD로 질의 처리를 요청하며, 실제로 질의가 수행될 shard DB를 지정하기 위해 SQL 구문 내에 **shard\_key** 힌트를 추가한다.
- SQL 힌트는 SQL 구문 내에서 위 예에서와 마찬가지로 shard key로 설정된 칼럼의 바인드 또는 리터럴 값 바로 앞에 위치해야 한다.

바인드 변수에 설정된 shard SQL 힌트는 다음과 같다.

```

SELECT name FROM student WHERE student_no = /**+ shard_key */ ?
 ↓ ↓ ↓
shard_key 컬럼 Shard SQL Hint 바인드 변수

```

리터럴 값에 지정된 shard SQL 힌트는 다음과 같다.

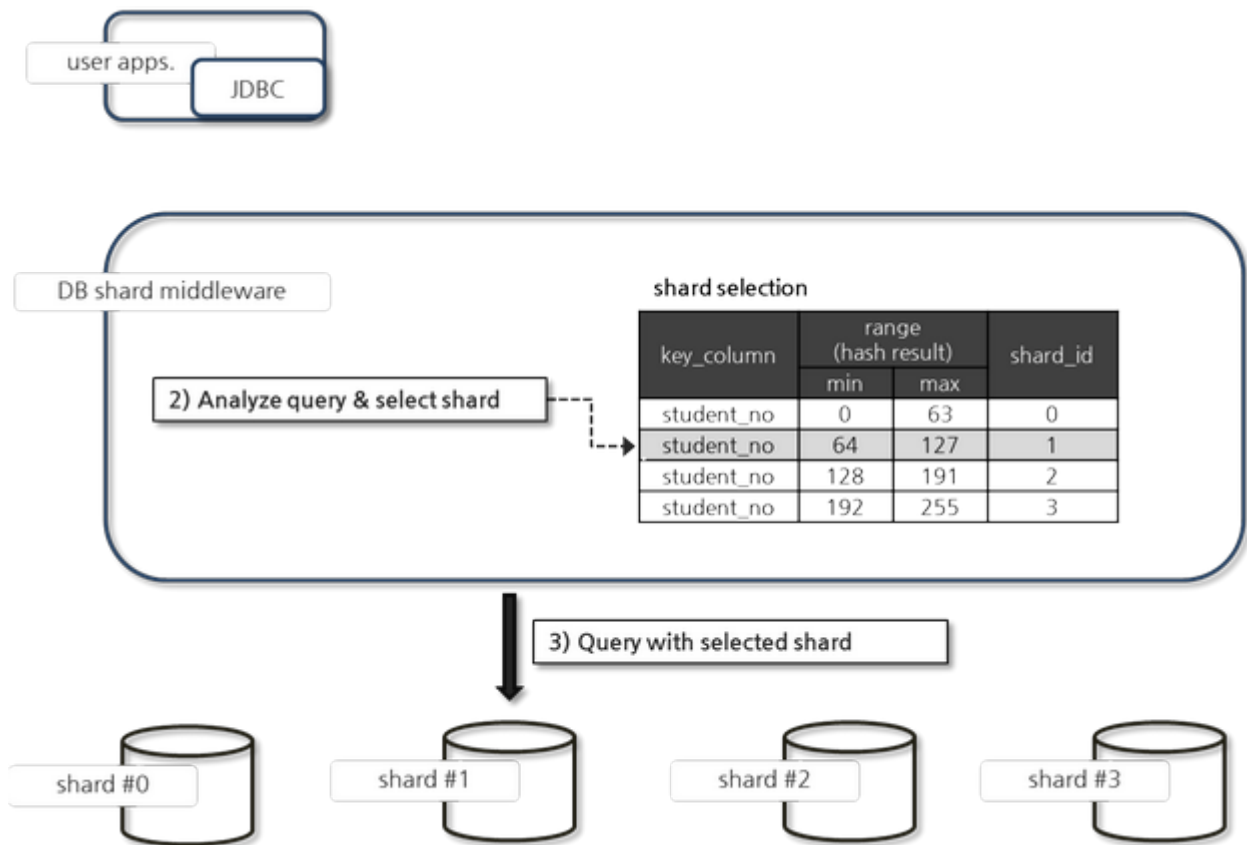
```

SELECT name FROM student WHERE student_no = /**+ shard_key */ 100
 ↓ ↓ ↓
shard_key 컬럼 Shard SQL Hint 리터럴 값

```

### 2. 질의 분석 및 실제 요청을 처리할 shard DB 선택

질의를 분석하고 실제로 요청을 처리할 shard DB를 선택하는 과정은 다음과 같다.



- 사용자로부터 수신한 SQL 질의를 내부에서 처리하기 위한 형태로 다시 작성된다(query rewrite).
- 사용자가 요청한 SQL 구문과 힌트를 이용하여 실제 질의를 수행한 shard DB를 선택한다.
  - 바인드 변수에 SQL 힌트가 설정된 경우, execute 시 shard\_key 바인드 변수에 대입된 값을 해시한 결과와 설정 정보를 이용하여 실제 질의가 수행될 shard DB를 선택한다.
  - 해시 함수는 사용자가 별도로 지정할 수 있으며, 지정하지 않은 경우 기본 내장된 해시 함수를 이용하여 shard\_key 값을 해싱한다. 기본 내장된 해시 함수는 다음과 같다.
  - shard\_key가 정수인 경우

기본 해시 함수(shard\_key) = shard\_key mod SHARD\_KEY\_MODULAR 파라미터(기본값 256)

- shard\_key가 문자열인 경우

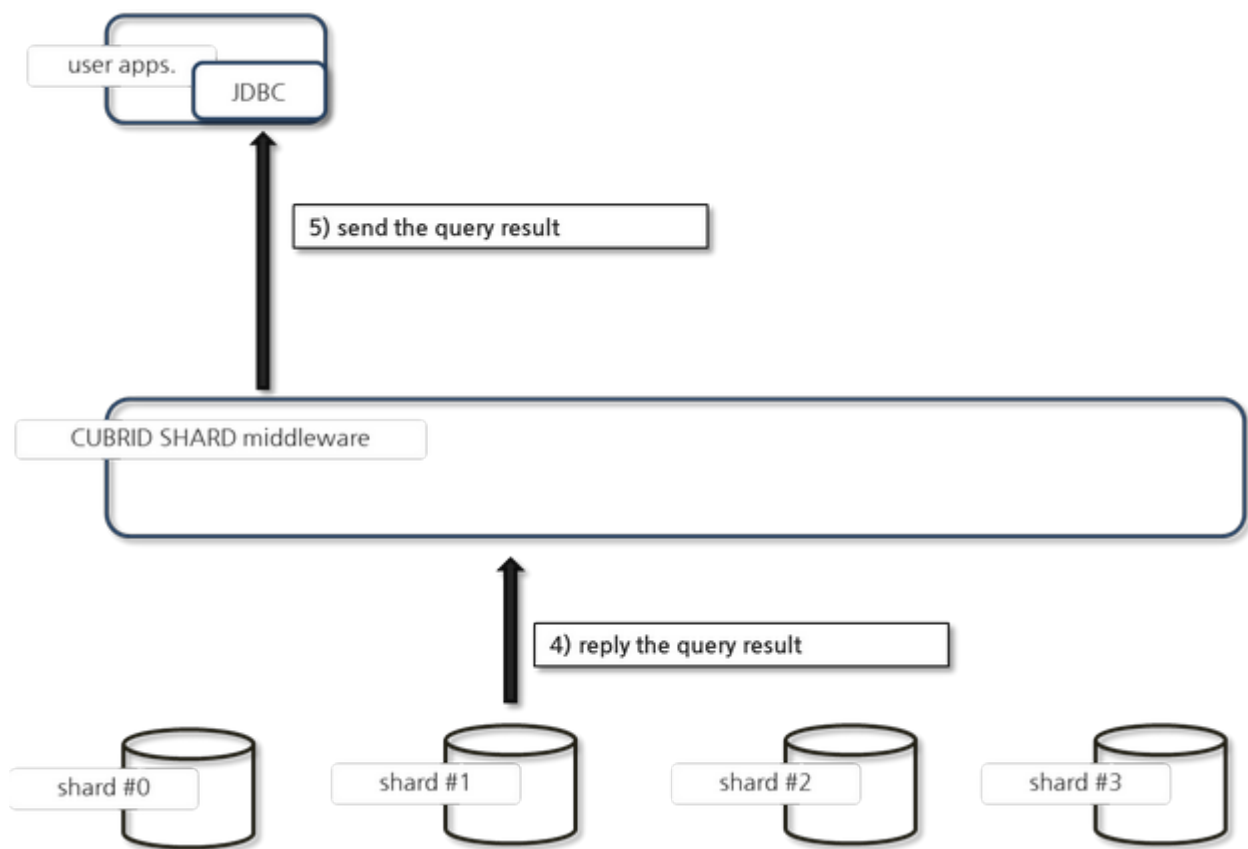
기본 해시 함수(shard\_key) = shard\_key[0] mod SHARD\_KEY\_MODULAR 파라미터(기본값 256)

### 참고

shard\_key 바인드 변수의 값이 100인 경우, “기본 hash 함수( $\text{shard\_key} = 100 \% 256 = 100$ )”이므로, 설정에 의해 해시 결과 100에 해당하는 shard DB #1이 선택되며, 선택된 shard DB #1으로 사용자 요청을 전달하게 된다.

### 3. 질의 수행 결과 반환

질의 수행 결과를 반환하는 과정은 다음과 같다.



- shard DB #1 에서 수행한 처리 결과를 수신하여, 요청한 응용으로 결과를 반환한다.

### 참고

여러 개의 값을 바인딩하는 배열로 질의를 일괄 처리하는 드라이버 함수(예: JDBC의 `executeBatch`, CCI의 `cci_execute_array`, `cci_execute_batch`)에서 다른 shard에 접근하는 값이 존재하면 오류 처리한다.

## 트랜잭션 지원

### 트랜잭션 처리

CUBRID SHARD는 ACID 중 Atomicity(원자성)을 보장하기 위한 내부적인 처리 절차를 수행한다. 예를 들어, 트랜잭션 중 응용이 비정상 종료하는 등의 예외가 발생하면 해당 응용의 질의를 처리하던 shard DB로 롤백 요청을 전달하여 해당 트랜잭션 중 변경된 내용을 모두 무효화한다.

그 외 일반적인 트랜잭션의 특성인 ACID는 backend DBMS의 특성과 설정에 따라 보장된다.

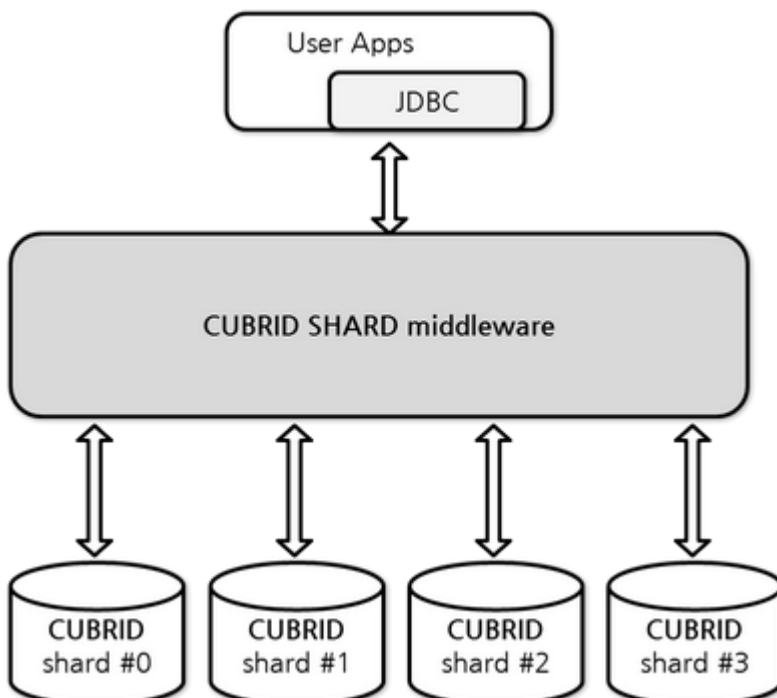
### 제약 사항

2PC(2 Phase commit)는 불가능하며, 이 때문에 하나의 트랜잭션 중 여러 개의 shard DB로 질의를 수행하는 경우 에러 처리된다.

## 빠른 시작

### 구성 예

예로 설명될 CUBRID SHARD는 아래 그림과 같이 4개의 CUBRID SHARD DB로 구성되었으며, 응용은 JDBC 인터페이스를 사용하여 사용자 요청을 처리한다.



shard DB 및 사용자 계정 생성 후 시작



위 구성의 예와 같이 각 shard DB 노드에서 shard DB 및 사용자 계정을 생성한 후 데이터베이스를 인스턴스를 시작한다.

- shard DB 이름 : *shard1*
- shard DB 사용자 계정 : *shard*
- shard DB 사용자 비밀번호 : *shard123*

```
sh> # CUBRID SHARD DB 생성
sh> cubrid createdb shard1 en_US

sh> # CUBRID SHARD 사용자 계정 생성
sh> csql -S -u dba shard1 -c "create user shard password 'shard123'"

sh> # CUBRID SHARD DB 시작
sh> cubrid server start shard1
```

## shard 설정 변경

### cubrid\_broker.conf

**cubrid\_broker.conf.shard**를 참조하여 **cubrid\_broker.conf**를 아래와 같이 변경한다.

#### ⚠ 경고

포트 번호 및 공유 메모리 식별자는 현재 시스템에서 사용하지 않는 값으로 적절히 변경해야 한다.

```
[broker]
MASTER_SHM_ID =30001
ADMIN_LOG_FILE =log/broker/cubrid-broker.log

[%shard1]
SERVICE =ON
BROKER_PORT =36000
MIN_NUM_APPL_SERVER =20
MAX_NUM_APPL_SERVER =40
APPL_SERVER_SHM_ID =36000
LOG_DIR =log/broker/sql_log
ERROR_LOG_DIR =log/broker/error_log
SQL_LOG =ON
TIME_TO_KILL =120
SESSION_TIMEOUT =300
KEEP_CONNECTION =ON
```

```

MAX_PREPARED_STMT_COUNT =1024
SHARD =ON
SHARD_DB_NAME =shard1
SHARD_DB_USER =shard
SHARD_DB_PASSWORD =shard123
SHARD_NUM_PROXY =1
SHARD_PROXY_LOG_DIR =log/broker/proxy_log
SHARD_PROXY_LOG =ERROR
SHARD_MAX_CLIENTS =256
SHARD_PROXY_SHM_ID =36090
SHARD_CONNECTION_FILE =shard_connection.txt
SHARD_KEY_FILE =shard_key.txt

```

CUBRID의 경우 **shard\_connection.txt**에 서버의 포트 번호를 별도로 설정하지 않고 **cubrid.conf** 설정 파일의 **cubrid\_port\_id** 파라미터를 사용하므로, **cubrid.conf** 의 **cubrid\_port\_id** 파라미터를 서버와 동일하게 설정한다.

```

TCP port id for the CUBRID programs (used by all clients).
cubrid_port_id=41523

```

### shard\_key.txt

shard key 해시 값에 대한 shard DB 매핑 설정 파일인 **shard\_key.txt** 파일을 아래와 같이 설정한다.

- [%shard\_key] : shard key 섹션 설정
- 기본 해시 함수에 의한 shard key 해시 결과가 0~63인 경우 shard #0 에서 질의 수행
- 기본 해시 함수에 의한 shard key 해시 결과가 64~127인 경우 shard #1 에서 질의 수행
- 기본 해시 함수에 의한 shard key 해시 결과가 128~191인 경우 shard #2 에서 질의 수행
- 기본 해시 함수에 의한 shard key 해시 결과가 192~255인 경우 shard #3 에서 질의 수행

```

[%shard_key]
#min max shard_id
0 63 0
64 127 1
128 191 2
192 255 3

```

### shard\_connection.txt

shard 구성 데이터베이스 설정 파일인 **shard\_connection.txt** 파일을 아래와 같이 설정한다.

- shard #0의 실제 데이터베이스 이름과 connection 정보
- shard #1의 실제 데이터베이스 이름과 connection 정보

- shard #2의 실제 데이터베이스 이름과 connection 정보
- shard #3의 실제 데이터베이스 이름과 connection 정보

```
shard-id real-db-name connection-info
#
0 shard1 HostA
1 shard1 HostB
2 shard1 HostC
3 shard1 HostD
```

## 서비스 시작 및 모니터링

### CUBRID SHARD 시작

CUBRID SHARD 기능을 사용하려면 아래와 같이 브로커를 구동한다.

```
sh> cubrid broker start
@cubrid broker start
++ cubrid broker start: success
```

### CUBRID SHARD 상태 조회

아래와 같이 CUBRID SHARD의 상태를 조회하여, 설정된 파라미터와 프로세스의 상태를 확인한다.

```
sh> cubrid broker status
@cubrid broker status
% shard1

ID PID QPS LQS PSIZE STATUS

1-0-1 21272 0 0 53292 IDLE
1-1-1 21273 0 0 53292 IDLE
1-2-1 21274 0 0 53292 IDLE
1-3-1 21275 0 0 53292 IDLE

sh> cubrid broker status -f
@cubrid broker status
% shard1

ID PID QPS LQS PSIZE STATUS LAST ACCESS
TIME DB HOST LAST CONNECT TIME
SQL_LOG_MODE


```

```

1-0-1 21272 0 0 53292 IDLE 2013/01/31 15:00:24
shard1@HostA HostA 2013/01/31 15:00:25 -
1-1-1 21273 0 0 53292 IDLE 2013/01/31 15:00:24
shard1@HostB HostB 2013/01/31 15:00:25 -
1-2-1 21274 0 0 53292 IDLE 2013/01/31 15:00:24
shard1@HostC HostC 2013/01/31 15:00:25 -
1-3-1 21275 0 0 53292 IDLE 2013/01/31 15:00:24
shard1@HostD HostD 2013/01/31 15:00:25 -

```

## 응용 예제 프로그램 작성

간단한 Java 프로그램을 이용하여 CUBRID SHARD 기능이 정상 동작함을 확인한다.

### 예제 테이블 생성

모든 shard DB에서 예제 프로그램을 위한 임시 테이블을 아래와 같이 생성한다.

```

sh> csql -C -u shard -p 'shard123' shard1@localhost -c "create table
student (s_no int, s_name varchar, s_age int, primary key(s_no))"

```

### 예제 프로그램 작성

다음은 0~1023번의 학생 정보를 shard DB로 입력하는 예제 프로그램이다. 앞선 절차에서 수정한 **cubrid\_broker.conf** 를 확인하여 주소/포트 및 사용자 정보를 연결 URL에 설정한다.

```

import java.sql.DriverManager;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.PreparedStatement;
import java.sql.Date;
import java.sql.*;
import cubrid.jdbc.driver.*;

public class TestInsert {

 static {
 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 } catch (ClassNotFoundException e) {
 throw new RuntimeException(e);
 }
 }
}

```

```

 public static void DoTest(int thread_id) throws SQLException
 {
 Connection connection = null;

 try {
 connection =
DriverManager.getConnection("jdbc:cubrid:localhost:36000:shard1:::?
charSet=utf8", "shard", "shard123");
 connection.setAutoCommit(false);

 for (int i=0; i < 1024; i++) {
 String query = "INSERT INTO student
VALUES (/*+ shard_key */ ?, ?, ?)";
 PreparedStatement query_stmt =
connection.prepareStatement(query);

 String name="name_" + i;
 query_stmt.setInt(1, i);
 query_stmt.setString(2, name);
 query_stmt.setInt(3, (i%64)+10);

 query_stmt.executeUpdate();
 System.out.print(".");

 query_stmt.close();
 connection.commit();
 }

 connection.close();
 } catch(SQLException e) {
 System.out.print("exception occurs : " +
e.getErrorCode() + " - " + e.getMessage());
 System.out.println();
 connection.close();
 }
 }

 /**
 * @param args
 */
 public static void main(String[] args) {
 // TODO Auto-generated method stub

 try {
 DoTest(1);
 } catch(Exception e){
 e.printStackTrace();
 }
 }
}

```

## 예제 프로그램 수행

위에서 작성한 예제 프로그램을 다음과 같이 수행한다.

```
sh> javac -cp ".:$CUBRID/jdbc/cubrid_jdbc.jar" *.java
sh> java -cp ".:$CUBRID/jdbc/cubrid_jdbc.jar" TestInsert
```

## 결과 확인

각 shard DB에서 질의를 수행하여 의도한 대로 분할된 정보가 정확하게 입력되었는지 확인한다.

- shard #0

```
sh> csql -C -u shard -p 'shard123' shard1@localhost -c "select *
from student order by s_no"
```

| s_no | s_name   | s_age |
|------|----------|-------|
| 0    | 'name_0' | 10    |
| 1    | 'name_1' | 11    |
| 2    | 'name_2' | 12    |
| 3    | 'name_3' | 13    |
| ...  |          |       |

- shard #1

```
sh> $ csql -C -u shard -p 'shard123' shard1@localhost -c "select *
from student order by s_no"
```

| s_no | s_name    | s_age |
|------|-----------|-------|
| 64   | 'name_64' | 10    |
| 65   | 'name_65' | 11    |
| 66   | 'name_66' | 12    |
| 67   | 'name_67' | 13    |
| ...  |           |       |

- shard #2

```
sh> $ csql -C -u shard -p 'shard123' shard1@localhost -c "select *
from student order by s_no"
```

| s_no | s_name     | s_age |
|------|------------|-------|
| 128  | 'name_128' | 10    |
| 129  | 'name_129' | 11    |
| 130  | 'name_130' | 12    |

```
131 'name_131' 13
...
```

• shard #3

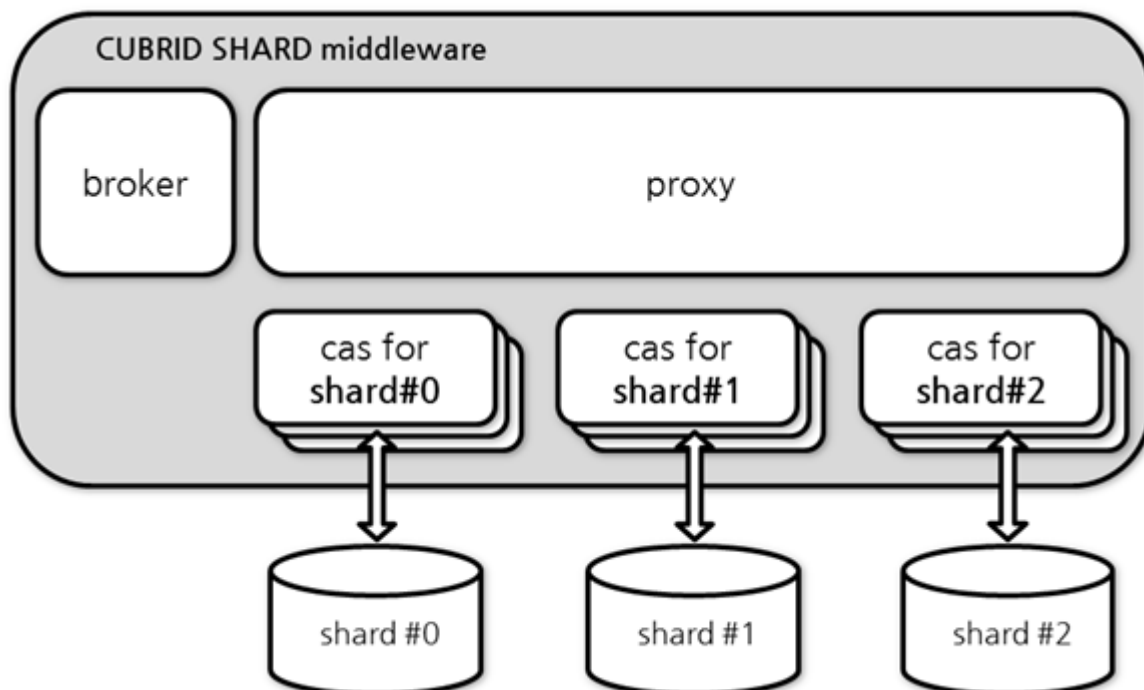
```
sh> $ csql -C -u shard -p 'shard123' shard1@localhost -c "select *
from student order by s_no"
```

| s_no | s_name     | s_age |
|------|------------|-------|
| 192  | 'name_192' | 10    |
| 193  | 'name_193' | 11    |
| 194  | 'name_194' | 12    |
| 195  | 'name_195' | 13    |
| ...  |            |       |

## 구성 및 설정

### 구성

CUBRID SHARD는 미들웨어로서 아래의 그림과 같이 broker, proxy, CAS 프로세스로 구성된다.



## 설정

CUBRID SHARD 기능을 사용하려면 **cubrid\_broker.conf** 파일에서 SHARD 관련 프로세스들의 실행에 필요한 파라미터를 설정하고, shard 연결 파일(SHARD\_CONNECTION\_FILE)과 shard key 파일(SHARD\_KEY\_FILE)을 설정해야 한다.

cubrid\_broker.conf

**cubrid\_broker.conf** 는 CUBRID SHARD 기능을 설정할 때 사용한다. 설정 시 **cubrid\_broker.conf.shard**를 참고하며, **cubrid\_broker.conf**에 대한 자세한 내용은 [브로커 설정](#)을 참고한다.

shard 연결 파일(SHARD\_CONNECTION\_FILE)

CUBRID SHARD는 브로커 구동 시 **cubrid\_broker.conf**의 **SHARD\_CONNECTION\_FILE** 파라미터에 지정된 shard 연결 설정 파일을 로딩하여 backend shard DB와의 연결을 수행한다.

설정 할 수 있는 shard DB 의 최대 개수는 256개 이다.

**cubrid\_broker.conf**에 **SHARD\_CONNECTION\_FILE**을 별도로 지정하지 않은 경우에는 기본값인 **shard\_connection.txt** 파일을 로딩한다.

## 기본 형식

shard 연결 설정 파일의 기본적인 예와 형식은 아래와 같다.

```
#
shard-id real-db-name connection-info
#
* cubrid : hostname, hostname, ...

CUBRID
0 shard1 HostA
1 shard1 HostB
2 shard1 HostC
3 shard1 HostD
```

### 참고

일반적인 CUBRID 설정과 마찬가지로 # 이후 내용은 주석으로 처리된다.

## CUBRID

backend shard DB가 CUBRID인 경우 연결 설정 파일의 형식은 다음과 같다.



```
CUBRID
shard-id real-db-name connection-info
shard 식별자(>0) 각 backend shard DB 의 실제 이름 호스트 이
름

0 shard_db_1 host1
1 shard_db_2 host2
2 shard_db_3 host3
3 shard_db_4 host4
```

CUBRID의 경우 별도의 backend shard DB의 포트 번호를 위 설정 파일에 지정하지 않고, **cubrid.conf**의 **CUBRID\_PORT\_ID** 파라미터를 사용한다. **cubrid.conf** 파일은 기본적으로 **\$CUBRID/conf** 디렉터리에 위치한다.

```
$ vi cubrid.conf

...

TCP port id for the CUBRID programs (used by all clients).
cubrid_port_id=41523
```

### shard key 파일(SHARD\_KEY\_FILE)

CUBRID SHARD는 시작 시 기본 설정 파일인 **cubrid\_broker.conf**의 **SHARD\_KEY\_FILE** 파라미터에 지정된 shard key 설정 파일을 로딩하여 사용자 요청을 어떤 backend shard DB에서 처리해야 할지 결정하는 데 사용한다.

**cubrid\_broker.conf**에 **SHARD\_KEY\_FILE**을 별도로 지정하지 않은 경우에는 기본값인 **shard\_key.txt** 파일을 로딩한다.

### 형식

shard key 설정 파일의 예와 형식은 다음과 같다.

```
[%student_no]
#min max shard_id
0 31 0
32 63 1
64 95 2
96 127 3
128 159 0
160 191 1
```

|     |     |   |
|-----|-----|---|
| 192 | 223 | 2 |
| 224 | 255 | 3 |

- [%shard\_key\_name] : shard key의 이름을 지정
- min : shard key 해시 결과의 최소값 범위
- max : shard key 해시 결과의 최대 범위
- shard\_id : shard 식별자

### 참고

일반적인 CUBRID 설정과 마찬가지로 # 이후 내용은 주석으로 처리된다.

### 경고

- shard key의 min은 항상 0부터 시작해야 한다.
- max는 최대 255까지 설정해야 한다.
- min~max 사이에는 빈 값이 존재하면 안 된다.
- 내장 해시 함수를 사용하는 경우 **SHARD\_KEY\_MODULAR** 파라미터 값(최소 1, 최대 256)을 초과할 수 없다.
- shard key 해시 결과는 0 ~ (**SHARD\_KEY\_MODULAR** - 1)의 범위에 반드시 포함되어야 한다.

## 사용자 정의 해시 함수

CUBRID SHARD는 질의를 수행할 shard를 선택하기 위해 shard key를 해싱한 결과와 메타데이터 설정 정보를 이용한다. 이를 위해 기본 내장된 해시 함수를 사용하거나, 또는 사용자가 별도로 해시 함수를 정의할 수 있다.

### 내장된 기본 해시 함수

**cubrid\_broker.conf** 의 **SHARD\_KEY\_LIBRARY\_NAME**, **SHARD\_KEY\_FUNCTION\_NAME** 파라미터를 설정하지 않는 경우 기본 내장된 해시 함수를 이용하여 shard key를 해시하며, 기본 해시 함수의 내용은 아래와 같다.

- shard\_key가 정수인 경우

기본 해시 함수(shard\_key) = shard\_key mod SHARD\_KEY\_MODULAR 파라미터(기본값 256)

- shard\_key가 문자열인 경우

기본 해시 함수(shard\_key) = shard\_key[0] mod SHARD\_KEY\_MODULAR 파라미터(기본값 256)

## 사용자 해시 함수 설정

CUBRID SHARD는 기본 내장된 해시 함수 외에 사용자 정의 해시 함수를 이용하여 질의에 포함된 shard key를 해싱할 수 있다.

### 라이브러리 구현 및 생성

사용자 정의 해시 함수는 실행 시간에 로딩 가능한 **.so** 형태의 라이브러리로 구현되어야 하며 프로토타입은 아래와 같다.

```
/*
 return value :
 success - shard key id(>0)
 fail - invalid argument(ERROR_ON_ARGUMENT),
shard key id make fail(ERROR_ON_MAKE_SHARD_KEY)
 type : shard key value type
 val : shard key value
*/
typedef int (*FN_GET_SHARD_KEY) (const char *shard_key,
 T_SHARD_U_TYPE type,
 const void *val, int
 val_size);
```

- 해시 함수의 반환 값은 **shard\_key.txt** 설정 파일의 해시 결과 범위에 반드시 포함되어야 한다.
- 라이브러리를 빌드하기 위해서는 반드시 **\$CUBRID/include/shard\_key.h** 파일을 include해야 한다. 이 파일에서 반환 가능한 예러 코드 등 자세한 내용도 확인할 수 있다.

### cubrid\_broker.conf 설정 파일 변경

생성한 사용자 정의 해시 함수를 반영하기 위해서는 **cubrid\_broker.conf**의 **SHARD\_KEY\_LIBRARY\_NAME**, **SHARD\_KEY\_FUNCTION\_NAME** 파라미터를 구현 내용에 맞도록 설정해야 한다.

- **SHARD\_KEY\_LIBRARY\_NAME** : 사용자 정의 해시 라이브러리의 (절대) 경로

- **SHARD\_KEY\_FUNCTION\_NAME** : 사용자 정의 해시 함수의 이름

## 예제

다음은 사용자 정의 해시 함수를 사용한 예이다. 먼저 **shard\_key.txt** 설정 파일을 확인한다.

```
[%student_no]
#min max shard_id
0 31 0
32 63 1
64 95 2
96 127 3
128 159 0
160 191 1
192 223 2
224 255 3
```

사용자 지정 해시 함수를 설정하기 위해서는 실행 시간에 로딩 가능한 **.so** 형태의 공유 라이브러리를 먼저 구현해야 한다. 해시 함수의 결과는 앞선 과정에서 확인한 **shard\_key.txt** 설정 파일에 정의된 해시 결과의 범위 안에 포함되는 값이어야 한다. 다음은 간단한 구현 예이다.

- shard\_key가 정수인 경우
  - shard\_key가 홀수인 경우 shard #0을 선택
  - shard\_key가 짝수인 경우 shard #1을 선택
- shard\_key가 문자열인 경우
  - shard\_key 문자열이 'a', 'A'로 시작되는 경우 shard #0을 선택
  - shard\_key 문자열이 'b', 'B'로 시작되는 경우 shard #1을 선택
  - shard\_key 문자열이 'c', 'C'로 시작되는 경우 shard #2를 선택
  - shard\_key 문자열이 'd', 'D'로 시작되는 경우 shard #3을 선택

```
// <shard_key_udf.c>

#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include "shard_key.h"

int
fn_shard_key_udf (const char *shard_key, T_SHARD_U_TYPE
type,
```

```
const void *value, int value_len)
{
 unsigned int ival;
 unsigned char c;

 if (value == NULL)
 {
 return ERROR_ON_ARGUMENT;
 }

 switch (type)
 {
 case SHARD_U_TYPE_INT:
 ival = (unsigned int) (*(unsigned int *) value);
 if (ival % 2)
 {
 return 32; // shard #1
 }
 else
 {
 return 0; // shard #0
 }
 break;

 case SHARD_U_TYPE_STRING:
 c = (unsigned char) (((unsigned char *) value)[0]);
 switch (c)
 {
 case 'a':
 case 'A':
 return 0; // shard #0
 case 'b':
 case 'B':
 return 32; // shard #1
 case 'c':
 case 'C':
 return 64; // shard #2
 case 'd':
 case 'D':
 return 96; // shard #3
 default:
 return ERROR_ON_ARGUMENT;
 }

 break;

 default:
 return ERROR_ON_ARGUMENT;
 }
 return ERROR_ON_MAKE_SHARD_KEY;
}
```

사용자 지정 해시 함수를 공유 라이브러리 형태로 빌드한다. 다음은 해시 함수 빌드를 위한 Makefile의 예이다.

```
Makefile

CC = gcc
LIBS = $(LIB_FLAG)
CFLAGS = $(CFLAGS_COMMON) -fPIC -I$(CUBRID)/include -I$(CUBRID_SRC)/src/broker

SHARD_CC = gcc -g -shared -Wl,-soname,shard_key_udf.so
SHARD_KEY_UDF_OBJS = shard_key_udf.o

all:$(SHARD_KEY_UDF_OBJS)
 $(SHARD_CC) $(CFLAGS) -o shard_key_udf.so $
 (SHARD_KEY_UDF_OBJS) $(LIBS)

clean:
 -rm -f *.o core shard_key_udf.so
```

사용자 정의 해시 함수를 포함하기 위해 **SHARD\_KEY\_LIBRARY\_NAME**, **SHARD\_KEY\_FUNCTION\_NAME** 파라미터를 위 구현과 일치하도록 수정한다.

```
[%student_no]
SHARD_KEY_LIBRARY_NAME = $CUBRID/conf/shard_key_udf.so
SHARD_KEY_FUNCTION_NAME = fn_shard_key_udf
```

#### 참고

- 응용 프로그램에서 사용자 해시 함수를 정의할 때 shard key의 입력 값으로 16bit(short), 32bit(int), 64bit(INT64) integer를 사용할 수 있다.
- VARCHAR를 사용해야 하는 경우 사용자가 해시 함수를 정의해야 한다.

## 구동 및 모니터링

cubrid broker 유틸리티를 이용하여 CUBRID SHARD 기능을 구동하거나 정지할 수 있고, 각종 상태 정보를 조회할 수 있다. 보다 자세한 내용은 [브로커](#)를 참고한다.

## 설정 테스트

cubrid broker test 명령을 이용하여 설정이 정상 동작하는지 테스트할 수 있다. 보다 자세한 내용은 [브로커와 DB 간 연결 테스트](#)를 참고한다.

## CUBRID SHARD 로그

SHARD 구동과 관련된 로그에는 접속 로그, 프록시 로그, SQL 로그, 에러 로그가 있다. 각각 로그의 저장 디렉터리 변경은 SHARD 환경 설정 파일( **cubrid\_broker.conf**) 의 **LOG\_DIR**, **ERROR\_LOG\_DIR**, **PROXY\_LOG\_DIR** 파라미터를 통해 설정할 수 있다.

## SHARD PROXY 로그

### 접속 로그

- 파라미터: **ACCESS\_LOG**
- 설명: 클라이언트의 접속을 logging한다(기존 브로커는 CAS에서 로그를 남긴다).
- 기본 저장 디렉터리: \$CUBRID/log/broker/
- 파일 이름: <broker\_name>\_<proxy\_index>.access
- 로그 형식: CAS에서 남기는 access log와 cas\_index 이외의 모든 string 동일

```
10.24.18.67 - - 1340243427.828 1340243427.828 2012/06/21 10:50:27 ~
2012/06/21 10:50:27 23377 - -1 shard1 shard1
10.24.18.67 - - 1340243427.858 1340243427.858 2012/06/21 10:50:27 ~
2012/06/21 10:50:27 23377 - -1 shard1 shard1
10.24.18.67 - - 1340243446.791 1340243446.791 2012/06/21 10:50:46 ~
2012/06/21 10:50:46 23377 - -1 shard1 shard1
10.24.18.67 - - 1340243446.821 1340243446.821 2012/06/21 10:50:46 ~
2012/06/21 10:50:46 23377 - -1 shard1 shard1
```

### 프록시 로그

- 파라미터: **SHARD\_PROXY\_LOG\_DIR**
- 설명: proxy 내부의 동작을 logging한다.
- 기본 저장 디렉터리: \$CUBRID/log/broker/proxy\_log
- 파일 이름: <broker\_name>\_<proxy\_index>.log

```
06/21 10:50:46.822 [SRD] ../../src/broker/shard_proxy_io.c(1045):
New socket io created. (fd:50).
06/21 10:50:46.822 [SRD] ../../src/broker/shard_proxy_io.c(2517):
New client connected. client(client_id:3, is_busy:Y, fd:50, ctx_cid:
3, ctx_uid:4).
06/21 10:50:46.825 [DBG] ../../src/broker/shard_proxy_io.c(3298):
Shard status. (num_cas_in_tran=1, shard_id=2).
06/21 10:50:46.827 [DBG] ../../src/broker/shard_proxy_io.c(3385):
Shard status. (num_cas_in_tran=0, shard_id=2).
```

## 프록시 로그 레벨

- 파라미터: **SHARD\_PROXY\_LOG**
- 프록시 로그 레벨 정책: 상위 level을 설정하면 하위의 모든 로그가 남는다.
  - 예) SCHEDULE을 설정하면, ERROR | TIMEOUT | NOTICE | SHARD | SCHEDULE 로그를 모두 남긴다.
- 프록시 로그 레벨 항목
  - NONE or OFF: 로그를 남기지 않는다.
  - ERROR(default): 내부적으로 에러가 발생하여 정상적으로 처리되지 못하는 경우
  - TIMEOUT: session timeout이나 query timeout 등의 timeout
  - NOTICE: 힌트 없는 query 및 기타 에러는 아닌 경우
  - SHARD: client 의 request가 어떤 shard의 어떤 CAS로 갔는지, 그것이 다시 client response 되었는지 등의 scheduling
  - SCHEDULE: 힌트 parsing 및 hash를 통해 shard key id 가져오는 것 등의 shard processing
  - ALL: 모든 로그

## SHARD CAS 로그

### SQL 로그

- 파라미터 : **SQL\_LOG**
- 설명 : prepare/execute/fetch 등의 query 및 기타 CAS 정보를 logging한다.
- 기본 저장 디렉터리 : \$CUBRID/log/broker/sql\_log
- 파일 이름 : <broker\_name>\_<proxy\_index>\_<shard\_index>\_<cas\_index>.sql.log



```

13-06-21 10:13:00.005 (0) STATE idle
13-06-21 10:13:01.035 (0) CAS TERMINATED pid 31595
13-06-21 10:14:20.198 (0) CAS STARTED pid 23378
13-06-21 10:14:21.227 (0) connect db shard1@HostA user dba url
shard1 session id 3
13-06-21 10:14:21.227 (0) DEFAULT isolation_level 3, lock_timeout -1
13-06-21 10:50:28.259 (1) prepare srv_h_id 1
13-06-21 10:50:28.259 (0) auto_rollback
13-06-21 10:50:28.259 (0) auto_rollback 0

```

## 에러 로그

- 파라미터 : **ERROR\_LOG\_DIR**
- 설명 : CUBRID의 경우 cs library에서 EID 및 error string을 해당 파일에 logging한다.
- 기본 저장 디렉터리 : \$CUBRID/log/broker/error\_log
- 파일 이름 : <broker\_name>\_<proxy\_index>\_<shard\_index>\_<cas\_index>.err

```

Time: 06/21/13 10:50:27.776 - DEBUG *** file ../../src/transaction/
boot_cl.c, line 1409
trying to connect 'shard1@localhost'
Time: 06/21/13 10:50:27.776 - DEBUG *** file ../../src/transaction/
boot_cl.c, line 1418
ping server with handshake
Time: 06/21/13 10:50:27.777 - DEBUG *** file ../../src/transaction/
boot_cl.c, line 966
boot_restart_client: register client { type 4 db shard1 user dba
password (null) program cubrid_cub_cas_1 login cubrid_user host
HostA pid 23270 }

```

## 제약 사항

### Linux만 지원

Linux에서만 CUBRID SHARD 기능을 사용할 수 있다.

### 하나의 트랜잭션은 하나의 shard DB에서만 수행 가능

하나의 트랜잭션은 오직 하나의 shard DB에서만 수행되어야 하며, 따라서 아래와 같은 제약사항이 존재한다.

- shard key 변경(**UPDATE**)으로 인해 여러 shard DB의 데이터를 변경하는 것은 불가능하며, 필요하다면 **DELETE** / **INSERT** 를 이용한다.

- 2개 이상의 샤드에 대한 절의(join, sub-query, or, union, group by, between, like, in, exist, any/some/all 등)를 지원하지 않는다.

### 세션 정보는 각 shard DB 내에서만 유효

세션 정보가 각 shard DB 내에서만 유효하므로, `LAST_INSERT_ID()` 와 같은 세션 관련 함수의 결과가 의도한 바와 다를 수 있다.

### SET NAMES 문 지원 안 함

SHARD 구성 환경에서는 SET NAMES 문이 정상 동작하지 않을 수 있으므로, 사용을 권장하지 않는다.

### auto increment는 각 shard DB 내에서만 유효

auto increment 속성 또는 SERIAL 등의 값이 각 shard DB 내에서만 유효하므로, 의도한 것과 다른 값을 반환할 수 있다.

### SHARD 힌트 구문이 작성된 DDL 구문은 지원 안함

SHARD 구성 환경에서 스키마 생성 및 변경 등의 DDL구문은 SHARD 힌트가 지원 되지 않으므로 각각의 SHARD DB에 접속하여 스키마 생성 및 변경을 처리하여야 한다. shard\_id(0)은 정상적으로 처리되며, shard\_id(1)부터는 에러를 발생시킨다.

오류 예시)

```
CREATE TABLE foo (col1 INT NOT NULL) /*+ SHARD_ID(1) /
```

```
DROP TABLE IF EXISTS foo /*+ SHARD_ID(1) */
```

## API 레퍼런스

이번 장에서는 다음과 같은 API에 대해 설명한다.

## JDBC 드라이버

### JDBC 개요

CUBRID JDBC 드라이버(cubrid\_jdbc.jar)를 사용하면 Java로 작성된 응용 프로그램에서 CUBRID 데이터베이스에 접속할 수 있다. CUBRID JDBC 드라이버는 <CUBRID 설치 디렉터리> /**jdbc** 디렉터리에 위

치한다. CUBRID JDBC 드라이버는 JDBC 2.0 스펙을 기준으로 개발되었으며, JDK 1.8에서 컴파일한 것을 기본으로 제공한다.

### CUBRID JDBC 드라이버 버전 확인

JDBC 드라이버 버전은 다음과 같은 방법으로 확인할 수 있다.

```
% jar -tf cubrid_jdbc.jar
META-INF/
META-INF/MANIFEST.MF
cubrid/
cubrid/jdbc/
cubrid/jdbc/driver/
cubrid/jdbc/jci/
cubrid/sql/
cubrid/jdbc/driver/CUBRIDBlob.class
...
CUBRID-JDBC-11.3.0.0047
```

### CUBRID JDBC 드라이버 등록

JDBC 드라이버 등록은 **Class.forName** (*driver-class-name*) 메서드를 사용하며, 아래는 CUBRID JDBC 드라이버를 등록하기 위해 cubrid.jdbc.driver.CUBRIDDriver 클래스를 로드하는 예제이다.

```
import java.sql.*;
import cubrid.jdbc.driver.*;

public class LoadDriver {
 public static void main(String[] Args) {
 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 } catch (Exception e) {
 System.err.println("Unable to load driver.");
 e.printStackTrace();
 }
 ...
 }
}
```

## JDBC 설치 및 설정

### 기본 환경

- JDK 1.8 이상
- CUBRID 2008 R2.0(8.2.0) 이상
- CUBRID JDBC 드라이버 2008 R1.0 이상

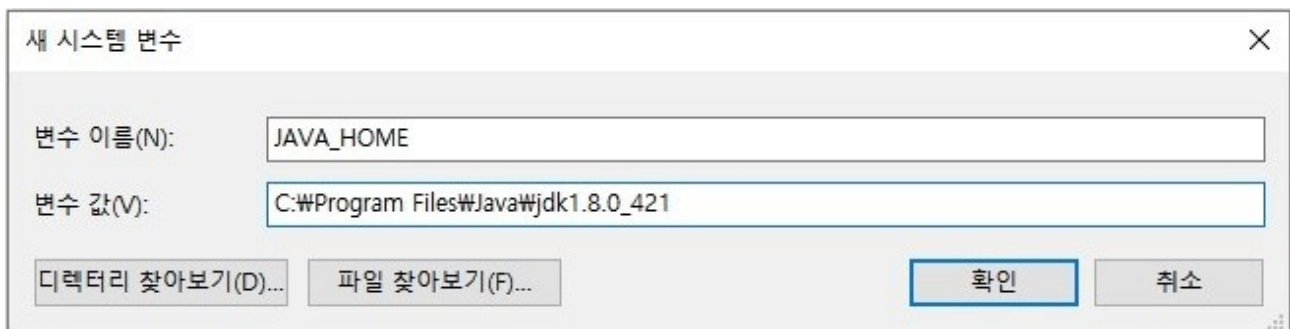
### Java 설치 및 환경 변수 설정

시스템에 Java가 설치되어 있고 **JAVA\_HOME** 환경 변수가 등록되어 있어야 한다. Java는 Developer Resources for Java Technology 사이트( <https://www.oracle.com/java/technologies/downloads/#java8> )에서 다운로드할 수 있다.

### Windows 환경에서 환경 변수 설정

Java 설치 후 [내 PC]를 마우스 오른쪽 버튼 클릭하여 [속성]을 선택한 후, 관련 설정의 [고급 시스템 설정]을 클릭하면 시스템 속성의 대화 상자가 나타난다. [고급]탭의 [환경 변수]를 클릭하면 [환경 변수] 대화 상자가 나타난다.

[시스템 변수]에서 [새로 만들기]를 선택한다. [변수 이름]에 **JAVA\_HOME** 을 입력하고, 변수 값으로 Java 설치 경로(예: C:\Program Files\Java\jdk1.8.0\_421)를 입력한 후 [확인]을 클릭한다.



[시스템 변수] 중 Path를 선택하고 [편집]을 클릭한다. [변수 값]에 **%JAVA\_HOME%\bin** 를 추가하고 [확인]을 클릭한다.



위의 방법을 사용하지 않고 다음과 같이 셸에서 **JAVA\_HOME** 과 **PATH** 환경 변수를 설정할 수도 있다.

```
set JAVA_HOME= C:\Program Files\Java\jdk1.8.0_421
set PATH=%PATH%;%JAVA_HOME%\bin
```

### Linux 환경에서 환경 변수 설정

다음과 같이 Java가 설치된 **JAVA\_HOME** 환경 변수로 디렉터리 경로(예: /usr/java/jdk1.8.0\_421)를 설정하고, **PATH** 환경 변수에 **\$JAVA\_HOME/bin** 을 추가한다.

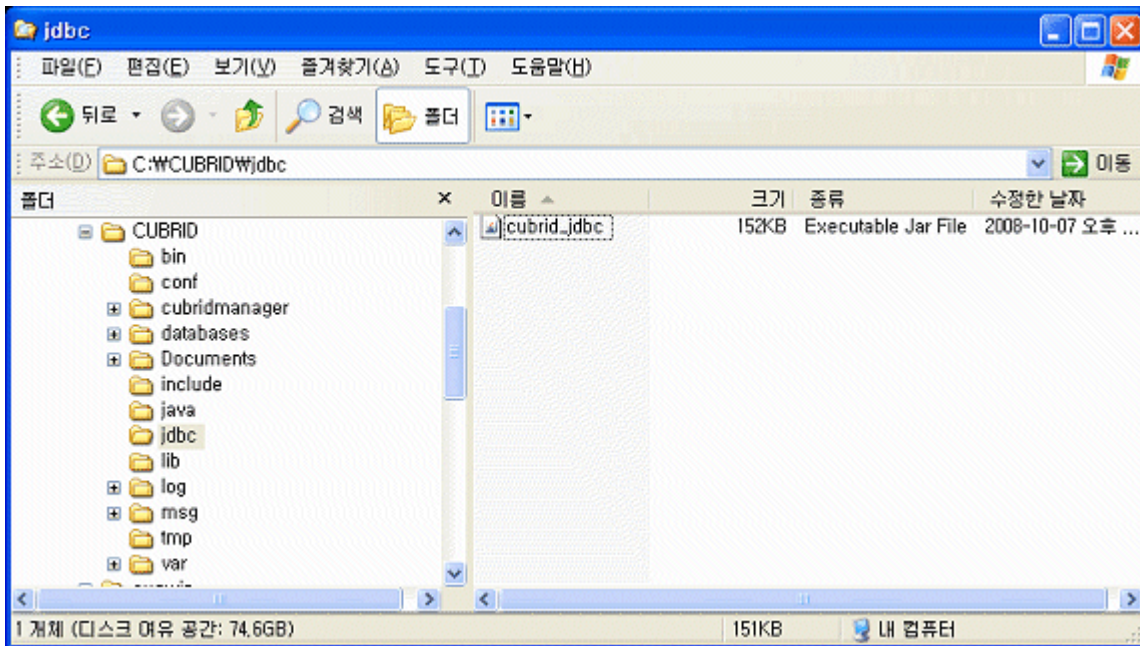
```
export JAVA_HOME=/usr/java/jdk1.8.0_421 #bash
export PATH=$JAVA_HOME/bin:$PATH #bash
```

```
setenv JAVA_HOME /usr/java/jdk1.8.0_421 #csh
set path = ($JAVA_HOME/bin $path) #csh
```

## JDBC 드라이버 설정

JDBC를 사용하려면 CUBRID JDBC 드라이버가 존재하는 경로를 환경 변수 **CLASSPATH**에 추가해야 한다.

CUBRID JDBC 드라이버(**cubrid\_jdbc.jar**)는 CUBRID 설치 디렉터리 아래의 jdbc 디렉터리에 위치한다.



## Windows 환경에서 CLASSPATH 환경 변수 설정

```
set CLASSPATH=C:\CUBRID\jdbc\cubrid_jdbc.jar:.
```

## Linux 환경에서 CLASSPATH 환경 변수 설정

```
export CLASSPATH=$HOME/CUBRID/jdbc/cubrid_jdbc.jar:.
```

### ⚠ 경고

만약 JRE가 설치된 라이브러리 디렉터리(**\$JAVA\_HOME/jre/lib/ext**)에 일반 CUBRID JDBC 드라이버가 설치되어 있다면, Java 저장 프로시저에서 사용하는 서버 사이드 JDBC 드라이버보다 먼저 로드되어 Java 저장 프로시저가 비정상적으로 구동될 수 있다. Java 저장 프로시저를 사용하는 환경에서는 JRE가 설치된 라이브러리 디렉터리(**\$JAVA\_HOME/jre/lib/ext**)에 일반 CUBRID JDBC 드라이버를 설치하지 않도록 주의한다.

## JDBC 프로그래밍

### 연결 설정

**DriverManager** 는 JDBC 드라이버를 관리하기 위한 기본적인 인터페이스이며, 데이터베이스 드라이버를 선택하고 새로운 데이터베이스 연결을 생성하는 기능을 한다. CUBRID JDBC 드라이버가 등록되어 있다면 **DriverManager.getConnection** (*db-url*, *user-id*, *password*) 메서드를 호출하여 데이터베이스에 접속한다.

**getConnection** 메서드는 **Connection** 객체를 반환한다. 그리고 그것은 질의 실행과 명령문 실행 그리고 트랜잭션의 커밋 또는 롤백에 사용된다. 연결 설정을 위한 *db-url* 인자의 구성은 다음과 같다.

```
jdbc:cubrid:<host>:<port>:<db-name>:[user-id]:[password]:[?
<property> [& <property>] ...]

<host> ::=
hostname | ip_address

<property> ::= altHosts=<alternative_hosts>
 | rcTime=<second>
 | loadBalance=<balance_mode>
 | connectTimeout=<second>
 | queryTimeout=<second>
 | charSet=<character_set>
 | zeroDateTimeBehavior=<behavior_type>
 | logOnException=<bool_type>
 |
logSlowQueries=<bool_type>&slowQueryThresholdMillis=<millisecond>
 | useLazyConnection=<bool_type>
 | useSSL=<bool_type>
 | clientCacheSize=<unit_size>
 | usePreparedStmtCache=<bool_type>
 | preparedStmtCacheSize=<unit_size>
 | preparedStmtCacheSqlLimit=<unit_size>
 | hold_cursor=<bool_type>

<alternative_hosts> ::=
<standby_broker1_host>:<port> [,<standby_broker2_host>:<port>]
<behavior_type> ::= exception | round | convertToNull
<bool_type> ::= true | false
<unit_size> ::= multiple of mega byte
<balance_mode> ::= true | false | rr | sh
```

- *host*: CUBRID 브로커가 동작하고 있는 서버의 IP 주소 또는 호스트 이름
- *port*: CUBRID 브로커의 포트 번호(기본값: 33000)
- *db-name*: 접속할 데이터베이스 이름

- **user-id**: 데이터베이스에 접속할 사용자 ID이다. 기본적으로 데이터베이스에는 **dba** 와 **public** 두 개의 사용자가 존재한다. 이 값이 NULL이면 **db-url**의 사용자 ID가 사용되며, 빈 문자열("")이면 **public**이 사용자 ID로 사용된다.
- **password**: 데이터베이스에 접속할 사용자의 암호이다. 이 값이 NULL이면 url의 암호가 사용되며, 빈 문자열("")이면 빈 문자열이 암호로 사용된다. **db-url** 내의 암호에는 ':'를 포함할 수 없다.
- **<property>**
  - **altHosts**: HA 환경에서 장애 시 fail-over할 하나 이상의 standby 브로커의 호스트 IP와 접속 포트이다.

#### 참고

메인 호스트와 **altHosts** 브로커들의 **ACCESS\_MODE** 설정에 **RW**와 **RO**가 섞여 있다 하더라도, 응용 프로그램은 **ACCESS\_MODE**와 무관하게 접속 대상 호스트를 결정한다. 따라서 사용자는 접속 대상 브로커의 **ACCESS\_MODE**를 감안해서 메인 호스트와 **altHosts**를 정해야 한다.

- **rcTime**: 첫 번째로 접속했던 브로커에 장애가 발생한 이후 **altHosts** 에 명시한 브로커로 접속한다(failover). 이후, rcTime만큼 시간이 경과할 때마다 원래의 브로커에 재접속을 시도한다(기본값 600초). 입력 방법은 아래 URL 예제를 참고한다.
- **loadBalance**: 클라이언트가 접속 가능한 다수의 호스트가 있는 경우, 접속 대상 호스트를 결정하는 알고리즘을 선택하는 속성이다. 로드 밸런싱을 통해 클라이언트의 연결 요청이 특정 서버에 집중되지 않도록 분산할 수 있다.(기본값: false)

설정값에 따른 동작은 다음과 같다.

- **false**: 해당 속성이 적용하지 않는다.
- **true** 또는 **rr**: 지정한 호스트들을 **Round-Robin** 방식으로 연결한다.
- **sh**: 지정한 호스트들을 무작위(**Shuffle**) 방식으로 연결한다.
- **connectTimeout**: 데이터베이스 접속에 대한 타임아웃 시간을 초 단위로 설정한다. 기본값은 30초이다. 이 값이 0인 경우 무한 대기를 의미한다. 이 값은 최초 접속 이후 내부적인 재접속이 발생하는 경우에도 적용된다. **DriverManger.setLoginTimeout ()** 메서드로 설정할 수도 있으나, 연결 URL에 이 값을 설정하면 메서드로 설정한 값은 무시된다.
- **queryTimeout**: 질의 수행에 대한 타임아웃 시간을 초 단위로 설정한다(기본값: 0, 무제한). 최대 값은 2,000,000이다. 이 값은 **DriverManger.setQueryTimeout ()** 메서드에 의해 변경될 수 있다. **executeBatch()** 메서드를 수행하는 경우 한 개의 질의에 대한 타임아웃이 아닌 한 번의 메서드 호출에 대한 타임아웃이 적용된다.

### 참고

executeBatch() 메서드를 수행하는 경우 한 개의 질의에 대한 타임아웃이 아닌 한 번의 메서드 호출에 대한 타임아웃이 적용된다.

- **charSet**: 접속하고자 하는 DB의 문자셋(charSet)이다.
- **zeroDateTimeBehavior**: JDBC에서는 java.sql.Date 형 객체에 날짜와 시간 값이 모두 0인 값을 허용하지 않으므로 이 값을 출력해야 할 때 어떻게 처리할 것인지를 정하는 속성. 기본 동작은 **exception** 이다. 날짜와 시간 값이 모두 0인 값에 대한 설명은 **날짜/시간 데이터 타입** 을 참고한다.

설정값에 따른 동작은 다음과 같다.

- **exception**: 기본 동작. SQLException 예외로 처리한다.
- **round**: 반환할 타입의 최소값으로 변환한다. 단, TIMESTAMP 타입은 '1970-01-01 00:00:00'(GST)를 반환한다.
- **convertToNull**: NULL 로 변환한다.
- **logOnException**: 디버깅용 예외 처리 로깅 여부(기본값: true). 출력 위치와 레벨은 **드라이버 로그 설정**을 참고한다.
- **logSlowQueries**: 디버깅용 슬로우 쿼리 로깅 여부(기본값: false). 출력 위치와 레벨은 **드라이버 로그 설정**을 참고한다.
  - **slowQueryThresholdMillis**: 디버깅용 슬로우 쿼리 로깅 시 슬로우 쿼리 제한 시간(기본값: 60000). 단위는 밀리 초이다.

### 참고

**logFile** 속성은 더 이상 사용되지 않는다. 연결 URL에 이 속성이 남아 있어도 오류 없이 무시되며, 드라이버 로그의 출력 위치는 응용 프로그램의 **java.util.logging** 설정이 결정한다(**드라이버 로그 설정** 참고).

- **useLazyConnection**: 이 값이 true이면 사용자의 연결 요청 시 브로커 연결 없이 성공을 반환(기본값: false)하고, prepare나 execute 등의 함수를 호출할 때 브로커에 연결한다. 이 값을 true로 설정하면 많은 응용 클라이언트가 동시에 재시작되면서 연결 풀(connection pool)을 생성할 때 접속이 지연되거나 실패하는 현상을 피할 수 있다.



- **useSSL**: 패킷 암호화 여부 (기본값: false)
  - 패킷 암호화: useSSL = true
  - 일반 평문: useSSL = false
- **clientCacheSize**: 결과를 캐시할 크기 \* 단위는 메가 바이트 \* 범위는 1 ~ 1024 (1메가 바이트에서 1기가 바이트) \* 기본 값은 1 (메가 바이트)
- **usePreparedStmtCache**: Prepared Statement 캐시 여부 (기본값: false)
- **preparedStmtCacheSize**: usePreparedStmtCache가 TRUE일 경우, 캐싱할 수 있는 개수 (기본: 25, 최소:1, 최대:2147483647)
- **preparedStmtCacheSqlLimit**: usePreparedStmtCache가 TRUE일 경우, 캐싱할 수 있는 SQL의 길이 (기본:256, 최소:1, 최대:2147483647)
- **hold\_cursor**: 커서 유지 기능 설정(기본값: true). 이 값이 false 이면 CLOSE\_CURSORS\_AT\_COMMIT이 설정되고, true 이면 HOLD\_CURSORS\_OVER\_COMMIT이 설정된다. 자세한 내용은 **커서 유지**을 참고한다.

## 예제 1

```
--connection URL string when user name and password omitted
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::

--connection URL string when zeroDateTimeBehavior property specified
URL=jdbc:CUBRID:127.0.0.1:33000:demodb:public:??
zeroDateTimeBehavior=convertToNull

--connection URL string when charSet property specified
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public:??charSet=utf-8

--connection URL string when queryTimeout and charSet property
specified
URL=jdbc:CUBRID:127.0.0.1:33000:demodb:public:??
queryTimeout=18charSet=utf-8

--connection URL string when a property(altHosts) specified for HA
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public:??
altHosts=192.168.0.2:33000,192.168.0.3:33000

--connection URL string when properties(altHosts,rcTime,
connectTimeout) specified for HA
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public:??
altHosts=192.168.0.2:33000,192.168.0.3:33000&rcTime=600&connectTimeou
t=5
```

```
--connection URL string when properties(altHosts,rcTime, charSet)
specified for HA
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::?
altHosts=192.168.0.2:33000,192.168.0.3:33000&rcTime=600&charSet=utf-8

--connection URL string when useSSL property specified for encrypted
connection
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::?useSSL=true

--connection URL string when clientCacheSize property specified for
result-cache
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::?clientCacheSize=1

--connection URL string when usePreparedStmtCache property specified
for prepared stament cache
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::?
usePreparedStmtCache=true&preparedStmtCacheSize=100&preparedStmtCache
SqlLimit=1024"

--connection URL string when hold_cursor property specified for
cursor holdability
URL=jdbc:CUBRID:192.168.0.1:33000:demodb:public::?hold_cursor=true
```

## 예제 2

```
String url = "jdbc:cubrid:192.168.0.1:33000:demodb:public::";
String userid = "";
String password = "";

try {
 Connection conn =
 DriverManager.getConnection(url,userid,password);
 // Do something with the Connection

 ...

} catch (SQLException e) {
 System.out.println("SQLException: " + e.getMessage());
 System.out.println("SQLState: " + e.getSQLState());
}

...
```

### 참고

- URL 문자열에서 콜론(:)과 물음표(?)는 구분자로 사용되므로, URL 문자열에 암호를 포함하는 경우 암호의 일부에 콜론이나 물음표를 사용할 수 없다. 암호에 콜론이나 물음표를 사용하려

면 getConnection 함수에서 사용자 이름(*user-id*)과 암호(*password*)를 별도의 인자로 지정해야 한다.

- 스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.
- 트랜잭션 롤백을 요청하는 rollback 메서드는 서버가 롤백 작업을 완료한 후 종료된다.
- 자동 커밋 모드에서 SELECT 문 수행 이후 모든 결과 셋이 fetch되지 않으면 커밋이 되지 않는다. 따라서, 자동 커밋 모드라 하더라도 프로그램 내에서 결과 셋에 대한 fetch 도중 어떠한 오류가 발생한다면 반드시 커밋 또는 롤백을 수행하여 트랜잭션을 종료 처리하도록 한다.

### ⚠ 경고

- useSSL의 flag는 **브로커 모드와 일치해야 한다**. 아래와 같이 브로커의 암호화 모드와 다른 flag로 접속을 요청하는 경우 **연결되지 않는다**.
  - useSSL=true, 브로커 '일반 모드' 일 때 연결 불가 ( **cubrid\_broker.conf**: SSL = OFF)
  - useSSL=false, 브로커 '암호화 모드' 일때 연결 불가 ( **cubrid\_broker.conf**: SSL = ON)
- **clientCacheSize** 는 브로커 파라미터인 **JDBC\_CACHE** 혹은 **JDBC\_CACHE\_ONLY\_HINT** 가 **ON** 으로 설정되어 있어야 유효하다.

## 드라이버 로그 설정

**logOnException** 과 **logSlowQueries** 로 기록되는 드라이버 로그는 **java.util.logging** 을 통해 출력된다. 로그를 어디에 남길지는 응용 프로그램의 **java.util.logging** 설정이 결정하며, 드라이버는 로그 파일을 직접 만들지 않는다.

로그는 **cubrid.jdbc.jci.UConnection** 로거로 출력되며, 설정은 상위 로거인 **cubrid.jdbc** 에 한다. 로그 종류에 따라 레벨이 다르다.

| 로그    | 레벨   | 내용              |
|-------|------|-----------------|
| 예외 덤프 | FINE | 예외 클래스와 스택 트레이스 |

| 로그     | 레벨     | 내용                            |
|--------|--------|-------------------------------|
| 슬로우 쿼리 | FINEST | CAS 정보, 수행 시간, SQL 문장, 바인딩된 값 |

두 레벨은 모두 JDK의 기본 임계값인 **INFO** 보다 낮으므로, **cubrid.jdbc**의 레벨을 설정하지 않으면 아무것도 출력되지 않는다.

예외 덤프는 **logOnException**의 기본값이 **true** 이므로 레벨만 설정하면 출력된다. 출력하지 않으려면 연결 URL에 **logOnException=false** 를 지정한다. 반면 슬로우 쿼리는 드라이버가 매 질의의 수행 시간을 측정해야 하므로, **logSlowQueries** 를 **true** 로 설정해야 출력된다.

```
handlers = java.util.logging.FileHandler
cubrid.jdbc.level = FINEST
java.util.logging.FileHandler.pattern = /var/log/myapp/
cubrid_jdbc%.log
java.util.logging.FileHandler.limit = 10485760
java.util.logging.FileHandler.count = 10
java.util.logging.FileHandler.append = true
java.util.logging.FileHandler.formatter =
java.util.logging.SimpleFormatter
```

**%g** 는 로그 파일의 세대 번호이다. 위 설정에서는 파일이 **cubrid\_jdbc0.log**부터 **cubrid\_jdbc9.log**까지 생성되며, 가장 최근 로그는 항상 **cubrid\_jdbc0.log** 이다. **limit** 를 지정하지 않으면 로그 파일의 크기가 제한되지 않는다.

위와 같이 작성한 설정 파일은 다음과 같이 지정한다.

```
java -Djava.util.logging.config.file=logging.properties -jar app.jar
```

**cubrid.jdbc.level** 을 **FINE** 으로 설정하면 예외 덤프만 출력되고, **FINEST** 로 설정하면 슬로우 쿼리까지 출력된다.

### 참고

- **FileHandler**의 기본 포맷터는 **XMLFormatter** 이다. 위 예제와 달리 **formatter** 를 지정하지 않으면 로그가 한 줄 단위 텍스트가 아닌 XML 형식으로 기록된다.
- 슬로우 쿼리 로그에는 SQL 문장과 바인딩된 값이 그대로 포함된다. **cubrid.jdbc.level** 을 **FINEST** 로 설정할 때에는 로그 파일의 접근 권한을 확인해야 한다.

- 드라이버 로그를 응용 프로그램의 로그와 분리하려면 **cubrid.jdbc.handlers** 를 지정하고 **cubrid.jdbc.useParentHandlers** 를 **false** 로 설정한다.

## DataSource 객체로 연결

DataSource는 JDBC 2.0 확장 API에 소개된 개념으로, 연결 풀링(connection pooling)과 분산 트랜잭션을 지원한다. CUBRID는 연결 풀링만 지원하며, 분산 트랜잭션과 JNDI는 지원하지 않는다.

CUBRIDDataSource는 CUBRID에서 구현한 DataSource이다.

## DataSource 객체 생성하기

DataSource 객체를 생성하려면 다음과 같이 호출한다.

```
CUBRIDDataSource ds = null;
ds = new CUBRIDDataSource();
```

## 연결 속성 설정하기

**연결 속성**(connection properties)은 datasource와 CUBRID DBMS 사이에 연결을 설정하는데 사용된다. 일반적인 속성은 DB 이름, 호스트 이름, 포트 번호, 사용자 이름, 암호이다.

속성(property) 값을 설정하거나 얻기 위해서는 cubrid.jdbc.driver.CUBRIDDataSource에서 구현된 다음 메서드들을 사용한다.

```
public PrintWriter getLogWriter();
public void setLogWriter(PrintWriter out);
public void setLoginTimeout(int seconds);
public int getLoginTimeout();
public String getDatabaseName();
public String getDatabaseName();
public String getDataSourceName();
public String getDescription();
public String getNetworkProtocol();
public String getPassword();
public int getPortNumber();
public int getPort();
public String getRoleName();
public String getServerName();
public String getUser();
public String getURL();
public String getUrl();
public void setDatabaseName(String dbName);
public void setDescription(String desc);
public void setNetworkProtocol(String netProtocol);
```

```
public void setPassword(String psswd);
public void setPortNumber(int p);
public void setPort(int p);
public void setRoleName(String rName);
public void setServerName(String svName);
public void setUser(String uName);
public void setUrl(String urlString);
public void setURL(String urlString);
```

특히, URL 문자열을 통해 속성을 지정하고자 하는 경우 setURL() 메서드를 사용한다. URL 문자열에 대해서는 [연결 설정](#)을 참고한다.

```
import cubrid.jdbc.driver.CUBRIDDataSource;
...
CUBRIDDataSource ds = null;
ds = new CUBRIDDataSource();
ds.setUrl("jdbc:cubrid:192.168.0.1:55300:demodb::?
charset=utf8&logSlowQueries=true&slowQueryThresholdMillis=1000");
```

DataSource로부터 연결 객체를 얻기 위해서는 getConnection 메서드를 호출한다.

```
Connection connection = null;
connection = ds.getConnection("dba", "");
```

CUBRIDConnectionPoolDataSource는 connectionpool datasource를 CUBRID에서 구현한 객체인데, CUBRIDDataSource의 메서드들과 같은 이름의 메서드들을 포함하고 있다.

보다 자세한 예제는 [JDBC 예제 프로그램](#)의 **DataSource 객체로 연결**을 참고한다.

## SQL LOG 확인

cubrid.jdbc.driver.CUBRIDConnection 클래스의 toString() 메서드를 사용하여 다음과 같은 연결 정보를 출력할 수 있다.

```
예) cubrid.jdbc.driver.CUBRIDConnection(CAS ID : 1, PROCESS ID :
22922)
```

위에서 출력되는 CAS ID를 통해 해당 CAS의 SQL 로그 파일을 쉽게 확인할 수 있다.

보다 자세한 사항은 [SQL 로그 확인](#)을 참고한다.

## 외래 키 정보 확인

**DatabaseMetaData** 인터페이스에서 제공되는 **getImportedKeys**, **getExportedKeys**, **getCrossReference** 메서드를 사용하여 외래 키 정보를 확인할 수 있다. 각 메서드의 사용법 및 예제는 다음과 같다.

```
getImportedKeys(String catalog, String schema, String table)

getExportedKeys(String catalog, String schema, String table)

getCrossReference(String parentCatalog, String parentSchema, String
parentTable, String foreignCatalog, String foreignSchema, String
foreignTable)
```

- **getImportedKeys** 메서드: 인자로 주어진 테이블의 외래 키 칼럼들이 참조하고 있는 기본 키 칼럼들의 정보를 조회한다. 결과는 **PKTABLE\_NAME** 및 **KEY\_SEQ** 순서로 정렬되어 반환된다.
- **getExportedKeys** 메서드: 주어진 테이블의 기본 키 칼럼들을 참조하는 모든 외래 키 칼럼들의 정보를 조회하며, 결과는 **FKTABLE\_NAME** 및 **KEY\_SEQ** 순서로 정렬된다.
- **getCrossReference** 메서드: 인자로 주어진 테이블의 외래 키 칼럼들이 참조하고 있는 기본 키 칼럼들의 정보를 조회한다. 결과는 **PKTABLE\_NAME** 및 **KEY\_SEQ** 순서로 정렬되어 반환된다.

## 반환 값

위 메서드를 호출하면 아래와 같이 14개의 칼럼으로 구성된 ResultSet을 반환한다.

| name          | type   | 비고          |
|---------------|--------|-------------|
| PKTABLE_CAT   | String | 항상 null     |
| PKTABLE_SCHEM | String | 항상 null     |
| PKTABLE_NAME  | String | 기본 키 테이블 이름 |
| PKCOLUMN_NAME | String | 기본 키 칼럼 이름  |
| FKTABLE_CAT   | String | 항상 null     |
| FKTABLE_SCHEM | String | 항상 null     |

| name          | type   | 비고                                                                                               |
|---------------|--------|--------------------------------------------------------------------------------------------------|
| FKTABLE_NAME  | String | 외래 키 테이블 이름                                                                                      |
| FKCOLUMN_NAME | String | 외래 키 칼럼 이름                                                                                       |
| KEY_SEQ       | short  | 외래 키 또는 기본 키 칼럼들의<br>순서(1부터 시작)                                                                  |
| UPDATE_RULE   | short  | 기본 키가 업데이트될 때 외래<br>키에 대해 정의된 참조 동작에<br>대응되는 값 Cascade=0, Restrict=2, No action=3, Set<br>null=4 |
| DELETE_RULE   | short  | 기본 키가 삭제될 때 외래 키<br>에 대해 정의된 참조 동작에<br>대응되는 값 Cascade=0, Restrict=2, No action=3, Set<br>null=4  |
| FK_NAME       | String | 외래 키 이름                                                                                          |
| PK_NAME       | String | 기본 키 이름                                                                                          |
| DEFERRABILITY | short  | 항상 6<br>(DatabaseMetaData.importedKeyInitiallyImmediate)                                         |

## 예제

```
import java.sql.*;
import java.lang.*;
import cubrid.jdbc.driver.*;

public class Metadata_Sample
```



```

{
 public static String getRuleName(String rule) {
 int r = Integer.parseInt(rule);
 switch(r) {
 case DatabaseMetaData.importedKeyCascade:
 return "cascade (" + rule + ")";
 case DatabaseMetaData.importedKeyRestrict:
 return "restrict (" + rule + ")";
 case DatabaseMetaData.importedKeySetNull:
 return "set null (" + rule + ")";
 case DatabaseMetaData.importedKeyNoAction:
 return "no action (" + rule + ")";
 case DatabaseMetaData.importedKeySetDefault:
 return "set default (" + rule + ")";
 default:
 return "unknown (" + rule + ")";
 }
 }

 public static String getDeferrabilityName(String
deferrability) {
 int d = Integer.parseInt(deferrability);
 switch(d) {
 case
DatabaseMetaData.importedKeyInitiallyDeferred:
 return "initially deferred (" +
deferrability + ")";
 case
DatabaseMetaData.importedKeyInitiallyImmediate:
 return "initially immediate (" +
deferrability + ")";
 case
DatabaseMetaData.importedKeyNotDeferrable:
 return "not deferrable (" +
deferrability + ")";
 default:
 return "unknown (" + deferrability +
")";
 }
 }

 public static void printFkInfo(ResultSet rs) throws
Exception {
 while(rs.next()) {
 System.out.println("\n<" +
rs.getString("FK_NAME") + ">");
 System.out.println("01. PKTABLE_CAT : " +
rs.getString("PKTABLE_CAT"));
 System.out.println("02. PKTABLE_SCHEM : " +
rs.getString("PKTABLE_SCHEM"));
 System.out.println("03. PKTABLE_NAME : " +
rs.getString("PKTABLE_NAME"));
 }
 }
}

```

```

 System.out.println("04. PKCOLUMN_NAME : " +
rs.getString("PKCOLUMN_NAME"));
 System.out.println("05. FKTABLE_CAT : " +
rs.getString("FKTABLE_CAT"));
 System.out.println("06. FKTABLE_SCHEM : " +
rs.getString("FKTABLE_SCHEM"));
 System.out.println("07. FKTABLE_NAME : " +
rs.getString("FKTABLE_NAME"));
 System.out.println("08. FKCOLUMN_NAME : " +
rs.getString("FKCOLUMN_NAME"));
 System.out.println("09. KEY_SEQ : " +
rs.getString("KEY_SEQ"));
 System.out.println("10. UPDATE_RULE : " +
getRuleName(rs.getString("UPDATE_RULE")));
 System.out.println("11. DELETE_RULE : " +
getRuleName(rs.getString("DELETE_RULE")));
 System.out.println("12. FK_NAME : " +
rs.getString("FK_NAME"));
 System.out.println("13. PK_NAME : " +
rs.getString("PK_NAME"));
 System.out.println("14. DEFERRABILITY : " +
getDeferrabilityName(rs.getString("DEFERRABILITY")));
 }
}

public static void main (String args []) {
 Connection conn = null;
 Statement stmt;
 DatabaseMetaData dbmd;
 ResultSet rs;

 try {
 conn =
(CUBRIDConnection)DriverManager.getConnection("jdbc:cubrid:localhost:
33000:demodb::", "dba", "");

 stmt = conn.createStatement();

 dbmd = conn.getMetaData();

 stmt.executeUpdate("drop table if exists
refer_1");
 stmt.executeUpdate("drop table if exists
refer_2");
 stmt.executeUpdate("drop table if exists
referred_1");
 stmt.executeUpdate("drop table if exists
referred_2");

 stmt.executeUpdate("create table referred_1
(
 + "pk1 int,"
 + "pk2 int,"

```

```

+ "primary key (pk1, pk2))");
stmt.executeUpdate("create table referred_2
("
+ "pk int primary key)");
stmt.executeUpdate("create table refer_1 ("
+ "pk int primary key,"
+ "fk1 int,"
+ "fk2 int,"
+ "fk3 int,"
+ "constraint
refer_1_fk1_2_fk foreign key (fk1, fk2) references referred_1(pk1,
pk2),"
+ "constraint refer_1_fk3_fk
foreign key (fk3) references referred_2(pk))");
stmt.executeUpdate("create table refer_2 ("
+ "pk int primary key,"
+ "fk1 int,"
+ "fk2 int,"
+ "fk3 int,"
+ "constraint
refer_2_fk1_2_fk foreign key (fk1, fk2) references referred_1(pk1,
pk2) "
+ "on update set null on
delete no action,"
+ "constraint refer_2_fk3_fk
foreign key (fk3) references referred_2(pk) "
+ "on update restrict on
delete cascade)");

System.out.println("\n=====
getImportedKeys() =====");

rs = dbmd.getImportedKeys(null, null,
"refer_1");
printFkInfo(rs);
rs.close();

rs = dbmd.getImportedKeys(null, null,
"refer_2");
printFkInfo(rs);
rs.close();

System.out.println("\n=====
getExportedKeys() =====");
rs = dbmd.getExportedKeys(null, null,
"referred_1");
printFkInfo(rs);
rs.close();

```

```

 rs = dbmd.getExportedKeys(null, null,
"referred_2");
 printFkInfo(rs);
 rs.close();

 System.out.println("\n=====
getCrossReference() =====");
 rs = dbmd.getCrossReference(null, null,
"referred_1", null, null, "refer_1");
 printFkInfo(rs);
 rs.close();

 rs = dbmd.getCrossReference(null, null,
"referred_1", null, null, "refer_2");
 printFkInfo(rs);
 rs.close();

 rs = dbmd.getCrossReference(null, null,
"referred_2", null, null, "refer_1");
 printFkInfo(rs);
 rs.close();

 rs = dbmd.getCrossReference(null, null,
"referred_2", null, null, "refer_2");
 printFkInfo(rs);
 rs.close();

 rs = dbmd.getCrossReference(null, null,
"refer_1", null, null, "refer_2");
 printFkInfo(rs);
 rs.close();
 } catch (Exception e) {
 e.printStackTrace();
 }
}
}

```

## OID와 컬렉션 사용

JDBC 스펙에 정의된 메서드 이외에 CUBRID JDBC 드라이버에서 추가로 OID, 컬렉션 타입( **SET**, **MULTISET**, **LIST**) 등을 다루는 메서드를 제공한다.

이 메서드의 사용을 위해서는 기본적으로 import하는 CUBRID JDBC 드라이버 클래스 이외에 **cubrid.sql.\***; 를 import해야 한다. 또한 표준 JDBC API에서 제공하는 **ResultSet** 클래스가 아닌 **CUBRIDResultSet** 클래스로 변환하여 결과를 받아야 한다.

```

import cubrid.jdbc.driver.* ;
import cubrid.sql.* ;
...

```

```
CUBRIDResultSet urs = (CUBRIDResultSet) stmt.executeQuery(
 "SELECT city FROM location");
```

### ⚠ 경고

CUBRID의 확장 API를 사용하면, **AUTOCOMMIT** 을 TRUE로 설정하였더라도 자동으로 커밋되지 않는다. 따라서 항상 open한 연결에 대해 명시적으로 커밋을 해야 한다. CUBRID 확장 API는 OID, 컬렉션 등을 다루는 메서드이다.

## OID 사용

OID를 사용할 때 다음의 규칙을 지켜야 한다.

- **CUBRIDOID** 를 사용하기 위해서는 반드시 **cubrid.sql.\*** 를 import 해야 한다. (a)
- **SELECT** 문에 클래스명을 주어 OID를 가져올 수 있다. 물론 다른 속성과 혼용해서 사용할 수도 있다. (b)
- 질의에 대한 **ResultSet** 은 반드시 **CUBRIDResultSet** 으로 받아야 한다. (c)
- **CUBRIDResultSet** 에서 OID를 가져오는 메서드는 **getOID()**이다. (d)
- OID에서 값을 가져오기 위해서는 **getValues()** 메서드를 통해 가져올 수 있다. 그 결과는 **ResultSet** 이다. (e)
- OID에 값을 대입하기 위해서는 **setValues()** 메서드를 통해서 적용할 수 있다. (f)
- 확장 API 사용시에는 연결에 대해 항상 **commit()**을 해주어야 한다. (g)

## 예제

```
import java.sql.*;
import cubrid.sql.*; //a
import cubrid.jdbc.driver.*;

/*
CREATE TABLE oid_test(
 id INTEGER,
 name VARCHAR(10),
 age INTEGER
);

INSERT INTO oid_test VALUES(1, 'Laura', 32);
INSERT INTO oid_test VALUES(2, 'Daniel', 39);
INSERT INTO oid_test VALUES(3, 'Stephen', 38);
```



```

 CUBRIDOID oid = rs.getOID(1); //d
 System.out.print("OID");
 System.out.print(" | ");
 rsoid = (CUBRIDResultSet)oid.getValues(attr); //e

 while (rsoid.next()) {
 for(int j=1; j <= attr.length; j++) {
 System.out.print(rsoid.getObject(j));
 System.out.print(" | ");
 }
 }
 System.out.print("\n");

 // New values of the first row
 Object[] value = { 4, "Yu-ri", 19 };
 if (k == 1) oid.setValues(attr, value); //f

 k = 0;
 }
 con.commit(); //g

} catch(CUBRIDException e) {
 e.printStackTrace();

} catch(SQLException ex) {
 ex.printStackTrace();

} finally {
 if(rs != null) try { rs.close(); } catch(SQLException e) {}
 if(stmt != null) try { stmt.close(); } catch(SQLException
e) {}

 if(con != null) try { con.close(); } catch(SQLException e)
{}

}

}
}

```

## 컬렉션 사용

아래 예제 1의 'a'에 해당하는 부분이 **CUBRIDResultSet** 으로부터 컬렉션 타입(**SET**, **MULTISET**, **LIST**)의 데이터를 가져오는 부분으로 그 결과는 배열 형태로 반환한다. 단, 컬렉션 타입 내에 정의된 원소들의 데이터 타입이 모두 같은 경우에만 사용할 수 있다.

### 예제 1

```

import java.sql.*;
import java.lang.*;
import cubrid.sql.*;
import cubrid.jdbc.driver.*;

```

```

// create class collection_test(
// settest set(integer),
// multisettest multiset(integer),
// listtest list(Integer)
//);
//

// insert into collection_test values({1,2,3},{1,2,3},{1,2,3});
// insert into collection_test values({2,3,4},{2,3,4},{2,3,4});
// insert into collection_test values({3,4,5},{3,4,5},{3,4,5});

class Collection_Sample
{
 public static void main (String args [])
 {
 String url= "jdbc:cubrid:127.0.0.1:33000:demodb:public::";
 String user = "";
 String passwd = "";
 String sql = "select settest,multisettest,listtest from
collection_test";
 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 } catch(Exception e){
 e.printStackTrace();
 }
 try {
 Connection con =
DriverManager.getConnection(url,user,passwd);
 Statement stmt = con.createStatement();
 CUBRIDResultSet rs = (CUBRIDResultSet)
stmt.executeQuery(sql);
 CUBRIDResultSetMetaData rsmd = (CUBRIDResultSetMetaData)
rs.getMetaData();
 int numbofColumn = rsmd.getColumnCount();
 while (rs.next ()) {
 for (int j=1; j<=numbofColumn; j++) {
 Object[] reset = (Object[])
rs.getCollection(j); //a
 for (int m=0 ; m < reset.length ; m++)
 System.out.print(reset[m] +",");
 System.out.print(" | ");
 }
 System.out.print("\n");
 }
 rs.close();
 stmt.close();
 con.close();
 } catch(SQLException e) {
 e.printStackTrace();
 }
 }
}

```



```

 }
}

```

## 예제 2

```

import java.sql.*;
import java.io.*;
import java.lang.*;
import cubrid.sql.*;
import cubrid.jdbc.driver.*;

// create class collection_test(
// settest set(integer),
// multisettest multiset(integer),
// listtest list(Integer)
//);
//
// insert into collection_test values({1,2,3},{1,2,3},{1,2,3});
// insert into collection_test values({2,3,4},{2,3,4},{2,3,4});
// insert into collection_test values({3,4,5},{3,4,5},{3,4,5});

class SetOP_Sample {
 public static void main(String args[]) {
 String url = "jdbc:cubrid:
127.0.0.1:33000:demodb:public::";
 String user = "";
 String passwd = "";
 String sql = "select collection_test from
collection_test";
 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 } catch (Exception e) {
 e.printStackTrace();
 }
 try {
 CUBRIDConnection con = (CUBRIDConnection)
 DriverManager.getConnection(url, user, passwd);
 Statement stmt = con.createStatement();
 CUBRIDResultSet rs = (CUBRIDResultSet)
 stmt.executeQuery(sql);
 while (rs.next()) {
 CUBRIDOID oid = rs.getOID(1);
 oid.addToSet("settest",
Integer.valueOf(10));
 oid.addToSet("multisettest",
Integer.valueOf(20));
 oid.addToSequence("listtest", 1,
Integer.valueOf(30));
 oid.addToSequence("listtest", 100,

```

```

Integer.valueOf(100));
Integer.valueOf(99));
Integer.valueOf(1));
Integer.valueOf(2));
99);
1);
 }
 con.commit();
 rs.close();
 stmt.close();
 con.close();
} catch (SQLException e) {
 e.printStackTrace();
}
}
}

```

## 자동 증가 특성의 칼럼 값 검색

자동 증가 특성(**AUTO\_INCREMENT**)은 자동으로 각 행의 숫자 값을 증가 생성하는 칼럼에 대한 특성으로서, 보다 자세한 사항은 **칼럼 정의** 절을 참고한다. 수치형 도메인(**SMALLINT**, **INTEGER**, **DECIMAL** ( $p, 0$ ), **NUMERIC** ( $p, 0$ ))에 대해서만 정의할 수 있다.

자동 증가 특성은 JDBC 프로그램에서 자동 생성된 키로 인식되고, 이 키의 검색을 사용하려면 자동 생성된 키 값을 검색할 행을 삽입할 시기를 표시해야 한다. 이를 수행하기 위하여 **Connection.prepareStatement** 와 **Statement.execute** 메서드를 호출하여 플래그를 설정해야 한다. 이때, 실행된 명령문은 **INSERT** 문 또는 **INSERT** within **SELECT** 문이어야 하며, 다른 명령문의 경우 JDBC 드라이버가 플래그를 설정하는 매개변수를 무시한다.

## 수행 단계

- 다음 방법 중 하나를 사용하여 자동 생성된 키를 반환하려는지 표시한다. 자동 증가 특성 칼럼을 지원하는 데이터베이스 서버의 테이블에 대해 다음의 양식을 사용하며, 각 양식은 단일 행 **INSERT** 문에 대해서만 적용 가능하다.

- 아래와 같이 **PreparedStatement** 객체를 작성한다.

```

Connection.prepareStatement(sql statement,
Statement.RETURN_GENERATED_KEYS);

```

- **Statement.execute** 메서드를 사용하여 행을 삽입할 경우, 아래와 같이 사용한다.

```
Statement.execute(sql statement,
Statement.RETURN_GENERATED_KEYS);
```

- **PreparedStatement.getGeneratedKeys** 메서드 또는 **Statement.getGeneratedKeys** 메서드를 호출하여 자동 생성된 키 값이 포함된 **ResultSet** 객체를 검색한다. **ResultSet** 에서 자동 생성된 키의 데이터 유형은 해당 도메인의 데이터 유형에 상관 없이 **DECIMAL** 이다.

## 예제

다음 예제는 자동 증가 특성이 있는 테이블을 생성하고, 데이터를 테이블에 입력하여, 자동 증가 특성 칼럼에 자동 생성된 키 값이 입력되고 해당 키값이 **Statement.getGeneratedKeys ()** 메서드를 통해 정상적으로 검색되는지를 점검하는 예제이다. 앞서 설명한 단계에 해당하는 명령문의 코멘트에 각 단계를 표시하였다.

```
import java.sql.*;
import java.math.*;
import cubrid.jdbc.driver.*;

Connection con;
Statement stmt;
ResultSet rs;
java.math.BigDecimal idColVar;
...
stmt = con.createStatement(); // Create a Statement object

// Create table with identity column
stmt.executeUpdate(
 "CREATE TABLE EMP_PHONE (EMPNO CHAR(6), PHONENO CHAR(4), " +
 "IDENTCOL INTEGER AUTO_INCREMENT)");

stmt.execute(
 "INSERT INTO EMP_PHONE (EMPNO, PHONENO) " +
 "VALUES ('000010', '5555')", // Insert a row
 Statement.RETURN_GENERATED_KEYS); // Indicate you want
// automatically

rs = stmt.getGeneratedKeys(); // generated keys

// Retrieve the automatically <Step 2>
// generated key value in a ResultSet.
// Only one row is returned.
// Create ResultSet for query
while (rs.next()) {
 java.math.BigDecimal idColVar = rs.getBigDecimal(1);
 // Get automatically generated key value
 System.out.println("automatically generated key value = " +
 idColVar);
```

```

}

rs.close(); // Close ResultSet
stmt.close(); // Close Statement

```

## BLOB/CLOB 사용

JDBC에서 **LOB** 데이터를 처리하는 인터페이스는 JDBC 4.0 스펙을 기반으로 구현되었으며, 다음과 같은 제약 사항을 가진다.

- **BLOB**, **CLOB** 객체를 생성할 때에는 순차 쓰기만을 지원한다. 임의 위치에 대한 쓰기는 지원하지 않는다.
- `ResultSet`에서 얻어온 **BLOB**, **CLOB** 객체의 메서드를 호출하여 **BLOB**, **CLOB** 데이터를 변경할 수 없다.
- `Blob.truncate`, `Clob.truncate`, `Blob.position`, `Clob.position` 메서드는 지원하지 않는다.
- **BLOB** / **CLOB** 타입 컬럼에 대해 `PreparedStatement.setAsciiStream`, `PreparedStatement.setBinaryStream`, `PreparedStatement.setCharacterStream` 메서드를 호출하여 **LOB** 데이터를 바인딩할 수 없다.
- JDBC 4.0을 지원하지 않는 환경(예: JDK 1.5 이하)에서 **BLOB** / **CLOB** 타입을 사용하기 위해서는 `conn` 객체를 **CUBRIDConnection** 로 명시적 타입 변환하여 사용하여야 한다. 아래 예제를 참고한다.

```

//JDK 1.6 이상

import java.sql.*;

Connection conn = DriverManager.getConnection(url, id, passwd);
Blob blob = conn.createBlob();

//JDK 1.6 미만

import java.sql.*;
import cubrid.jdbc.driver.*;

Connection conn = DriverManager.getConnection(url, id, passwd);
Blob blob = ((CUBRIDConnection)conn).createBlob();

```

## LOB 데이터 저장

**LOB** 타입 데이터를 바인딩하는 방법은 다음과 같다. 예제를 참고한다.

- java.sql.Blob 또는 java.sql.Clob 객체를 생성하고 그 객체에 파일 내용을 저장한 다음, PreparedStatement의 **setBlob()** 혹은 **setClob()**을 사용한다. (예제 1)
- 질의를 한 다음, 그 ResultSet 객체에서 java.sql.Blob 혹은 java.sql.Clob 객체를 얻고, 그 객체를 PreparedStatement에서 바인딩한다. (예제 2)

#### 예제 1

```
Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
Connection conn = DriverManager.getConnection
("jdbc:cubrid:localhost:33000:image_db:user1:password1:", "", "");

PreparedStatement pstmt1 = conn.prepareStatement("INSERT INTO
doc(image_id, doc_id, image) VALUES (?, ?, ?)");
pstmt1.setString(1, "image-21");
pstmt1.setString(2, "doc-21");

//Creating an empty file in the file system
Blob bImage = conn.createBlob();
byte[] bArray = new byte[256];
...

//Inserting data into the external file. Position is start with 1.
bImage.setBytes(1, bArray);
//Appending data into the external file
bImage.setBytes(257, bArray);
...

pstmt1.setBlob(3, bImage);
pstmt1.executeUpdate();
...
```

#### 예제 2

```
Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
Connection conn = DriverManager.getConnection
("jdbc:cubrid:localhost:33000:image_db:user1:password1:", "", "");
conn.setAutoCommit(false);

PreparedStatement pstmt1 = conn.prepareStatement("SELECT image FROM
doc WHERE image_id = ? ");
pstmt1.setString(1, "image-21");
ResultSet rs = pstmt1.executeQuery();

while (rs.next())
{
 Blob bImage = rs.getBlob(1);
}
```

```

 PreparedStatement pstmt2 = conn.prepareStatement("INSERT INTO
doc(image_id, doc_id, image) VALUES (?, ?, ?)");
 pstmt2.setString(1, "image-22")
 pstmt2.setString(2, "doc-22")
 pstmt2.setBlob(3, bImage);
 pstmt2.executeUpdate();
 pstmt2.close();
 }

 pstmt1.close();
 conn.commit();
 conn.setAutoCommit(true);
 conn.close();
 ...

```

## LOB 데이터 조회

LOB 타입 데이터를 조회하는 방법은 다음과 같다.

- ResultSet에서 **getBytes ()** 혹은 **getString ()** 메서드를 사용하여 데이터를 바로 인출한다. (예제 1)
- ResultSet에서 **getBlob ()** 혹은 **getClob ()** 메서드를 호출하여 java.sql.Blob 혹은 java.sql.Clob 객체를 얻은 다음, 이 객체에 대해 **getBytes ()** 혹은 **getSubString ()** 메서드를 사용하여 데이터를 인출한다. (예제 2)

### 예제 1

```

Connection conn = DriverManager.getConnection
("jdbc:cubrid:localhost:33000:image_db:user1:password1:", "", "");

// ResultSet에서 직접 데이터 인출
PreparedStatement pstmt1 = conn.prepareStatement("SELECT content FROM
doc_t WHERE doc_id = ? ");
pstmt1.setString(1, "doc-10");
ResultSet rs = pstmt1.executeQuery();

while (rs.next())
{
 String sContent = rs.getString(1);
 System.out.println("doc.content= "+sContent.);
}

```

### 예제 2

```

Connection conn = DriverManager.getConnection
("jdbc:cubrid:localhost:33000:image_db:user1:password1:", "", "");

//ResultSet에서 Blob 객체를 얻고 Blob 객체로부터 데이터 인출
PreparedStatement pstmt2 = conn.prepareStatement("SELECT image FROM

```

```
image_t WHERE image_id = ?");
pstmt2.setString(1,"image-20");
ResultSet rs = pstmt2.executeQuery();

while (rs.next())
{
 Blob bImage = rs.getBlob(1);
 Bytes[] bArray = bImage.getBytes(1, (int)bImage.length());
}
```

#### 참고

칼럼에서 정의한 크기보다 큰 문자열을 **INSERT** / **UPDATE** 하면 문자열이 잘려서 입력된다.

setBoolean

**prepareStatement.setBoolean(1, true)** 는 다음으로 지정된다.

- numeric 타입에서의 1.
- string 타입에서의 '1'.

**prepareStatement.setBooelan(1, false)** 는 다음으로 지정된다.

- numeric 타입에서 0.
- string 타입에서 '0'.

#### 참고

이전 버전에서 동작 방식

**prepareStatement.setBoolean(1, true)** 은 다음으로 지정된다.

- 2008 R4.1, 9.0 에서는 BIT(1) 타입의 1 을 의미한다.
- 2008 R4.3, 2008 R4.4, 9.1, 9.2, 9.3 에서는 SHORT 타입의 -128 을 의미한다.

JDBC 에러 코드와 에러 메시지

SQLException에서 발생하는 JDBC 에러 코드는 다음과 같다.

- 모든 에러 번호는 0보다 작은 음수이다.

- SQLException 발생 시 에러 번호는 SQLException.getErrorCode(), 에러 메시지는 SQLException.getMessage()를 통해 확인할 수 있다.
- 에러 번호가 -21001부터 -21999 사이이면, CUBRID JDBC 메서드에서 발생하는 에러이다.
- 에러 번호가 -10000부터 -10999 사이이면, CAS에서 발생하는 에러를 JDBC가 전달받아 반환하는 에러이다. CAS 에러는 **CAS 에러**를 참고한다.
- 에러 번호가 0부터 -9999 사이이면, DB 서버에서 발생하는 에러이다. DB 서버 에러는 **데이터베이스 서버 에러**를 참고한다.

| 에러 번호  | 에러 메시지                                                              |
|--------|---------------------------------------------------------------------|
| -21001 | Index's Column is Not Object                                        |
| -21002 | Server error                                                        |
| -21003 | Cannot communicate with the broker                                  |
| -21004 | Invalid cursor position                                             |
| -21005 | Type conversion error                                               |
| -21006 | Missing or invalid position of the bind variable provided           |
| -21007 | Attempt to execute the query when not all the parameters are binded |
| -21008 | Internal Error: NULL value                                          |
| -21009 | Column index is out of range                                        |
| -21010 | Data is truncated because receive buffer is too small               |



| 에러 번호  | 에러 메시지                                                        |
|--------|---------------------------------------------------------------|
| -21011 | Internal error: Illegal schema type                           |
| -21012 | File access failed                                            |
| -21013 | Cannot connect to a broker                                    |
| -21014 | Unknown transaction isolation level                           |
| -21015 | Internal error: The requested information is not available    |
| -21016 | The argument is invalid                                       |
| -21017 | Connection or Statement might be closed                       |
| -21018 | Internal error: Invalid argument                              |
| -21019 | Cannot communicate with the broker or received invalid packet |
| -21020 | No More Result                                                |
| -21021 | This ResultSet do not include the OID                         |
| -21022 | Command is not insert                                         |
| -21023 | Error                                                         |
| -21024 | Request timed out                                             |

| 에러 번호  | 에러 메시지                                                                              |
|--------|-------------------------------------------------------------------------------------|
| -21101 | Attempt to operate on a closed Connection.                                          |
| -21102 | Attempt to access a closed Statement.                                               |
| -21103 | Attempt to access a closed PreparedStatement.                                       |
| -21104 | Attempt to access a closed ResultSet.                                               |
| -21105 | Not supported method                                                                |
| -21106 | Unknown transaction isolation level.                                                |
| -21107 | invalid URL -                                                                       |
| -21108 | The database name should be given.                                                  |
| -21109 | The query is not applicable to the executeQuery(). Use the executeUpdate() instead. |
| -21110 | The query is not applicable to the executeUpdate(). Use the executeQuery() instead. |
| -21111 | The length of the stream cannot be negative.                                        |
| -21112 | An IOException was caught during reading the inputStream.                           |
| -21113 |                                                                                     |

| 에러 번호  | 에러 메시지                                                                         |
|--------|--------------------------------------------------------------------------------|
|        | Not supported method, because it is deprecated.                                |
| -21114 | The object does not seem to be a number.                                       |
| -21115 | Missing or invalid position of the bind variable provided.                     |
| -21116 | The column name is invalid.                                                    |
| -21117 | Invalid cursor position.                                                       |
| -21118 | Type conversion error.                                                         |
| -21119 | Internal error: The number of attributes is different from the expected.       |
| -21120 | The argument is invalid.                                                       |
| -21121 | The type of the column should be a collection type.                            |
| -21122 | Attempt to operate on a closed DatabaseMetaData.                               |
| -21123 | Attempt to call a method related to scrollability of non-scrollable ResultSet. |
| -21124 | Attempt to call a method related to sensitivity of non-sensitive ResultSet.    |
| -21125 |                                                                                |

| 에러 번호  | 에러 메시지                                                                            |
|--------|-----------------------------------------------------------------------------------|
|        | Attempt to call a method related to updatability of non-updatable ResultSet.      |
| -21126 | Attempt to update a column which cannot be updated.                               |
| -21127 | The query is not applicable to the executeInsert().                               |
| -21128 | The argument row can not be zero.                                                 |
| -21129 | Given InputStream object has no data.                                             |
| -21130 | Given Reader object has no data.                                                  |
| -21131 | Insertion query failed.                                                           |
| -21132 | Attempt to call a method related to scrollability of TYPE_FORWARD_ONLY Statement. |
| -21133 | Authentication failure                                                            |
| -21134 | Attempt to operate on a closed PooledConnection.                                  |
| -21135 | Attempt to operate on a closed XAConnection.                                      |
| -21136 | Illegal operation in a distributed transaction                                    |
| -21137 |                                                                                   |

| 에러 번호  | 에러 메시지                                                                            |
|--------|-----------------------------------------------------------------------------------|
|        | Attempt to access a CUBRIDOID associated with a Connection which has been closed. |
| -21138 | The table name is invalid.                                                        |
| -21139 | Lob position to write is invalid.                                                 |
| -21140 | Lob is not writable.                                                              |
| -21141 | Request timed out.                                                                |

## JDBC 예제 프로그램

다음은 JDBC 드라이버를 통해 CUBRID에 접속하여 데이터를 조회, 삽입하는 것을 간단하게 구성한 예제이다. 예제를 실행하려면 먼저 접속하고자 하는 데이터베이스와 CUBRID 브로커가 구동되어 있어야 한다. 예제에서는 설치 시 자동으로 생성되는 demodb 데이터베이스를 사용한다.

### JDBC 드라이버 로드

CUBRID에 접속하기 위해서는 **Class**의 **forName()** 메서드를 사용하여 JDBC 드라이버를 로드해야 한다. 자세한 내용은 [JDBC 개요](#)를 참고한다.

```
Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
```

### 데이터베이스 연결

JDBC 드라이버를 로드한 후 **DriverManager**의 **getConnection()** 메서드를 사용하여 데이터베이스와 연결한다. **Connection** 객체를 생성하기 위해서는 데이터베이스의 위치를 기술하기 위한 URL, 데이터베이스의 사용자 이름, 암호 등의 정보가 지정되어야 한다. 자세한 내용은 [연결 설정](#)을 참고한다.

```
String url = "jdbc:cubrid:localhost:33000:demodb::";
String userid = "dba";
String password = "";

Connection conn = DriverManager.getConnection(url,userid,password);
```

DataSource 객체를 사용하여 데이터베이스에 연결할 수도 있다. 연결 URL 문자열에 연결 속성 (connection property)을 포함하고자 하는 경우, CUBRIDDataSource에 구현된 setURL 메서드를 사용할 수 있다.

```
import cubrid.jdbc.driver.CUBRIDDataSource;
...

ds = new CUBRIDDataSource();
ds.setURL("jdbc:cubrid:127.0.0.1:33000:demodb::?
charset=utf8&logSlowQueries=true&slowQueryThresholdMillis=1000");
```

CUBRIDDataSource에 대한 자세한 설명은 [DataSource 객체로 연결](#)을 참고한다.

### DataSource 객체로 연결

다음은 CUBRID에 구현된 DataSource인 CUBRIDDataSource의 setURL을 이용하여 DB에 접속하고, 여러 개의 스레드에서 SELECT 문을 실행하는 예제이다. 소스는 DataSourceMT.java와 DataSourceExample.java의 두 개로 나뉘어져 있다.

- DataSourceMT.java는 main 함수를 포함하고 있다. CUBRIDDataSource 객체를 생성하고 setURL 메서드를 호출하여 DB에 접속한 후, 여러 개의 스레드가 DataSourceExample.test 메서드를 수행한다.
- DataSourceExample.java에는 DataSourceMT.java에서 생성된 스레드가 수행할 DataSourceExample.test 메서드가 구현되어 있다.

*DataSourceMT.java*

```
import cubrid.jdbc.driver.*;

public class DataSourceMT {
 static int num_thread = 20;

 public static void main(String[] args) {
 CUBRIDDataSource ds = null;
 thrCPDSMT thread[];

 ds = new CUBRIDDataSource();
 ds.setURL("jdbc:cubrid:127.0.0.1:33000:demodb::?
charset=utf8&logSlowQueries=true&slowQueryThresholdMillis=1000");

 try {
 thread = new thrCPDSMT[num_thread];

 for (int i = 0; i < num_thread; i++) {
 Thread.sleep(1);
 thread[i] = new thrCPDSMT(i, ds);
 try {
```

```

 Thread.sleep(1);
 thread[i].start();
 } catch (Exception e) {
 }
}

for (int i = 0; i < num_thread; i++) {
 thread[i].join();
 System.err.println("join thread : " + i);
}

} catch (Exception e) {
 e.printStackTrace();
 System.exit(-1);
}
}
}

class thrCPDSMT extends Thread {
 CUBRIDDataSource thread_ds;
 int thread_id;

 thrCPDSMT(int tid, CUBRIDDataSource ds) {
 thread_id = tid;
 thread_ds = ds;
 }

 public void run() {
 try {
 DataSourceExample.test(thread_ds);
 } catch (Exception e) {
 e.printStackTrace();
 System.exit(-1);
 }
 }
}
}

```

DataSourceExample.java

```

import java.sql.*;
import javax.sql.*;
import cubrid.jdbc.driver.*;

public class DataSourceExample {

 public static void printdata(ResultSet rs) throws SQLException {
 try {
 ResultSetMetaData rsmd = null;

```

```

 rsmd = rs.getMetaData();
 int numberOfColumn = rsmd.getColumnCount();

 while (rs.next()) {
 for (int j = 1; j <= numberOfColumn; j++)
 System.out.print(rs.getString(j) + " ");
 System.out.println("");
 }
 } catch (SQLException e) {
 System.out.println("SQLException : " + e.getMessage());
 throw e;
 }
}

public static void test(CUBRIDDataSource ds) throws Exception {
 Connection connection = null;
 Statement statement = null;
 ResultSet resultSet = null;

 for (int i = 1; i <= 20; i++) {
 try {
 connection = ds.getConnection("dba", "");
 statement = connection.createStatement();
 String SQL = "SELECT * FROM code";
 resultSet = statement.executeQuery(SQL);

 while (resultSet.next()) {
 printdata(resultSet);
 }

 if (i % 5 == 0) {
 System.gc();
 }
 } catch (Exception e) {
 e.printStackTrace();
 } finally {
 closeAll(resultSet, statement, connection);
 }
 }
}

public static void closeAll(ResultSet resultSet, Statement
statement,
 Connection connection) {
 if (resultSet != null) {
 try {
 resultSet.close();
 } catch (SQLException e) {
 }
 }
 if (statement != null) {
 try {

```



```

 statement.close();
 } catch (SQLException e) {
 }
}
if (connection != null) {
 try {
 connection.close();
 } catch (SQLException e) {
 }
}
}
}

```

### 데이터베이스 조작(질의 수행 및 ResultSet 처리)

접속된 데이터베이스에 질의문을 전달하고 실행시키기 위하여 **Statement** , **PreparedStatement** , **CallableStatement** 객체를 생성한다. **Statement** 객체가 생성되면, **Statement** 객체의 **executeQuery ()** 메서드나 **executeUpdate ()** 메서드를 사용하여 질의문을 실행한다. **next ()** 메서드를 사용하여 **executeQuery ()** 메서드의 결과로 반환된 **ResultSet** 의 다음 행을 처리할 수 있다.

#### 참고

2008 R4.x 이하 버전에서 질의 수행 후 커밋을 수행하면 ResultSet을 자동으로 닫으므로, 커밋 이후에는 ResultSet을 사용하지 않아야 한다. CUBRID는 기본적으로 자동 커밋 모드로 수행되므로, 이를 원하지 않으면 반드시 **conn.setAutoCommit(false);** 를 코드에 명시해야 한다.

9.1 이상 버전부터는 **커서 유지(cursor holdability)**가 지원되므로 커밋 이후에도 **ResultSet**을 사용할 수 있다.

### 데이터베이스 연결 해제

각 객체에 대해 **close ()** 메서드를 수행하여 데이터베이스와의 연결을 해제할 수 있다.

### CREATE, INSERT

다음은 *demodb*에 접속하여 테이블을 생성하고, prepared statement로 질의문을 수행한 후 질의를 롤백시키는 예제 코드이다.

```

import java.util.*;
import java.sql.*;

public class Basic {
 public static Connection connect() {
 Connection conn = null;
 try {

```

```

 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 conn = DriverManager.getConnection("jdbc:cubrid:localhost:
33000:demodb::", "dba", "");
 conn.setAutoCommit (false) ;
 } catch (Exception e) {
 System.err.println("SQLException : " + e.getMessage());
 }
 return conn;
}

public static void printdata(ResultSet rs) {
 try {
 ResultSetMetaData rsmd = null;

 rsmd = rs.getMetaData();
 int numberOfColumn = rsmd.getColumnCount();

 while (rs.next ()) {
 for(int j=1; j<=numberOfColumn; j++)
 System.out.print(rs.getString(j) + " ");
 System.out.println("");
 }
 } catch (Exception e) {
 System.err.println("SQLException : " + e.getMessage());
 }
}

public static void main(String[] args) throws Exception {
 Connection conn = null;
 Statement stmt = null;
 ResultSet rs = null;
 PreparedStatement preStmt = null;

 try {
 conn = connect();

 stmt = conn.createStatement();
 stmt.executeUpdate("CREATE TABLE xoo (a INT, b INT, c
CHAR(10))");

 preStmt = conn.prepareStatement("INSERT INTO xoo
VALUES(?,?, '100')");
 preStmt.setInt (1, 1) ;
 preStmt.setInt (2, 1*10) ;
 int rst = preStmt.executeUpdate () ;

 rs = stmt.executeQuery("select a,b,c from xoo");

 printdata(rs);

 conn.rollback();
 stmt.close();
 }
}

```

```

 conn.close();
 } catch (Exception e) {
 conn.rollback();
 System.err.println("SQLException : " + e.getMessage());
 } finally {
 if (conn != null) conn.close();
 }
}
}

```

## SELECT

다음은 CUBRID 설치 시 기본 제공되는 *demodb* 에 접속하여 **SELECT** 질의를 수행하는 예제이다.

```

import java.sql.*;

public class SelectData {
 public static void main(String[] args) throws Exception {
 Connection conn = null;
 Statement stmt = null;
 ResultSet rs = null;

 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 conn =
DriverManager.getConnection("jdbc:cubrid:localhost:
33000:demodb::", "dba", "");

 String sql = "SELECT name, players FROM event";
 stmt = conn.createStatement();
 rs = stmt.executeQuery(sql);

 while(rs.next()) {
 String name = rs.getString("name");
 String players = rs.getString("players");
 System.out.println("name ==> " + name);
 System.out.println("Number of players==> " + players);

 System.out.println("\n=====");
 }

 rs.close();
 stmt.close();
 conn.close();
 } catch (SQLException e) {
 System.err.println(e.getMessage());
 } catch (Exception e) {
 System.err.println(e.getMessage());
 } finally {
 if (conn != null) conn.close();
 }
 }
}

```

```

 }
}

```

## INSERT

다음은 CUBRID 설치 시 기본 제공되는 *demodb*에 접속하여 **INSERT** 질의를 수행하는 예제이다. 데이터 삭제 및 갱신 방법은 데이터 삽입 방법과 동일하므로 아래 코드에서 질의문만 변경하여 사용할 수 있다.

```

import java.sql.*;

public class insertData {
 public static void main(String[] args) throws Exception {
 Connection conn = null;
 Statement stmt = null;

 try {
 Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
 conn = DriverManager.getConnection("jdbc:cubrid:localhost:
33000:demodb::", "dba", "");
 String sql = "insert into olympic(host_year, host_nation,
host_city, opening_date, closing_date) values (2008, 'China',
'Beijing', to_date('08-08-2008', 'mm-dd-yyyy'),
to_date('08-24-2008', 'mm-dd-yyyy'))";
 stmt = conn.createStatement();
 stmt.executeUpdate(sql);
 System.out.println("데이터가 입력되었습니다.");
 stmt.close();
 } catch (SQLException e) {
 System.err.println(e.getMessage());
 } catch (Exception e) {
 System.err.println(e.getMessage());
 } finally {
 if (conn != null) conn.close();
 }
 }
}

```

## JDBC API

JDBC API에 대한 자세한 내용은 Java API Specification 문서(<https://docs.oracle.com/javase/7/docs/api/>)를 참고한다. 기타 Java에 대한 자세한 내용은 Java SE Documentation 문서(<https://www.oracle.com/java/technologies/javase-documentation.html>)를 참고한다.

**커서 유지(cursor holdability)**와 관련하여 설정을 명시하지 않으면 기본으로 커서가 유지된다.

다음은 CUBRID에서 지원하는 JDBC 표준 인터페이스를 및 확장 인터페이스를 정리한 목록이다. JDBC 2.0 스펙에 포함된 메서드 중 일부는 지원하지 않으므로 프로그램 작성 시 주의한다.

### JDBC 인터페이스 지원 여부

| JDBC 표준 인터페이스              | CUBRID 확장 인터페이스                  | 지원 여부                                     |
|----------------------------|----------------------------------|-------------------------------------------|
| java.sql.Blob              |                                  | 지원                                        |
| java.sql.CallableStatement |                                  | 지원                                        |
| java.sql.Clob              |                                  | 지원                                        |
| java.sql.Connection        |                                  | 지원                                        |
| java.sql.DatabaseMetaData  |                                  | 지원                                        |
| java.sql.Driver            |                                  | 지원                                        |
| java.sql.PreparedStatement | java.sql.CUBRIDPreparedStatement | 지원                                        |
| java.sql.ResultSet         | java.sql.CUBRIDResultSet         | 지원                                        |
| java.sql.ResultSetMetaData | java.sql.CUBRIDResultSetMetaData | 지원                                        |
| N/A                        | CUBRIDOID                        | 지원                                        |
| java.sql.Statement         | java.sql.CUBRIDStatement         | JDBC 3.0의<br>getGeneratedKeys() 메서드<br>지원 |
| java.sql.DriverManager     |                                  | 지원                                        |

| JDBC 표준 인터페이스              | CUBRID 확장 인터페이스          | 지원 여부 |
|----------------------------|--------------------------|-------|
| Java.sql.SQLException      | Java.sql.CUBRIDException | 지원    |
| java.sql.Array             |                          | 미지원   |
| java.sql.ParameterMetaData |                          | 미지원   |
| java.sql.Ref               |                          | 미지원   |
| java.sql.Savepoint         |                          | 미지원   |
| java.sql.SQLData           |                          | 미지원   |
| java.sql.SQLInput          |                          | 미지원   |
| java.sql.Struct            |                          | 미지원   |

### 참고

- 커서 유지(cursor holdability)와 관련하여 설정을 명시하지 않으면 기본으로 커서가 유지된다.
- 2008 R4.3부터 자동 커밋이 ON일 때 질의문을 일괄 처리하는 메서드의 동작 방식이 변경되었음에 주의한다. 질의문을 일괄 처리하는 메서드는 PreparedStatement.executeBatch와 Statement.executeBatch이다. 이들은 2008 R4.1 버전까지 자동 커밋 모드에서 배열 내의 모든 질의를 수행한 후에 커밋했으나, 2008 R4.3버전부터는 각 질의를 수행할 때마다 커밋하도록 변경되었다.
- 자동 커밋이 OFF일 때 질의문을 일괄 처리하는 메서드에서 배열 내의 질의 수행 중 일부에서 일반적인 오류가 발생하는 경우, 이를 건너뛰고 다음 질의를 계속 수행한다. 그러나, 교착 상태가 발생하면 트랜잭션을 롤백하고 오류 처리한다.

## CCI 드라이버

### CCI 개요

CCI(C Client Interface)는 CUBRID 브로커와 응용 클라이언트 사이에 위치하여, C 기반의 응용 클라이언트가 브로커를 통해 CUBRID 데이터베이스 서버로 접근할 수 있는 인터페이스로서, 브로커 응용 서버(CAS)를 이용하는 도구(예: PHP, ODBC)를 만들기 위한 하부 구조로도 사용된다. 여기서, CUBRID 브로커는 응용 클라이언트로부터 받은 질의를 데이터베이스에 전달하고, 실행 결과를 응용 클라이언트로 전송하는 역할을 수행한다.

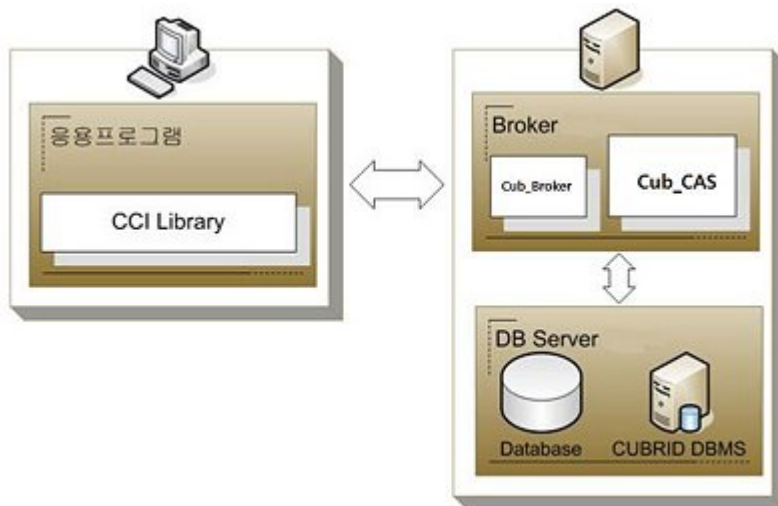
CCI를 사용하기 위해서는 헤더 파일과 라이브러리 파일이 필요하다.

|          | Windows               | Unix/Linux            |
|----------|-----------------------|-----------------------|
| C 헤더 파일  | cci/include/cas_cci.h | cci/include/cas_cci.h |
| 정적 라이브러리 | cci/lib/cascci.lib    | cci/lib/libcascci.a   |
| 동적 라이브러리 | cci/bin/cascci.dll    | cci/lib/libcascci.so  |

#### 참고

- Windows의 경우 CCI 드라이버를 사용하려면 Microsoft Visual C ++ 2015 재배포 가능 패키지 (x86 또는 x64)를 설치해야 한다.
- 11.2 버전부터는 CCI 드라이버에 대한 경로가 변경되었다. (cci/inculde, cci/lib, cci/bin)

CCI 드라이버는 CUBRID에서 제공되는 C 언어 인터페이스로, CUBRID 설치 패키지에 포함되어 있다. CCI는 브로커를 통해서 접속하므로 다른 인터페이스인 JDBC, PHP, ODBC, Python, Ruby 등과 동일하게 관리될 수 있다. 실제로 PHP, ODBC, Python, Ruby 인터페이스는 CCI를 기반으로 개발되었다. 단, JDBC는 CCI를 기반으로 개발되지 않았다.



## CCI 프로그래밍

### CCI 응용 프로그램 작성

CCI를 이용하는 응용 프로그램은 기본적으로 CAS와 연결하기, 질의 준비, 질의 수행, 응답 처리, 연결 끊기의 과정을 통해 CUBRID를 이용한다. 각 과정에서 CCI는 연결 핸들(connection handle), 질의 핸들(query handle), 응답 핸들(response handle)을 통해 응용 프로그램과 소통한다.

브로커 파라미터인 `CCI_DEFAULT_AUTOCOMMIT`으로 응용 프로그램 시작 시 자동 커밋 모드 of 기본값을 설정할 수 있으며, 브로커 파라미터 설정을 생략하면 기본값은 **ON** 이다. 응용 프로그램 내에서 자동 커밋 모드를 변경하려면 `cci_set_autocommit()` 함수를 이용하며, 자동 커밋 모드가 **OFF** 이면 `cci_end_tran()` 함수를 이용하여 명시적으로 트랜잭션을 커밋하거나 롤백해야 한다.

기본적인 작성 순서는 다음과 같으며, prepared statement 사용을 위해서는 변수에 데이터를 바인딩하는 작업이 추가된다. 이를 예제 1 및 예제 2에 구현하였다.

- 데이터베이스 연결 핸들 열기(관련 함수: `cci_connect()` , `cci_connect_with_url()` )
- prepared statement를 위한 요청 핸들 얻기 (관련 함수: `cci_prepare()` )
- prepared statement에 데이터 바인딩하기(관련 함수: `cci_bind_param()` )
- prepared statement 실행하기(관련 함수: `cci_execute()` )
- 실행 결과 처리하기(관련 함수: `cci_cursor()` , `cci_fetch()` , `cci_get_data()` , `cci_get_result_info()` )
- 요청 핸들 닫기(관련 함수: `cci_close_req_handle()` )
- 데이터베이스 연결 핸들 닫기(관련 함수: `cci_disconnect()` )
- 데이터베이스 연결 풀 사용하기(관련 함수: `cci_property_create()` , `cci_property_destroy()` , `cci_property_set()` , `cci_datasource_create()` , `cci_datasource_destroy()` , `cci_datasource_borrow()` , `cci_datasource_release()` , `cci_datasource_change_property()` )



## 참고

- Windows에서 CCI 응용 프로그램을 컴파일하려면 “WINDOWS”가 define되어야 하므로 “-DWINDOWS” 옵션을 컴파일러에 반드시 포함하도록 한다.
- 스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.
- 자동 커밋 모드에서 SELECT 문 수행 이후 모든 결과 셋이 fetch되지 않으면 커밋이 되지 않는다. 따라서, 자동 커밋 모드라 하더라도 프로그램 내에서 결과 셋에 대한 fetch 도중 어떠한 오류가 발생한다면 반드시 `cci_end_tran()` 을 호출하여 트랜잭션을 종료 처리하도록 한다.

## 예제 1

```
// Example to execute a simple query
// In Linux: gcc -o simple simple.c -m64 -I${CUBRID}/cci/include -
lnsl ${CUBRID}/cci/lib/libcascci.so -lpthread

#include <stdio.h>
#include "cas_cci.h"
#define BUFSIZE (1024)

int
main (void)
{
 int con = 0, req = 0, col_count = 0, i, ind;
 int error;
 char *data;
 T_CCI_ERROR cci_error;
 T_CCI_COL_INFO *col_info;
 T_CCI_CUBRID_STMT stmt_type;
 char *query = "select * from code";

 //getting a connection handle for a connection with a server
 con = cci_connect ("localhost", 33000, "demodb", "dba", "");
 if (con < 0)
 {
 printf ("cannot connect to database\n");
 return 1;
 }

 //preparing the SQL statement
 req = cci_prepare (con, query, 0, &cci_error);
 if (req < 0)
 {
 printf ("prepare error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }
}
```

```

 }

 //getting column information when the prepared statement is the
 SELECT query
 col_info = cci_get_result_info (req, &stmt_type, &col_count);
 if (col_info == NULL)
 {
 printf ("get_result_info error: %d, %s\n",
 cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }

 //Executing the prepared SQL statement
 error = cci_execute (req, 0, 0, &cci_error);
 if (error < 0)
 {
 printf ("execute error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }
 while (1)
 {

 //Moving the cursor to access a specific tuple of results
 error = cci_cursor (req, 1, CCI_CURSOR_CURRENT, &cci_error);
 if (error == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 if (error < 0)
 {
 printf ("cursor error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }

 //Fetching the query result into a client buffer
 error = cci_fetch (req, &cci_error);
 if (error < 0)
 {
 printf ("fetch error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }
 for (i = 1; i <= col_count; i++)
 {

 //Getting data from the fetched result
 error = cci_get_data (req, i, CCI_A_TYPE_STR, &data,
&ind);
 if (error < 0)

```

```

 {
 printf ("get_data error: %d, %d\n", error, i);
 goto handle_error;
 }
 printf ("%s\t", data);
 }
 printf ("\n");
}

//Closing the request handle
error = cci_close_req_handle (req);
if (error < 0)
{
 printf ("close_req_handle error: %d, %s\n",
cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
}

//Disconnecting with the server
error = cci_disconnect (con, &cci_error);
if (error < 0)
{
 printf ("error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}

return 0;

handle_error:
if (req > 0)
 cci_close_req_handle (req);
if (con > 0)
 cci_disconnect (con, &cci_error);

return 1;
}

```

## 예제 2

```

// Example to execute a query with a bind variable
// In Linux: gcc -o cci_bind cci_bind.c -m64 -I${CUBRID}/cci/include
-lsl ${CUBRID}/cci/lib/libcascci.so -lpthread

#include <stdio.h>
#include <string.h>
#include "cas_cci.h"
#define BUFSIZE (1024)

```

```
int
main (void)
{
 int con = 0, req = 0, col_count = 0, i, ind;
 int error;
 char *data;
 T_CCI_ERROR cci_error;
 T_CCI_COL_INFO *col_info;
 T_CCI_CUBRID_STMT stmt_type;
 char *query = "select * from nation where name = ?";
 char namebuf[128];

 //getting a connection handle for a connection with a server
 con = cci_connect ("localhost", 33000, "demodb", "dba", "");
 if (con < 0)
 {
 printf ("cannot connect to database\n");
 return 1;
 }

 //preparing the SQL statement
 req = cci_prepare (con, query, 0, &cci_error);
 if (req < 0)
 {
 printf ("prepare error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }

 //Binding date into a value
 strcpy (namebuf, "Korea");
 error =
 cci_bind_param (req, 1, CCI_A_TYPE_STR, namebuf,
CCI_U_TYPE_STRING,
 CCI_BIND_PTR);
 if (error < 0)
 {
 printf ("bind_param error: %d ", error);
 goto handle_error;
 }

 //getting column information when the prepared statement is the
 SELECT query
 col_info = cci_get_result_info (req, &stmt_type, &col_count);
 if (col_info == NULL)
 {
 printf ("get_result_info error: %d, %s\n",
cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }
}
```

```

//Executing the prepared SQL statement
error = cci_execute (req, 0, 0, &cci_error);
if (error < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
}

//Executing the prepared SQL statement
error = cci_execute (req, 0, 0, &cci_error);
if (error < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
}

while (1)
{
 //Moving the cursor to access a specific tuple of results
 error = cci_cursor (req, 1, CCI_CURSOR_CURRENT, &cci_error);
 if (error == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 if (error < 0)
 {
 printf ("cursor error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }

 //Fetching the query result into a client buffer
 error = cci_fetch (req, &cci_error);
 if (error < 0)
 {
 printf ("fetch error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }
 for (i = 1; i <= col_count; i++)
 {
 //Getting data from the fetched result
 error = cci_get_data (req, i, CCI_A_TYPE_STR, &data,
&ind);
 if (error < 0)
 {
 printf ("get_data error: %d, %d\n", error, i);
 goto handle_error;

```

```

 }
 if (ind == -1)
 {
 printf ("NULL\t");
 }
 else
 {
 printf ("%s\t", data);
 }
}
printf ("\n");
}

//Closing the request handle
error = cci_close_req_handle (req);
if (error < 0)
{
 printf ("close_req_handle error: %d, %s\n",
cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
}

//Disconnecting with the server
error = cci_disconnect (con, &cci_error);
if (error < 0)
{
 printf ("error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}

return 0;

handle_error:
if (req > 0)
 cci_close_req_handle (req);
if (con > 0)
 cci_disconnect (con, &cci_error);
return 1;
}

```

### 예제 3

```

// Example to use connection/statement pool in CCI
// In Linux: gcc -o cci_pool cci_pool.c -m64 -I${CUBRID}/cci/include
-lnsl ${CUBRID}/cci/lib/libcascci.so -lpthread

#include <stdio.h>
#include "cas_cci.h"

```

```

int main ()
{
 T_CCI_PROPERTIES *ps = NULL;
 T_CCI_DATASOURCE *ds = NULL;
 T_CCI_ERROR err;
 T_CCI_CONN cons;
 int rc = 1, i;

 ps = cci_property_create ();
 if (ps == NULL)
 {
 fprintf (stderr, "Could not create T_CCI_PROPERTIES.\n");
 rc = 0;
 goto cci_pool_end;
 }

 cci_property_set (ps, "user", "dba");
 cci_property_set (ps, "url", "cci:cubrid:localhost:
33000:demodb::");
 cci_property_set (ps, "pool_size", "10");
 cci_property_set (ps, "max_wait", "1200");
 cci_property_set (ps, "pool_prepared_statement", "true");
 cci_property_set (ps, "login_timeout", "300000");
 cci_property_set (ps, "query_timeout", "3000");

 ds = cci_datasource_create (ps, &err);
 if (ds == NULL)
 {
 fprintf (stderr, "Could not create T_CCI_DATASOURCE.\n");
 fprintf (stderr, "E[%d,%s]\n", err.err_code, err.err_msg);
 rc = 0;
 goto cci_pool_end;
 }

 for (i = 0; i < 3; i++)
 {
 cons = cci_datasource_borrow (ds, &err);
 if (cons < 0)
 {
 fprintf (stderr,
 "Could not borrow a connection from the data
source.\n");
 fprintf (stderr, "E[%d,%s]\n", err.err_code,
err.err_msg);
 continue;
 }
 // put working code here.
 cci_work (cons);
 cci_datasource_release (ds, cons, &err);
 }
}

```

```
cci_pool_end:
 cci_property_destroy (ps);
 cci_datasource_destroy (ds);

 return 0;
}

// working code
int cci_work (T_CCI_CONN con)
{
 T_CCI_ERROR err;
 char sql[4096];
 int req, res, error, ind;
 int data;

 cci_set_autocommit (con, CCI_AUTOCOMMIT_TRUE);
 cci_set_lock_timeout (con, 100, &err);
 cci_set_isolation_level (con, TRAN_REP_CLASS_COMMIT_INSTANCE,
&err);

 error = 0;
 snprintf (sql, 4096, "SELECT host_year FROM record WHERE
athlete_code=11744");
 req = cci_prepare (con, sql, 0, &err);
 if (req < 0)
 {
 printf ("prepare error: %d, %s\n", err.err_code,
err.err_msg);
 return error;
 }

 res = cci_execute (req, 0, 0, &err);
 if (res < 0)
 {
 printf ("execute error: %d, %s\n", err.err_code,
err.err_msg);
 goto cci_work_end;
 }

 while (1)
 {
 error = cci_cursor (req, 1, CCI_CURSOR_CURRENT, &err);
 if (error == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 if (error < 0)
 {
 printf ("cursor error: %d, %s\n", err.err_code, err.err_msg);
 goto cci_work_end;
 }
 }
}
```



```

error = cci_fetch (req, &err);
if (error < 0)
{
 printf ("fetch error: %d, %s\n", err.err_code, err.err_msg);
 goto cci_work_end;
}

error = cci_get_data (req, 1, CCI_A_TYPE_INT, &data, &ind);
if (error < 0)
{
 printf ("get data error: %d\n", error);
 goto cci_work_end;
}
printf ("%d\n", data);
}

error = 1;
cci_work_end:
 cci_close_req_handle (req);
 return error;
}

```

## 라이브러리 적용

CCI를 이용한 응용 프로그램을 작성했다면 프로그램 특성에 따라 정적 링크 형태로 프로그램을 수행시킬 것인지, 아니면 동적으로 CCI를 호출하여 사용할 것인지를 결정하여 프로그램을 빌드한다. CCI 개요의 표를 참조하여 사용할 라이브러리를 결정한다.

다음은 유닉스/Linux에서 동적인 라이브러리를 사용하여 링크하는 Makefile의 예제이다.

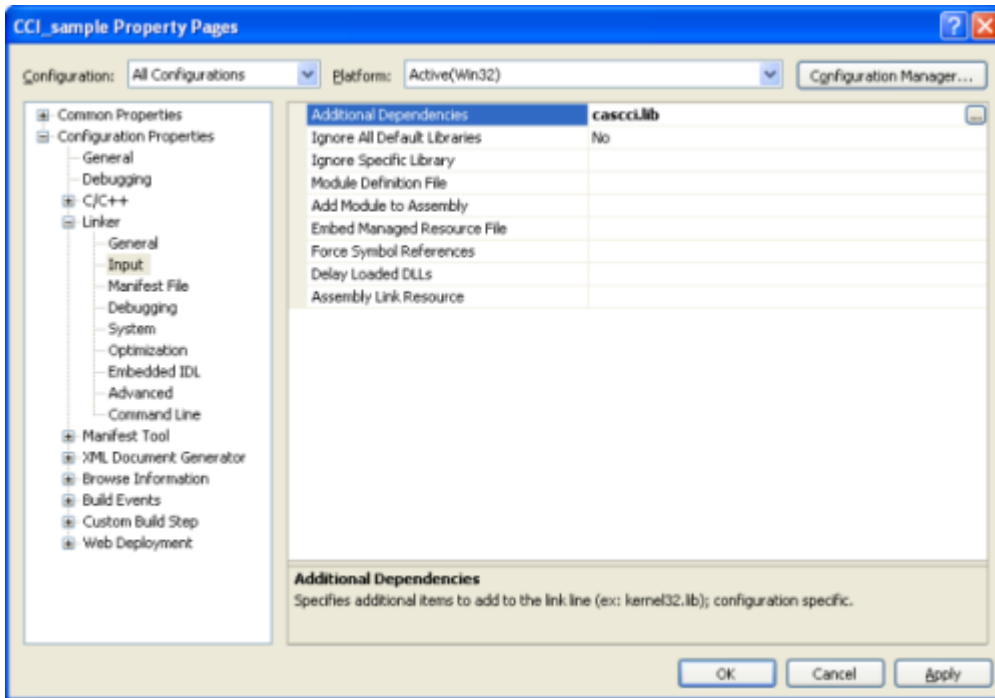
```

CC=gcc
CFLAGS = -g -Wall -I. -I$(CUBRID)/cci/include
LDFLAGS = -L$(CUBRID)/cci/lib -lcascci -lnsl

TEST_OBJS = test.o
EXES = test
all: $(EXES)
test: $(TEST_OBJS)
 $(CC) -o $@ $(TEST_OBJS) $(LDFLAGS)

```

다음은 Windows에서 정적 라이브러리를 적용하기 위한 설정이다.



## BLOB/CLOB 사용

### LOB 데이터 저장

CCI 응용 프로그램에서 다음 함수를 사용하여 **LOB** 데이터 파일을 생성하고 데이터를 바인딩할 수 있다.

- **LOB** 데이터 파일 생성하기 (관련 함수: `cci_blob_new()`, `cci_blob_write()`)
- **LOB** 데이터를 바인딩하기 (관련 함수: `cci_bind_param()`)
- **LOB** 구조체에 대한 메모리 해제하기 (관련 함수: `cci_blob_free()`)

### 예제

```
int con = 0; /* connection handle */
int req = 0; /* request handle */
int res;
int n_executed;
int i;
T_CCI_ERROR error;
T_CCI_BLOB blob = NULL;
char data[1024] = "bulabula";

con = cci_connect ("localhost", 33000, "tdb", "PUBLIC", "");
if (con < 0) {
 goto handle_error;
}
req = cci_prepare (con, "insert into doc (doc_id, content) values
(?,?)", 0, &error);
```

```

if (req < 0)
{
 goto handle_error;
}

res = cci_bind_param (req, 1 /* binding index*/, CCI_A_TYPE_STR,
"doc-10", CCI_U_TYPE_STRING, CCI_BIND_PTR);

/* Creating an empty LOB data file */
res = cci_blob_new (con, &blob, &error);
res = cci_blob_write (con, blob, 0 /* start position */, 1024 /*
length */, data, &error);

/* Binding BLOB data */
res = cci_bind_param (req, 2 /* binding index*/, CCI_A_TYPE_BLOB,
(void *)blob, CCI_U_TYPE_BLOB, CCI_BIND_PTR);

n_executed = cci_execute (req, 0, 0, &error);
if (n_executed < 0)
{
 goto handle_error;
}

/* Commit */
if (cci_end_tran(con, CCI_TRAN_COMMIT, &error) < 0)
{
 goto handle_error;
}

/* Memory free */
cci_blob_free(blob);
return 0;

handle_error:
if (blob != NULL)
{
 cci_blob_free(blob);
}
if (req > 0)
{
 cci_close_req_handle (req);
}
if (con > 0)
{
 cci_disconnect(con, &error);
}
return -1;

```

## LOB 데이터 조회

CCI 응용 프로그램에서 다음 함수를 사용하여 **LOB** 데이터를 조회할 수 있다. **LOB** 타입 칼럼에 데이터를 입력하면 실제 **LOB** 데이터는 외부 저장소 내 파일에 저장되고 **LOB** 타입 칼럼에는 해당 파일을 참조하는 Locator 값이 저장되므로, 파일에 저장된 **LOB** 데이터를 조회하기 위해서는 `cci_get_data()` 가 아닌 `cci_blob_read()` 함수를 호출해야 한다.

- **LOB** 타입 칼럼 메타 데이터(Locator) 인출하기 (관련 함수: `cci_get_data()`)
- **LOB** 데이터를 인출하기 (관련 함수: `cci_blob_read()`)
- **LOB** 구조체에 대한 메모리 해제하기 (관련 함수: `cci_blob_free()`)

## 예제

```
int con = 0; /* connection handle */
int req = 0; /* request handle */
int ind; /* NULL indicator, 0 if not NULL, -1 if NULL*/
int res;
int i;
T_CCI_ERROR error;
T_CCI_BLOB blob;
char buffer[1024];

con = cci_connect ("localhost", 33000, "image_db", "PUBLIC", "");
if (con < 0)
{
 goto handle_error;
}
req = cci_prepare (con, "select content from doc_t", 0 /*flag*/,
&error);
if (req < 0)
{
 goto handle_error;
}

res = cci_execute (req, 0/*flag*/, 0/*max_col_size*/, &error);

while (1) {
 res = cci_cursor (req, 1/* offset */, CCI_CURSOR_CURRENT/*
cursor position */, &error);
 if (res == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 res = cci_fetch (req, &error);

 /* Fetching CLOB Locator */
 res = cci_get_data (req, 1 /* colume index */, CCI_A_TYPE_BLOB,
(void *)&blob /* BLOB handle */, &ind /* NULL indicator */);
 /* Fetching CLOB data */
 res = cci_blob_read (con, blob, 0 /* start position */, 1024 /*
```

```
length */, buffer, &error);
 printf ("content = %s\n", buffer);
}

/* Memory free */
cci_blob_free(blob);
res=cci_close_req_handle(req);
res = cci_disconnect (con, &error);
return 0;

handle_error:
if (req > 0)
{
 cci_close_req_handle (req);
}
if (con > 0)
{
 cci_disconnect(con, &error);
}
return -1;
```

## CCI 에러 코드와 에러 메시지

CCI API 함수는 에러 발생 시 반환 값이 음수인 CCI 에러 코드 혹은 CAS(브로커 응용 서버) 에러 코드를 반환한다. CCI 에러 코드는 CCI API 함수에서 발생하며, CAS 에러 코드는 CAS에서 발생한다.

- 모든 에러 코드의 값은 0보다 작은 음수이다.
- T\_CCI\_ERROR err\_buf를 인자로 가지는 모든 함수의 에러 코드와 에러 메시지는 err\_buf.err\_code 와 err\_buf.err\_msg에서 확인할 수 있다.
- T\_CCI\_ERROR err\_buf 인자가 없는 함수의 에러 메시지는 `cci_get_err_msg()` 함수를 이용하여 에러 코드가 나타내는 에러 메시지를 출력할 수 있다.
- 에러 번호가 -20002부터 -20999 사이이면, CCI API 함수에서 발생하는 에러이다.
- 에러 번호가 -10000부터 -10999 사이이면, CAS에서 발생하는 에러를 CCI API 함수가 전달받아 반환하는 에러이다. CAS 에러는 **CAS 에러**를 참고한다.
- 함수가 리턴하는 에러 코드의 값이 **CCI\_ER\_DBMS** (-20001)인 경우, 데이터베이스 서버에서 발생하는 에러이다. 데이터베이스 서버 에러와 관련한 내용은 **데이터베이스 서버 에러**를 참고한다.

### ⚠ 경고

서버에서 에러가 발생한 경우 함수가 리턴하는 에러 코드인 **CCI\_ER\_DBMS** 와 `err_buf.err_code` 값이 서로 다름에 주의한다. 서버 에러 외에 `err_buf`에 저장되는 모든 에러 코드는 함수가 리턴하는 에러 코드와 동일하다.

### 참고

CUBRID 9.0 미만 버전에서의 CCI, CAS 에러 코드는 CUBRID 9.0 이상 버전의 에러 코드와 다른 값을 가진다. 따라서 에러 코드명을 사용하여 개발한 사용자는 응용 프로그램을 재컴파일하여 사용해야 하며, 에러 코드 번호를 직접 부여하여 개발한 사용자는 번호 값을 바꾼 후 응용 프로그램을 재컴파일해야 한다.

데이터베이스 에러 버퍼(`err_buf`)는 **cas\_cci.h** 헤더 파일의 **T\_CCI\_ERROR** 구조체 변수이다. 사용법은 아래의 예제 프로그램을 참고한다.

**CCI\_ER** 로 시작되는 CCI 에러 코드는 `$CUBRID/include/cas_cci.h` 파일에 **T\_CCI\_ERROR\_CODE** 라는 enum 구조체 내에 정의되어 있다. 따라서 프로그램 코드에서 이 에러 코드 명을 사용하려면 코드 상단에 **#include "cas\_cci.h"** 를 입력하여 헤더 파일을 포함해야 한다.

아래의 프로그램은 에러 메시지를 출력한다. 이때 `cci_prepare()` 가 리턴하는 에러 코드 값 `req`의 값은 **CCI\_ER\_DMBS** 이고, 데이터베이스 에러 버퍼의 **cci\_error.err\_code** 에는 서버 에러 코드인 -493이, **cci\_error.err\_msg** 에는 'Syntax: Unknown class "notable". select \* from notable'이라는 에러 메시지가 저장된다.

```
// gcc -o err err.c -m64 -I${CUBRID}/cci/include -lnsl ${CUBRID}/cci/
lib/libcascci.so -lpthread
#include <stdio.h>
#include "cas_cci.h"

#define BUFSIZE (1024)

int
main (void)
{
 int con = 0, req = 0, col_count = 0, i, ind;
 int error;
 char *data;
 T_CCI_ERROR err_buf;
 char *query = "select * from notable";

 //getting a connection handle for a connection with a server
 con = cci_connect ("localhost", 33000, "demodb", "dba", "");
 if (con < 0)
 {
```

```

 printf ("cannot connect to database\n");
 return 1;
 }

 //preparing the SQL statement
 req = cci_prepare (con, query, 0, &err_buf);
 if (req < 0)
 {
 if (req == CCI_ER_DBMS)
 {
 printf ("error from server: %d, %s\n", err_buf.err_code,
err_buf.err_msg);
 }
 else
 {
 printf ("error from cci or cas: %d, %s\n",
err_buf.err_code, err_buf.err_msg);
 }
 goto handle_error;
 }
 // ...
}

```

다음은 CCI 함수의 에러 코드를 나타낸다. CAS 에러는 **CAS 에러**를 참고한다.

| 에러 코드명(에러 번호)                 | 에러 메시지                         | 비고                                                                                                  |
|-------------------------------|--------------------------------|-----------------------------------------------------------------------------------------------------|
| CCI_ER_DBMS(-20001)           | CUBRID DBMS Error              | 서버에서 에러가 발생한 경우<br>함수가 반환하는 에러 코드.<br>실패 원인은 T_CCI_ERROR 구<br>조체에 저장되는 err_code와<br>err_msg로 확인 가능. |
| CCI_ER_CON_HANDLE(-20002)     | Invalid connection handle      |                                                                                                     |
| CCI_ER_NO_MORE_MEMORY(-20003) | Memory allocation error        | 사용 가능한 메모리가 부족함.                                                                                    |
| CCI_ER_COMMUNICATION(-20004)  | Cannot communicate with server |                                                                                                     |
|                               | Invalid cursor position        |                                                                                                     |

| 에러 코드명(에러 번호)                  | 에러 메시지                          | 비고                                                                                                            |
|--------------------------------|---------------------------------|---------------------------------------------------------------------------------------------------------------|
| CCI_ER_NO_MORE_DATA(-20005)    |                                 |                                                                                                               |
| CCI_ER_TRAN_TYPE(-20006)       | Unknown transaction type        |                                                                                                               |
| CCI_ER_STRING_PARAM(-20007)    | Invalid string argument         | <code>cci_prepare()</code> , <code>cci_prepare_and_execute()</code> 에서 <code>sql_stmt</code> 가 NULL이면 발생하는 에러 |
| CCI_ER_TYPE_CONVERSION(-20008) | Type conversion error           | 주어진 타입의 값을 실제 데이터의 타입으로 변경할 수 없음.                                                                             |
| CCI_ER_BIND_INDEX(-20009)      | Parameter index is out of range | 바인드할 데이터의 index가 유효하지 않음.                                                                                     |
| CCI_ER_ATYPE(-20010)           | Invalid T_CCI_A_TYPE value      |                                                                                                               |
| CCI_ER_NOT_BIND(-20011)        |                                 | 사용되지 않음                                                                                                       |
| CCI_ER_PARAM_NAME(-20012)      | Invalid T_CCI_DB_PARAM value    |                                                                                                               |
| CCI_ER_COLUMN_INDEX(-20013)    | Column index is out of range    |                                                                                                               |
| CCI_ER_SCHEMA_TYPE(-20014)     |                                 | 사용되지 않음                                                                                                       |
| CCI_ER_FILE(-20015)            | Cannot open file                | 파일을 열거나 읽기/쓰기 실패함.                                                                                            |



| 에러 코드명(에러 번호)                     | 에러 메시지                              | 비고                                           |
|-----------------------------------|-------------------------------------|----------------------------------------------|
| CCI_ER_CONNECT(-20016)            | Cannot connect to CUBRID CAS        | 서버와 연결 시도 시 CAS 접속에 실패함.                     |
| CCI_ER_ALLOC_CON_HANDLE(-20017)   | Cannot allocate connection handle % |                                              |
| CCI_ER_REQ_HANDLE(-20018)         | Cannot allocate request handle %    |                                              |
| CCI_ER_INVALID_CURSOR_POS(-20019) | Invalid cursor position             |                                              |
| CCI_ER_OBJECT(-20020)             | Invalid oid string                  |                                              |
| CCI_ER_CAS(-20021)                |                                     | 사용되지 않음                                      |
| CCI_ER_HOSTNAME(-20022)           | Unknown host name                   |                                              |
| CCI_ER_OID_CMD(-20023)            | Invalid T_CCI_OID_CMD value         |                                              |
| CCI_ER_BIND_ARRAY_SIZE(-20024)    | Array binding size is not specified |                                              |
| CCI_ER_ISOLATION_LEVEL(-20025)    | Unknown transaction isolation level |                                              |
| CCI_ER_SET_INDEX(-20026)          | Invalid set index                   | T_CCI_SET 구조체에 포함된 set원소를 가져올 때 잘못된 위치가 지정됨. |

| 에러 코드명(에러 번호)                       | 에러 메시지                            | 비고                                                                       |
|-------------------------------------|-----------------------------------|--------------------------------------------------------------------------|
| CCI_ER_DELETED_TUPLE(-20027)        | Current row was deleted %         |                                                                          |
| CCI_ER_SAVEPOINT_CMD(-20028)        | Invalid T_CCI_SAVEPOINT_CMD value | <code>cci_savepoint()</code> 함수의 인자로 유효하지 않은 T_CCI_SAVEPOINT_CMD 값이 사용됨. |
| CCI_ER_THREAD_RUNNING(-20029)       | I                                 |                                                                          |
| CCI_ER_INVALID_URL(-20030)          | Invalid url string                |                                                                          |
| CCI_ER_INVALID_LOB_READ_POS(-20031) | Invalid lob read position         |                                                                          |
| CCI_ER_INVALID_LOB_HANDLE(-20032)   | Invalid lob handle                |                                                                          |
| CCI_ER_NO_PROPERTY(-20033)          | Could not find a property         |                                                                          |
| CCI_ER_PROPERTY_TYPE(-20034)        | Invalid property type             |                                                                          |
| CCI_ER_INVALID_DATASOURCE(-20035)   | Invalid CCI datasource            |                                                                          |
| CCI_ER_DATASOURCE_TIMEOUT(-20036)   | All connections are used          |                                                                          |
| CCI_ER_DATASOURCE_TIMEDWAIT(-20037) | pthread_cond_timedwait error      |                                                                          |

| 에러 코드명(에러 번호)                      | 에러 메시지                                                        | 비고 |
|------------------------------------|---------------------------------------------------------------|----|
| CCI_ER_LOGIN_TIMEOUT(-20038)       | Connection timed out                                          |    |
| CCI_ER_QUERY_TIMEOUT(-20039)       | Request timed out                                             |    |
| CCI_ER_RESULT_SET_CLOSED(-20040)   |                                                               |    |
| CCI_ER_INVALID_HOLDABILITY(-20041) | Invalid holdability mode. The only accepted values are 0 or 1 |    |
| CCI_ER_NOT_UPDATABLE(-20042)       | Request handle is not updatable                               |    |
| CCI_ER_INVALID_ARGS(-20043)        | Invalid argument                                              |    |
| CCI_ER_USED_CONNECTION(-20044)     | This connection is used already.                              |    |

## C Type Definition

다음은 CCI API 함수에서 사용하는 구조체들이다.

| 이름                 | 타입     | 멤버                 | 설명              |
|--------------------|--------|--------------------|-----------------|
| <b>T_CCI_ERROR</b> | struct | char err_msg[1024] | 데이터베이스 에러 정보 표현 |
|                    |        | int err_code       |                 |
| <b>T_CCI_BIT</b>   | struct | int size           | bit 타입 표현       |

| 이름            | 타입     | 멤버        | 설명                                          |
|---------------|--------|-----------|---------------------------------------------|
| T_CCI_DATE    | struct | char *buf | datetime,<br>timestamp, date,<br>time 타입 표현 |
|               |        | short yr  |                                             |
|               |        | short mon |                                             |
|               |        | short day |                                             |
|               |        | short hh  |                                             |
|               |        | short mm  |                                             |
|               |        | short ss  |                                             |
|               |        | short ms  |                                             |
| T_CCI_DATE_TZ | struct | short yr  | timezone과 date/<br>time 타입 표현               |
|               |        | short mon |                                             |
|               |        | short day |                                             |
|               |        | short hh  |                                             |
|               |        | short mm  |                                             |
|               |        | short ss  |                                             |
|               |        | short ms  |                                             |
|               |        |           |                                             |

| 이름                        | 타입     | 멤버                                                                             | 설명                           |
|---------------------------|--------|--------------------------------------------------------------------------------|------------------------------|
|                           |        | char tz[64]                                                                    |                              |
| <b>T_CCI_SET</b>          | void*  |                                                                                | set 타입 표현                    |
| <b>T_CCI_COL_INFO</b>     | struct | <b>T_CCI_U_EXT_TYPE</b><br>type                                                | <b>SELECT</b> 문에 대한 칼럼 정보 표현 |
|                           |        | char is_non_null                                                               |                              |
|                           |        | short scale                                                                    |                              |
|                           |        | int precision                                                                  |                              |
|                           |        | char *col_name                                                                 |                              |
|                           |        | char *real_attr                                                                |                              |
|                           |        | char *class_name                                                               |                              |
| <b>T_CCI_QUERY_RESULT</b> | struct | int result_count<br><br>int stmt_type<br><br>char *err_msg<br><br>char oid[32] | batch 실행에 대한 결과              |
| <b>T_CCI_PARAM_INFO</b>   | struct | <b>T_CCI_PARAM_MODE</b><br>mode                                                | input 파라미터에 대한 정보 표현         |

| 이름                        | 타입            | 멤버                             | 설명           |
|---------------------------|---------------|--------------------------------|--------------|
| <b>T_CCI_U_TYPE</b> type  |               |                                |              |
| short scale               |               |                                |              |
| int precision             |               |                                |              |
| <b>T_CCI_U_EXT_TYPE</b>   | unsigned char |                                | 데이터베이스 타입 정보 |
| <b>T_CCI_U_TYPE</b>       | enum          | <b>CCI_U_TYPE_UNKNO<br/>WN</b> | 데이터베이스 타입 정보 |
| <b>CCI_U_TYPE_NULL</b>    |               |                                |              |
| <b>CCI_U_TYPE_CHAR</b>    |               |                                |              |
| <b>CCI_U_TYPE_STRING</b>  |               |                                |              |
| <b>CCI_U_TYPE_BIT</b>     |               |                                |              |
| <b>CCI_U_TYPE_VARBIT</b>  |               |                                |              |
| <b>CCI_U_TYPE_NUMERIC</b> |               |                                |              |
| <b>CCI_U_TYPE_INT</b>     |               |                                |              |
| <b>CCI_U_TYPE_SHORT</b>   |               |                                |              |
| <b>CCI_U_TYPE_FLOAT</b>   |               |                                |              |

| 이름 | 타입 | 멤버                   | 설명 |
|----|----|----------------------|----|
|    |    | CCI_U_TYPE_DOUBLE    |    |
|    |    | CCI_U_TYPE_DATE      |    |
|    |    | CCI_U_TYPE_TIME      |    |
|    |    | CCI_U_TYPE_TIMESTAMP |    |
|    |    | CCI_U_TYPE_SET       |    |
|    |    | CCI_U_TYPE_MULTiset  |    |
|    |    | CCI_U_TYPE_SEQUENCE  |    |
|    |    | CCI_U_TYPE_OBJECT    |    |
|    |    | CCI_U_TYPE_BIGINT    |    |
|    |    | CCI_U_TYPE_DATETIME  |    |
|    |    | CCI_U_TYPE_BLOB      |    |
|    |    | CCI_U_TYPE_CLOB      |    |
|    |    | CCI_U_TYPE_ENUM      |    |
|    |    | CCI_U_TYPE_UINT      |    |

| 이름           | 타입   | 멤버                       | 설명                  |
|--------------|------|--------------------------|---------------------|
| T_CCI_A_TYPE | enum | CCI_U_TYPE_USHORT        | API에서 사용되는 타입 정보 표현 |
|              |      | CCI_U_TYPE_UBIGINT       |                     |
|              |      | CCI_U_TYPE_TIMESTAMPPTZ  |                     |
|              |      | CCI_U_TYPE_TIMESTAMPPLTZ |                     |
|              |      | CCI_U_TYPE_DATETIME_TZ   |                     |
|              |      | CCI_U_TYPE_DATETIME_LTZ  |                     |
|              |      | CCI_A_TYPE_STR           |                     |
|              |      | CCI_A_TYPE_INT           |                     |
|              |      | CCI_A_TYPE_FLOAT         |                     |
|              |      | CCI_A_TYPE_DOUBLE        |                     |
|              |      | CCI_A_TYPE_BIT           |                     |
|              |      | CCI_A_TYPE_DATE          |                     |
|              |      | CCI_A_TYPE_SET           |                     |



| 이름             | 타입   | 멤버                          | 설명          |
|----------------|------|-----------------------------|-------------|
|                |      | CCI_A_TYPE_BIGINT           |             |
|                |      | CCI_A_TYPE_BLOB             |             |
|                |      | CCI_A_TYPE_CLOB             |             |
|                |      | CCI_A_TYPE_CLOB             |             |
|                |      | CCI_A_TYPE_REQ_HANDLE       |             |
|                |      | CCI_A_TYPE_UINT             |             |
|                |      | CCI_A_TYPE_UBIGINT          |             |
|                |      | CCI_A_TYPE_DATE_TZ          |             |
|                |      | CCI_A_TYPE_UINT             |             |
|                |      | CCI_A_TYPE_UINT             |             |
| T_CCI_DB_PARAM | enum | CCI_PARAM_ISOLATION_LEVEL   | 시스템 파라미터 이름 |
|                |      | CCI_PARAM_LOCK_TIMEOUT      |             |
|                |      | CCI_PARAM_MAX_STRING_LENGTH |             |
|                |      | CCI_PARAM_AUTO_COMMIT       |             |

| 이름             | 타입   | 멤버                      | 설명 |
|----------------|------|-------------------------|----|
| T_CCI_SCH_TYPE | enum | CCI_SCH_CLASS           |    |
|                |      | CCI_SCH_VCLASS          |    |
|                |      | CCI_SCH_QUERY_SPEC      |    |
|                |      | CCI_SCH_ATTRIBUTE       |    |
|                |      | CCI_SCH_CLASS_ATTRIBUTE |    |
|                |      | CCI_SCH_METHOD          |    |
|                |      | CCI_SCH_CLASS_METHOD    |    |
|                |      | CCI_SCH_METHOD_FILE     |    |
|                |      | CCI_SCH_SUPERCLASS      |    |
|                |      | CCI_SCH_SUBCLASS        |    |
|                |      | CCI_SCH_CONSTRIT        |    |
|                |      | CCI_SCH_TRIGGER         |    |
|                |      | CCI_SCH_CLASS_PRIVILEGE |    |

| 이름                | 타입   | 멤버                             | 설명 |
|-------------------|------|--------------------------------|----|
|                   |      | CCI_SCH_ATTR_PRIVILEGE         |    |
|                   |      | CCI_SCH_DIRECT_SUP<br>ER_CLASS |    |
|                   |      | CCI_SCH_PRIMARY_KEY            |    |
|                   |      | CCI_SCH_IMPORTED_KEYS          |    |
|                   |      | CCI_SCH_EXPORTED_KEYS          |    |
|                   |      | CCI_SCH_CROSS_REFERENCE        |    |
| T_CCI_CUBRID_STMT | enum | CUBRID_STMT_ALTER<br>_CLASS    |    |
|                   |      | CUBRID_STMT_ALTER<br>_SERIAL   |    |
|                   |      | CUBRID_STMT_COMMIT_WORK        |    |
|                   |      | CUBRID_STMT_REGISTER_DATABASE  |    |
|                   |      | CUBRID_STMT_CREATE<br>_CLASS   |    |

| 이름 | 타입 | 멤버                         | 설명 |
|----|----|----------------------------|----|
|    |    | CUBRID_STMT_CREATE_INDEX   |    |
|    |    | CUBRID_STMT_CREATE_TRIGGER |    |
|    |    | CUBRID_STMT_CREATE_SERIAL  |    |
|    |    | CUBRID_STMT_DROP_DATABASE  |    |
|    |    | CUBRID_STMT_DROP_CLASS     |    |
|    |    | CUBRID_STMT_DROP_INDEX     |    |
|    |    | CUBRID_STMT_DROP_LABEL     |    |
|    |    | CUBRID_STMT_DROP_TRIGGER   |    |
|    |    | CUBRID_STMT_DROP_SERIAL    |    |
|    |    | CUBRID_STMT_EVALUATE       |    |
|    |    | CUBRID_STMT_RENAME_CLASS   |    |

| 이름 | 타입 | 멤버                            | 설명 |
|----|----|-------------------------------|----|
|    |    | CUBRID_STMT_ROLLBACK_WORK     |    |
|    |    | CUBRID_STMT_GRANT             |    |
|    |    | CUBRID_STMT_REVOKE            |    |
|    |    | CUBRID_STMT_STATISTICS        |    |
|    |    | CUBRID_STMT_INSERT            |    |
|    |    | CUBRID_STMT_SELECT            |    |
|    |    | CUBRID_STMT_UPDATE            |    |
|    |    | CUBRID_STMT_DELETE            |    |
|    |    | CUBRID_STMT_CALL              |    |
|    |    | CUBRID_STMT_GET_ISO_LVL       |    |
|    |    | CUBRID_STMT_GET_TIMEOUT       |    |
|    |    | CUBRID_STMT_GET_OPERATION_LVL |    |

| 이름 | 타입 | 멤버                           | 설명 |
|----|----|------------------------------|----|
|    |    | CUBRID_STMT_SET_OPERATOR_LVL |    |
|    |    | CUBRID_STMT_SCOPE            |    |
|    |    | CUBRID_STMT_GET_TRIGGER      |    |
|    |    | CUBRID_STMT_SET_TRIGGER      |    |
|    |    | CUBRID_STMT_SAVEPOINT        |    |
|    |    | CUBRID_STMT_PREPARE          |    |
|    |    | CUBRID_STMT_ATTACH           |    |
|    |    | CUBRID_STMT_USE              |    |
|    |    | CUBRID_STMT_REMOVE_TRIGGER   |    |
|    |    | CUBRID_STMT_RENAME_TRIGGER   |    |
|    |    | CUBRID_STMT_ON_LOAD          |    |

| 이름 | 타입 | 멤버                                      | 설명 |
|----|----|-----------------------------------------|----|
|    |    | CUBRID_STMT_GET_LD<br>B                 |    |
|    |    | CUBRID_STMT_SET_LD<br>B                 |    |
|    |    | CUBRID_STMT_GET_ST<br>ATS               |    |
|    |    | CUBRID_STMT_CREATE<br>_USER             |    |
|    |    | CUBRID_STMT_DROP_<br>USER               |    |
|    |    | CUBRID_STMT_ALTER_<br>USER              |    |
|    |    | CUBRID_STMT_SET_SY<br>S_PARAMS          |    |
|    |    | CUBRID_STMT_ALTER_I<br>NDEX             |    |
|    |    | CUBRID_STMT_CREATE<br>_STORED_PROCEDURE |    |
|    |    | CUBRID_STMT_DROP_S<br>TORED_PROCEDURE   |    |
|    |    | CUBRID_STMT_PREPAR<br>E_STATEMENT       |    |

| 이름               | 타입   | 멤버                                 | 설명 |
|------------------|------|------------------------------------|----|
| T_CCI_CURSOR_POS | enum | CUBRID_STMT_EXECUTE_PREPARE        |    |
|                  |      | CUBRID_STMT_DEALLOCATE_PREPARE     |    |
|                  |      | CUBRID_STMT_TRUNCATE               |    |
|                  |      | CUBRID_STMT_DO                     |    |
|                  |      | CUBRID_STMT_SELECT_UPDATE          |    |
|                  |      | CUBRID_STMT_SET_SESSION_VARIABLES  |    |
|                  |      | CUBRID_STMT_DROP_SESSION_VARIABLES |    |
|                  |      | CUBRID_STMT_MERGE                  |    |
|                  |      | CUBRID_STMT_SET_NAMES              |    |
|                  |      | CUBRID_STMT_ALTER_STORED_PROCEDURE |    |
|                  |      | CUBRID_STMT_KILL                   |    |
| T_CCI_CURSOR_POS | enum | CCI_CURSOR_FIRST                   |    |



| 이름                   | 타입   | 멤버                     | 설명 |
|----------------------|------|------------------------|----|
|                      |      | CCI_CURSOR_CURRENT     |    |
|                      |      | CCI_CURSOR_LAST        |    |
|                      |      |                        |    |
| T_CCI_TRAN_ISOLATION | enum | TRAN_READ_COMMITTED    |    |
|                      |      | TRAN_REPEATABLE_READ   |    |
|                      |      | TRAN_SERIALIZABLE      |    |
| T_CCI_PARAM_MODE     | enum | CCI_PARAM_MODE_UNKNOWN |    |
|                      |      | CCI_PARAM_MODE_IN      |    |
|                      |      | CCI_PARAM_MODE_OUT     |    |
|                      |      | CCI_PARAM_MODE_INOUT   |    |

#### 참고

칼럼에서 정의한 크기보다 큰 문자열을 **INSERT** / **UPDATE** 하면 문자열이 잘려서 입력된다.

## CCI 예제 프로그램

예제 프로그램은 CUBRID 설치 과정에서 기본적으로 배포되는 데이터베이스인 *demodb* 를 활용하여 CCI를 사용하는 응용 프로그램을 간단하게 작성한 것이다. 예제를 통하여 CAS와 연결하기, 질의

준비, 질의 수행, 응답 처리, 연결 끊기 등의 과정을 따라한다. 예제는 Linux 기반의 동적 링크를 적용하는 방법으로 작성되었다.

다음은 예제에서 사용하는 *demodb* 데이터베이스의 *olympic* 테이블의 스키마 정보이다.

```
csql> ;sc olympic

=== <Help: Schema of a Class> ===

<Class Name>

 olympic

<Attributes>

 host_year INTEGER NOT NULL
 host_nation CHARACTER VARYING(40) NOT NULL
 host_city CHARACTER VARYING(20) NOT NULL
 opening_date DATE NOT NULL
 closing_date DATE NOT NULL
 mascot CHARACTER VARYING(20)
 slogan CHARACTER VARYING(40)
 introduction CHARACTER VARYING(1500)

<Constraints>

 PRIMARY KEY pk_olympic_host_year ON olympic (host_year)
```

## 준비

예제 프로그램을 수행하기 전에 반드시 확인해야 할 사항은 *demodb* 데이터베이스와 브로커의 가동 여부이다. *demodb* 데이터베이스와 브로커는 **cubrid** 유틸리티를 이용하여 시작할 수 있다. 다음은 **cubrid** 유틸리티를 이용하여 데이터베이스 서버와 브로커를 가동하는 예제이다.

```
[tester@testdb ~]$ cubrid server start demodb
@ cubrid master start
++ cubrid master start: success
@ cubrid server start: demodb

This may take a long time depending on the amount of recovery works
to do.

CUBRID 9.2

++ cubrid server start: success
[tester@testdb ~]$ cubrid broker start
```

```
@ cubrid broker start
++ cubrid broker start: success
```

## 빌드

프로그램 소스와 Makefile이 준비된 상태에서 **make** 를 수행하면 *test* 라는 실행 파일이 생성된다. 정적 라이브러리를 사용하면 추가로 파일을 배포할 필요가 없고 속도가 빠르다. 하지만, 프로그램의 크기와 메모리 사용량이 커지는 단점이 있다. 동적 라이브러리를 사용하면 성능상의 오버헤드는 있지만, 메모리와 프로그램 크기에 있어 최적화를 이룰 수 있다.

다음은 Linux에서 **make** 를 사용하지 않고 동적인 라이브러리를 사용하여 테스트 프로그램을 빌드하는 명령 행의 예제이다.

```
cc -o test test.c -I$CUBRID/cci/include -L$CUBRID/cci/lib -lnsl -lcascci
```

## 예제 코드

```
#include <stdio.h>
#include <cas_cci.h>
char *cci_client_name = "test";
int main (int argc, char *argv[])
{
 int con = 0, req = 0, col_count = 0, res, ind, i;
 T_CCI_ERROR error;
 T_CCI_COL_INFO *res_col_info;
 T_CCI_CUBRID_STMT stmt_type;
 char *buffer, db_ver[16];
 printf("Program started!\n");
 if ((con=cci_connect("localhost", 30000, "demodb", "PUBLIC",
 ""))<0) {
 printf("%s(%d): cci_connect fail\n", __FILE__, __LINE__);
 return -1;
 }

 if ((res=cci_get_db_version(con, db_ver, sizeof(db_ver)))<0) {
 printf("%s(%d): cci_get_db_version fail\n", __FILE__,
 __LINE__);
 goto handle_error;
 }
 printf("DB Version is %s\n",db_ver);
 if ((req=cci_prepare(con, "select * from event", 0,&error))<0) {
 if (req < 0) {
 printf("%s(%d): cci_prepare fail(%d)\n", __FILE__,
 __LINE__,error.err_code);
 }
 goto handle_error;
 }
}
```

```

printf("Prepare ok!(%d)\n",req);
res_col_info = cci_get_result_info(req, &stmt_type, &col_count);
if (!res_col_info) {
 printf("%s(%d): cci_get_result_info fail\n", __FILE__,
__LINE__);
 goto handle_error;
}

printf("Result column information\n"
"=====\n");
for (i=1; i<=col_count; i++) {
 printf("name:%s type:%d(precision:%d scale:%d)\n",
 CCI_GET_RESULT_INFO_NAME(res_col_info, i),
 CCI_GET_RESULT_INFO_TYPE(res_col_info, i),
 CCI_GET_RESULT_INFO_PRECISION(res_col_info, i),
 CCI_GET_RESULT_INFO_SCALE(res_col_info, i));
}
printf("=====\n");
if ((res=cci_execute(req, 0, 0, &error))<0) {
 if (req < 0) {
 printf("%s(%d): cci_execute fail(%d)\n", __FILE__,
__LINE__,error.err_code);
 }
 goto handle_error;
}

while (1) {
 res = cci_cursor(req, 1, CCI_CURSOR_CURRENT, &error);
 if (res == CCI_ER_NO_MORE_DATA) {
 printf("Query END!\n");
 break;
 }
 if (res<0) {
 if (req < 0) {
 printf("%s(%d): cci_cursor fail(%d)\n", __FILE__,
__LINE__,error.err_code);
 }
 goto handle_error;
 }

 if ((res=cci_fetch(req, &error))<0) {
 if (res < 0) {
 printf("%s(%d): cci_fetch fail(%d)\n", __FILE__,
__LINE__,error.err_code);
 }
 goto handle_error;
 }

 for (i=1; i<=col_count; i++) {
 if ((res=cci_get_data(req, i, CCI_A_TYPE_STR, &buffer,
&ind))<0) {
 printf("%s(%d): cci_get_data fail\n", __FILE__,

```

```

__LINE__);
 goto handle_error;
 }
 printf("%s \t|", buffer);
}
printf("\n");
}
if ((res=cci_close_req_handle(req))<0) {
 printf("%s(%d): cci_close_req_handle fail", __FILE__,
__LINE__);
 goto handle_error;
}
if ((res=cci_disconnect(con, &error))<0) {
 if (res < 0) {
 printf("%s(%d): cci_disconnect fail(%d)", __FILE__,
__LINE__, error.err_code);
 }
 goto handle_error;
}
printf("Program ended!\n");
return 0;

handle_error:
if (req > 0)
 cci_close_req_handle(req);
if (con > 0)
 cci_disconnect(con, &error);
printf("Program failed!\n");
return -1;
}

```

## CCI API 레퍼런스

### 목차

|                             |             |
|-----------------------------|-------------|
| <b>CCI API 레퍼런스</b>         | <b>2297</b> |
| ● cci_bind_param            | 2303        |
| ● cci_bind_param_array      | 2307        |
| ● cci_bind_param_array_size | 2308        |
| ● cci_bind_param_ex         | 2309        |

|                          |      |
|--------------------------|------|
| ● cci_blob_free          | 2309 |
| ● cci_blob_new           | 2310 |
| ● cci_blob_read          | 2310 |
| ● cci_blob_size          | 2311 |
| ● cci_blob_write         | 2312 |
| ● cci_cancel             | 2313 |
| ● cci_clob_free          | 2317 |
| ● cci_clob_new           | 2317 |
| ● cci_clob_read          | 2318 |
| ● cci_clob_size          | 2319 |
| ● cci_clob_write         | 2320 |
| ● cci_close_query_result | 2320 |
| ● cci_close_req_handle   | 2321 |
| ● cci_col_get            | 2322 |
| ● cci_col_seq_drop       | 2323 |
| ● cci_col_seq_insert     | 2324 |
| ● cci_col_seq_put        | 2324 |
| ● cci_col_set_add        | 2325 |
| ● cci_col_set_drop       | 2326 |
| ● cci_col_size           | 2327 |
| ● cci_connect            | 2327 |

|                                  |      |
|----------------------------------|------|
| ● cci_connect_ex                 | 2328 |
| ● cci_connect_with_url           | 2329 |
| ● cci_connect_with_url_ex        | 2333 |
| ● cci_cursor                     | 2334 |
| ● cci_cursor_update              | 2335 |
| ● cci_datasource_borrow          | 2336 |
| ● cci_datasource_change_property | 2337 |
| ● cci_datasource_create          | 2339 |
| ● cci_datasource_destroy         | 2340 |
| ● cci_datasource_release         | 2340 |
| ● cci_disconnect                 | 2341 |
| ● cci_end_tran                   | 2341 |
| ● cci_escape_string              | 2342 |
| ● cci_execute                    | 2343 |
| ● cci_execute_array              | 2345 |
| ● cci_execute_batch              | 2348 |
| ● cci_execute_result             | 2350 |
| ● cci_fetch                      | 2351 |
| ● cci_fetch_buffer_clear         | 2352 |
| ● cci_fetch_sensitive            | 2352 |
| ● cci_fetch_size                 | 2353 |

|                                   |      |
|-----------------------------------|------|
| ● cci_get_autocommit              | 2353 |
| ● cci_get_bind_num                | 2354 |
| ● cci_get_cas_info                | 2354 |
| ● cci_get_class_num_objs          | 2355 |
| ● CCI_GET_COLLECTION_DOMAIN       | 2356 |
| ● cci_get_cur_oid                 | 2356 |
| ● cci_get_data                    | 2357 |
| ● cci_get_db_parameter            | 2360 |
| ● cci_get_db_version              | 2361 |
| ● cci_get_err_msg                 | 2362 |
| ● cci_get_error_msg               | 2363 |
| ● cci_get_holdability             | 2364 |
| ● cci_get_last_insert_id          | 2364 |
| ● cci_get_login_timeout           | 2366 |
| ● cci_get_query_plan              | 2366 |
| ● cci_query_info_free             | 2367 |
| ● cci_get_query_timeout           | 2368 |
| ● cci_get_result_info             | 2368 |
| ● CCI_GET_RESULT_INFO_ATTR_NAME   | 2370 |
| ● CCI_GET_RESULT_INFO_CLASS_NAME  | 2370 |
| ● CCI_GET_RESULT_INFO_IS_NON_NULL | 2371 |



|                                 |      |
|---------------------------------|------|
| ● CCI_GET_RESULT_INFO_NAME      | 2371 |
| ● CCI_GET_RESULT_INFO_PRECISION | 2372 |
| ● CCI_GET_RESULT_INFO_SCALE     | 2372 |
| ● CCI_GET_RESULT_INFO_TYPE      | 2373 |
| ● CCI_IS_SET_TYPE               | 2373 |
| ● CCI_IS_MULTISSET_TYPE         | 2374 |
| ● CCI_IS_SEQUENCE_TYPE          | 2374 |
| ● CCI_IS_COLLECTION_TYPE        | 2374 |
| ● cci_get_version               | 2375 |
| ● cci_init                      | 2375 |
| ● cci_is_holdable               | 2376 |
| ● cci_is_updatable              | 2376 |
| ● cci_next_result               | 2377 |
| ● cci_oid                       | 2378 |
| ● cci_oid_get                   | 2379 |
| ● cci_oid_get_class_name        | 2379 |
| ● cci_oid_put                   | 2380 |
| ● cci_oid_put2                  | 2381 |
| ● cci_prepare                   | 2383 |
| ● cci_prepare_and_execute       | 2384 |
| ● cci_property_create           | 2385 |

|                              |      |
|------------------------------|------|
| ● cci_property_destroy       | 2385 |
| ● cci_property_get           | 2386 |
| ● cci_property_set           | 2386 |
| ● cci_query_result_free      | 2391 |
| ● CCI_QUERY_RESULT_ERR_NO    | 2392 |
| ● CCI_QUERY_RESULT_ERR_MSG   | 2392 |
| ● CCI_QUERY_RESULT_RESULT    | 2393 |
| ● CCI_QUERY_RESULT_STMT_TYPE | 2393 |
| ● cci_register_out_param     | 2394 |
| ● cci_row_count              | 2397 |
| ● cci_savepoint              | 2398 |
| ● cci_schema_info            | 2399 |
| ● cci_set_allocators         | 2412 |
| ● cci_set_autocommit         | 2416 |
| ● cci_set_db_parameter       | 2417 |
| ● cci_set_element_type       | 2417 |
| ● cci_set_free               | 2418 |
| ● cci_set_get                | 2418 |
| ● cci_set_holdability        | 2419 |
| ● cci_set_isolation_level    | 2420 |
| ● cci_set_lock_timeout       | 2421 |

|                         |      |
|-------------------------|------|
| ● cci_set_login_timeout | 2421 |
| ● cci_set_make          | 2422 |
| ● cci_set_max_row       | 2422 |
| ● cci_set_query_timeout | 2423 |
| ● cci_set_size          | 2424 |

## cci\_bind\_param

int cci\_bind\_param(int req\_handle, int index, T\_CCI\_A\_TYPE a\_type, void \*value, T\_CCI\_U\_TYPE u\_type, char flag)

prepared statement에서 *bind* 변수에 데이터를 바인딩하기 위하여 사용되는 함수이다. 이때, 주어진 *a\_type* 의 *value* 의 값을 실제 바인딩되어야 하는 타입으로 변환하여 저장한다. 이후, `cci_execute()` 가 호출될 때 저장된 데이터가 서버로 전송된다. 같은 *index* 에 대해서 여러 번 `cci_bind_param()` 을 호출할 경우 마지막으로 설정한 값이 유효하다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들.
- **index** – (IN) 바인딩될 위치이며, 1부터 시작.
- **a\_type** – (IN) *value* 의 타입.
- **value** – (IN) 바인딩될 데이터 값.
- **u\_type** – (IN) 데이터베이스에 반영될 데이터 타입.
- **flag** – (IN) bind\_flag(`CCI_BIND_PTR`).

반환:

에러 코드(0: 성공)

- **CCI\_ER\_BIND\_INDEX**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_REQ\_HANDLE**

- **CCI\_ER\_TYPE\_CONVERSION**

- **CCI\_ER\_USED\_CONNECTION**

데이터베이스에 **NULL**을 바인딩하려면 다음의 두 가지 중 하나를 설정한다.

- *value* 값을 **NULL** 포인터로 설정
- *u\_type*을 **CCI\_U\_TYPE\_NULL** 로 설정

다음은 NULL을 바인딩하는 코드의 일부이다.

```
res = cci_bind_param (req, 2 /* binding index */,
CCI_A_TYPE_STR, NULL, CCI_U_TYPE_STRING, CCI_BIND_PTR);
```

또는

```
res = cci_bind_param (req, 2 /* binding index */,
CCI_A_TYPE_STR, data, CCI_U_TYPE_NULL, CCI_BIND_PTR);
```

가 사용될 수 있다.

*flag*에 **CCI\_BIND\_PTR** 이 설정되어 있을 경우 *value* 변수의 포인터만 복사하고(shallow copy) 값은 복사하지 않는다. *flag*가 설정되지 않는 경우 메모리를 할당하여 *value* 변수의 값을 복사(deep copy)한다. 만약 같은 메모리 버퍼를 이용하여 여러 개의 칼럼을 바인딩할 경우라면, **CCI\_BIND\_PTR** *flag*를 설정하지 않아야 한다.

**T\_CCI\_A\_TYPE** 은 CCI 응용 프로그램 내에서 데이터 바인딩에 사용되는 C 언어의 타입을 의미하며, int, float 등의 primitive 타입과 **T\_CCI\_BIT**, **T\_CCI\_DATE** 등의 CCI 가 정의한 user-defined 타입으로 구성된다. 각 타입에 대한 식별자는 아래의 표와 같이 정의되어 있다.

| a_type                   | value 타입 |
|--------------------------|----------|
| <b>CCI_A_TYPE_STR</b>    | char *   |
| <b>CCI_A_TYPE_INT</b>    | int *    |
| <b>CCI_A_TYPE_FLOAT</b>  | float *  |
| <b>CCI_A_TYPE_DOUBLE</b> | double * |

| a_type            | value 타입                           |
|-------------------|------------------------------------|
| CCI_A_TYPE_BIT    | T_CCI_BIT *                        |
| CCI_A_TYPE_SET    | T_CCI_SET                          |
| CCI_A_TYPE_DATE   | T_CCI_DATE *                       |
| CCI_A_TYPE_BIGINT | int64_t * (For Windows: __int64 *) |
| CCI_A_TYPE_BLOB   | T_CCI_BLOB                         |
| CCI_A_TYPE_CLOB   | T_CCI_CLOB                         |

`T_CCI_U_TYPE` 은 데이터베이스의 칼럼 타입으로, value 인자를 통해 바인딩된 데이터를 이 타입으로 변환한다. `cci_bind_param()` 함수는 C 언어가 이해하는 A 타입의 데이터를 데이터베이스가 이해할 수 있는 U 타입의 데이터로 변환하기 위한 정보를 전달하기 위해서 두 가지 타입을 사용한다.

U 타입이 허용하는 A 타입은 여러 가지이다. 예를 들어 `CCI_U_TYPE_INT` 는 `CCI_A_TYPE_INT` 외에 `CCI_A_TYPE_STR` 도 A 타입으로 받을 수 있다. 타입 변환은 **묵시적 타입 변환**을 따른다.

`T_CCI_A_TYPE` 및 `T_CCI_U_TYPE` enum은 모두 `cas_cci.h` 파일에 정의되어 있다. 각 타입에 대한 식별자 정의는 아래 표를 참고한다.

| u_type            | 대응되는 기본 a_type |
|-------------------|----------------|
| CCI_U_TYPE_CHAR   | CCI_A_TYPE_STR |
| CCI_U_TYPE_STRING | CCI_A_TYPE_STR |
| CCI_U_TYPE_BIT    | CCI_A_TYPE_BIT |
| CCI_U_TYPE_VARBIT | CCI_A_TYPE_BIT |

| <b>u_type</b>               | <b>대응되는 기본 a_type</b>    |
|-----------------------------|--------------------------|
| <b>CCI_U_TYPE_NUMERIC</b>   | <b>CCI_A_TYPE_STR</b>    |
| <b>CCI_U_TYPE_INT</b>       | <b>CCI_A_TYPE_INT</b>    |
| <b>CCI_U_TYPE_SHORT</b>     | <b>CCI_A_TYPE_INT</b>    |
| <b>CCI_U_TYPE_FLOAT</b>     | <b>CCI_A_TYPE_FLOAT</b>  |
| <b>CCI_U_TYPE_DOUBLE</b>    | <b>CCI_A_TYPE_DOUBLE</b> |
| <b>CCI_U_TYPE_DATE</b>      | <b>CCI_A_TYPE_DATE</b>   |
| <b>CCI_U_TYPE_TIME</b>      | <b>CCI_A_TYPE_DATE</b>   |
| <b>CCI_U_TYPE_TIMESTAMP</b> | <b>CCI_A_TYPE_DATE</b>   |
| <b>CCI_U_TYPE_OBJECT</b>    | <b>CCI_A_TYPE_STR</b>    |
| <b>CCI_U_TYPE_BIGINT</b>    | <b>CCI_A_TYPE_BIGINT</b> |
| <b>CCI_U_TYPE_DATETIME</b>  | <b>CCI_A_TYPE_DATE</b>   |
| <b>CCI_U_TYPE_BLOB</b>      | <b>CCI_A_TYPE_BLOB</b>   |
| <b>CCI_U_TYPE_CLOB</b>      | <b>CCI_A_TYPE_CLOB</b>   |
| <b>CCI_U_TYPE_ENUM</b>      | <b>CCI_A_TYPE_STR</b>    |

날짜를 포함하는 문자열을 **DATE**, **DATETIME** 또는 **TIMESTAMP**의 입력 인자로 할 때, 날짜 문자열의 형식은 “YYYY/MM/DD” 형식 또는 “YYYY-MM-DD” 형식만 허용한다. 즉, “2012/01/31” 또는

“2012-01-31”과 같은 형식은 허용하지만 “01/31/2012”와 같은 형식은 허용하지 않는다. 날짜를 포함하는 문자열을 날짜 타입의 입력 인자로 하는 예는 다음과 같다.

```
// "CREATE TABLE tbl(aa date, bb datetime)";

char *values[][2] =
{
 {"1994/11/30", "1994/11/30 20:08:08"},
 {"2008-10-31", "2008-10-31 20:08:08"}
};

req = cci_prepare(conn, "insert into tbl (aa, bb) values
(?, ?)", CCI_PREPARE_INCLUDE_OID, &error);

for(i=0; i< 2; i++)
{
 res = cci_bind_param(req, 1, CCI_A_TYPE_STR, values[i][0],
CCI_U_TYPE_DATE, (char)0);
 res = cci_bind_param(req, 2, CCI_A_TYPE_STR, values[i][1],
CCI_U_TYPE_DATETIME, (char)0);
 cci_execute(req, CCI_EXEC_QUERY_ALL, 0, err_buf);
}
```

## cci\_bind\_param\_array

int cci\_bind\_param\_array(int req\_handle, int index, T\_CCI\_A\_TYPE a\_type, void \*value, int \*null\_ind, T\_CCI\_U\_TYPE u\_type)

prepare된 *req\_handle* 에 대해서 파라미터 배열을 바인딩한다. 이후, `cci_execute_array()` 가 호출될 때 저장된 *value* 포인터에 의해 데이터가 서버로 전송된다. 같은 *index* 에 대해서 여러 번 `cci_bind_param_array()` 가 호출될 경우 마지막 설정된 값이 유효하다. 데이터에 **NULL**을 바인딩할 경우 *null\_ind*에 0이 아닌 값을 설정한다. *value* 값이 **NULL** 포인터인 경우, 또는 *u\_type*이 **CCI\_U\_TYPE\_NULL**인 경우 모든 데이터가 **NULL**로 바인딩되며 *value*에 의해 사용되는 데이터 버퍼는 재사용될 수 없다. *a\_type*에 대한 *value*의 데이터 타입은 `cci_bind_param()` 의 설명을 참조한다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **index** – (IN) 바인딩될 위치
- **a\_type** – (IN) *value* 의 타입
- **value** – (IN) 바인딩될 데이터 값
- **null\_ind** – (IN) **NULL** 식별자 배열(0 : not **NULL**, 1 : **NULL**)

- **u\_type** – (IN) 데이터베이스에 반영될 데이터 타입

반환:

에러 코드(0: 성공)

- **CCI\_ER\_BIND\_INDEX**
- **CCI\_ER\_BIND\_ARRAY\_SIZE**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_TYPE\_CONVERSION**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_bind\_param\_array\_size

int cci\_bind\_param\_array\_size(int req\_handle, int array\_size)

`cci_bind_param_array()` 에서 사용될 array의 크기를 결정한다. `cci_bind_param_array()` 가 사용되기 전에 반드시 `cci_bind_param_array_size()` 가 먼저 호출 되어야 한다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **array\_size** – (IN) 바인딩할 배열 크기

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**



## cci\_bind\_param\_ex

int cci\_bind\_param\_ex(int req\_handle, int index, T\_CCI\_A\_TYPE a\_type, void \*value, int length, T\_CCI\_U\_TYPE u\_type, char flag)

`cci_bind_param()` 과 동일한 동작을 수행한다. 다만 문자열 타입인 경우 문자열의 바이트 길이를 지정하는 *length* 인자가 추가로 존재한다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **index** – (IN) 바인딩될 위치이며, 1부터 시작
- **a\_type** – (IN) *value* 의 타입
- **value** – (IN) 바인딩할 데이터 값
- **length** – (IN) 바인딩할 문자열의 바이트 길이
- **u\_type** – (IN) 데이터베이스에 반영될 데이터 타입
- **flag** – (IN) bind\_flag(`CCI_BIND_PTR`)

반환:

에러 코드(0: 성공)

*length* 인자는 아래와 같이 '\0'을 포함하는 문자열을 바인딩하기 위해 사용할 수 있다.

```
cci_bind_param_ex(req, 1, CCI_A_TYPE_STR, "aaa\0bbb", 7,
CCI_U_TYPE_STRING, 0);
```

## cci\_blob\_free

int cci\_blob\_free(T\_CCI\_BLOB blob)

**BLOB** 구조체에 대한 메모리를 해제한다.

반환:

에러 코드(0: 성공)

- **CCI\_ER\_INVALID\_LOB\_HANDLE**

## cci\_blob\_new

int cci\_blob\_new(int conn\_handle, T\_CCI\_BLOB \*blob, T\_CCI\_ERROR \*error\_buf)

**LOB** 데이터가 저장될 빈 파일을 하나 생성하고, 해당 파일을 참조하는 Locator를 *blob* 구조체에 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **blob** – (OUT) **LOB** Locator
- **error\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_INVALID\_LOB\_HANDLE**
- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_blob\_read

int cci\_blob\_read(int conn\_handle, T\_CCI\_BLOB blob, long long start\_pos, int length, char \*buf, T\_CCI\_ERROR \*error\_buf)

*blob*에 명시한 **LOB** 데이터 파일의 *start\_pos*부터 *length*만큼 데이터를 읽어 *buf*에 저장한 후 이를 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

- **blob** – (IN) **LOB** Locator
- **start\_pos** – (IN) **LOB** 데이터 파일의 위치 인덱스
- **length** – (IN) 파일로부터 가져올 **LOB** 데이터 길이
- **buf** – (IN) 데이터 읽기 버퍼
- **error\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_INVALID\_LOB\_HANDLE**
- **CCI\_ER\_INVALID\_LOB\_READ\_POS**
- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_QUERY\_TIMEOUT**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_blob\_size

long long cci\_blob\_size(T\_CCI\_BLOB blob)

*blob* 에 명시한 데이터 파일의 크기를 반환한다.

매개변수:

- **blob** – (IN) **LOB** Locator

반환:

**BLOB** 데이터 파일의 크기( $\geq 0$  : 성공), 에러 코드( $< 0$  : 에러)

- **CCI\_ER\_INVALID\_LOB\_HANDLE**

## cci\_blob\_write

int cci\_blob\_write(int conn\_handle, T\_CCI\_BLOB blob, long long start\_pos, int length, const char \*buf, T\_CCI\_ERROR \*error\_buf)

*buf*로부터 *length* 만큼 데이터를 읽어 *blob*에 명시한 **LOB** 데이터 파일의 *start\_pos*부터 저장한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **blob** – (IN) **LOB** Locator
- **start\_pos** – (IN) **LOB** 데이터 파일의 위치 인덱스
- **length** – (IN) 버퍼로부터 가져올 데이터 길이
- **buf** – (OUT) 데이터 쓰기 버퍼
- **error\_buf** – (OUT) 에러 버퍼

반환:

write한 크기(>=0 : 성공), 에러 코드(<0 : 에러)

- CCI\_ER\_COMMUNICATION
- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_CONNECT
- CCI\_ER\_DBMS
- CCI\_ER\_INVALID\_LOB\_HANDLE
- CCI\_ER\_LOGIN\_TIMEOUT
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_QUERY\_TIMEOUT
- CCI\_ER\_USED\_CONNECTION

## cci\_cancel

int cci\_cancel(int conn\_handle)

다른 스레드에서 실행 중인 질의를 취소시킨다. Java의 Statement.cancel() 메서드와 같은 기능을 수행한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

반환:

에러 코드

- CCI\_ER\_COMMUNICATION
- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_CONNECT

다음은 main 함수에서 스레드의 질의 실행을 취소하는 예이다.

```
/* gcc -o pthr pthr.c -m64 -I${CUBRID}/include -lnsl ${CUBRID}/
lib/libcascci.so -lpthread
```

```

*/

#include <stdio.h>
#include <cas_cci.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>
#include <time.h>

#define QUERY "select * from db_class A, db_class B, db_class C,
db_class D, db_class E"

static void *thread_main (void *arg);
static void *execute_statement (int con, char *sql_stmt);

int
main (int argc, char *argv[])
{
 int thr_id = 0, conn_handle = 0, res = 0;
 void *jret;
 pthread_t th;
 char url[1024];
 T_CCI_ERROR error;
 snprintf (url, 1024, "cci:CUBRID:localhost:
33000:demodb:PUBLIC::");

 conn_handle = cci_connect_with_url_ex (url, NULL, NULL,
&error);

 if (conn_handle < 0)
 {
 printf ("ERROR: %s\n", error.err_msg);
 return -1;
 }

 res = pthread_create (&th, NULL, &thread_main, (void *)
&conn_handle);

 if (res < 0)
 {
 printf ("thread fork failed.\n");
 return -1;
 }
 else
 {
 printf ("thread started\n");
 }
 sleep (5);
 // If thread_main is still running, below cancels the query
of thread_main.
 res = cci_cancel (conn_handle);

```

```

 if (res < 0)
 {
 printf ("cci_cancel failed\n");
 return -1;
 }
 else
 {
 printf ("The query was canceled by cci_cancel.\n");
 }
 res = pthread_join (th, &jret);
 if (res < 0)
 {
 printf ("thread join failed.\n");
 return -1;
 }

 printf ("thread_main was cancelled with\n\t%s\n", (char *)
jret);
 free (jret);

 res = cci_disconnect (conn_handle, &error);
 if (res < 0)
 {
 printf ("ERROR: %s\n", error.err_msg);
 return res;
 }

 return 0;
}

void *
thread_main (void *arg)
{
 int con = *((int *) arg);
 int ret_val;
 void *ret_ptr;
 T_CCI_ERROR error;

 cci_set_autocommit (con, CCI_AUTOCOMMIT_TRUE);
 ret_ptr = execute_statement (con, QUERY);
 return ret_ptr;
}

static void *
execute_statement (int con, char *sql_stmt)
{
 int col_count = 1, ind, i, req;
 T_CCI_ERROR error;
 char *buffer;
 char *error_msg;
 int res = 0;

```

```

 error_msg = (char *) malloc (128);
 if ((req = cci_prepare (con, sql_stmt, 0, &error)) < 0)
 {
 snprintf (error_msg, 128, "cci_prepare ERROR: %s\n",
error.err_msg);
 goto conn_err;
 }

 if ((res = cci_execute (req, 0, 0, &error)) < 0)
 {
 snprintf (error_msg, 128, "cci_execute ERROR: %s\n",
error.err_msg);
 goto execute_error;
 }

 if (res >= 0)
 {
 while (1)
 {
 res = cci_cursor (req, 1, CCI_CURSOR_CURRENT,
&error);
 if (res == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 if (res < 0)
 {
 snprintf (error_msg, 128, "cci_cursor ERROR:
%s\n",
 error.err_msg);
 return error_msg;
 }

 if ((res = cci_fetch (req, &error)) < 0)
 {
 snprintf (error_msg, 128, "cci_fetch ERROR:
%s\n",
 error.err_msg);
 return error_msg;
 }

 for (i = 1; i <= col_count; i++)
 {
 if ((res = cci_get_data (req, i, CCI_A_TYPE_STR,
&buffer, &ind)) < 0)
 {
 snprintf (error_msg, 128, "cci_get_data
ERROR\n");
 return error_msg;
 }
 }
 }
 }
}

```



```

 }

 if ((res = cci_close_query_result (req, &error)) < 0)
 {
 snprintf (error_msg, 128, "cci_close_query_result ERROR:
%s\n", error.err_msg);
 return error_msg;
 }
execute_error:
 if ((res = cci_close_req_handle (req)) < 0)
 {
 snprintf (error_msg, 128, "cci_close_req_handle
ERROR\n");
 }
conn_err:
 return error_msg;
}

```

## cci\_clob\_free

int cci\_clob\_free(T\_CCI\_CLOB clob)

**CLOB** 구조체에 대한 메모리를 해제한다.

매개변수:

- **clob** – (IN) **LOB** Locator

반환:

에러 코드(0: 성공)

- **CCI\_ER\_INVALID\_LOB\_HANDLE**

## cci\_clob\_new

int cci\_clob\_new(int conn\_handle, T\_CCI\_CLOB \*clob, T\_CCI\_ERROR \*error\_buf)

**LOB** 데이터가 저장될 빈 파일을 하나 생성하고, 해당 파일을 참조하는 Locator를 *clob* 구조체에 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

- **clob** – (OUT) **LOB** Locator
- **error\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(<0 : 에러)

- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_INVALID\_LOB\_HANDLE**
- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_clob\_read

```
int cci_clob_read(int conn_handle, T_CCI_CLOB clob, long start_pos, int length, char *buf,
T_CCI_ERROR *error_buf)
```

*clob* 에 명시한 **LOB** 데이터 파일의 *start\_pos* 부터 *length* 만큼 데이터를 읽어 *buf* 에 저장한 후 이를 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **clob** – (IN) **LOB** Locator
- **start\_pos** – (IN) **LOB** 데이터 파일의 위치 인덱스
- **length** – (IN) 파일로부터 가져올 **LOB** 데이터 길이
- **buf** – (IN) 데이터 읽기 버퍼
- **error\_buf** – (OUT) 에러 버퍼

반환:

read한 크기(>=0 : 성공), 에러 코드(<0 : 에러)

- CCI\_ER\_COMMUNICATION
- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_CONNECT
- CCI\_ER\_DBMS
- CCI\_ER\_INVALID\_LOB\_HANDLE
- CCI\_ER\_INVALID\_LOB\_READ\_POS
- CCI\_ER\_LOGIN\_TIMEOUT
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_QUERY\_TIMEOUT
- CCI\_ER\_USED\_CONNECTION

## cci\_clob\_size

long long cci\_clob\_size(T\_CCI\_CLOB \*clob)

*clob* 에 명시한 데이터 파일의 크기를 반환한다.

매개변수:

- **clob** – (IN) **LOB** Locator

반환:

**CLOB** 데이터 파일의 크기(>=0 : 성공), 에러 코드(<0 : 에러)

- CCI\_ER\_INVALID\_LOB\_HANDLE

## cci\_clob\_write

int cci\_clob\_write(int conn\_handle, T\_CCI\_CLOB clob, long start\_pos, int length, const char \*buf, T\_CCI\_ERROR \*error\_buf)

*buf*로부터 *length* 만큼 데이터를 읽어 *clob*에 명시한 **LOB** 데이터 파일의 *start\_pos*부터 저장한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **clob** – (IN) **LOB** Locator
- **start\_pos** – (IN) **LOB** 데이터 파일의 위치 인덱스
- **length** – (IN) 버퍼로부터 가져올 데이터 길이
- **buf** – (OUT) 데이터 쓰기 버퍼
- **error\_buf** – (OUT) 에러 버퍼

반환:

write한 크기(>=0 : 성공), 에러 코드(<0 : 에러)

- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_INVALID\_LOB\_HANDLE**
- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_QUERY\_TIMEOUT**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_close\_query\_result

int cci\_close\_query\_result(int req\_handle, T\_CCI\_ERROR \*err\_buf)

`cci_execute()`, `cci_execute_array()` 또는 `cci_execute_batch()` 함수가 반환한 resultset을 종료(close)한다. 요청 핸들(req\_handle)의 종료 없이 `cci_prepare()` 를 반복 수행하는 경우 `cci_close_req_handle()` 함수를 호출하기 전에 이 함수를 호출할 것을 권장한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드 (0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_RESULT\_SET\_CLOSED**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_close\_req\_handle

int cci\_close\_req\_handle(int req\_handle)

`cci_prepare()` 로 획득한 요청 핸들을 종료(close)한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들

반환:

에러 코드(0 : 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_REQ\_HANDLE**

- CCI\_ER\_COMMUNICATION
- CCI\_ER\_DBMS
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_USED\_CONNECTION

## cci\_col\_get

int cci\_col\_get(int conn\_handle, char \*oid\_str, char \*col\_attr, int \*col\_size, int \*col\_type, T\_CCI\_ERROR \*err\_buf)

collection type의 속성 값을 가져온다. 클래스 이름이 C이고 set\_attr의 domain이 set(multiset, sequence)인 경우 다음의 질의와 같다.

```
SELECT a FROM C, TABLE(set_attr) AS t(a) WHERE C = oid;
```

즉, 멤버 개수가 레코드 개수가 된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **col\_size** – (OUT) collection 크기 (-1 : null)
- **col\_type** – (OUT) collection 타입 (set, multiset, sequence : u\_type)
- **err\_buf** – (OUT) 에러 버퍼

반환:

## 요청 핸들

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

**cci\_col\_seq\_drop**

int cci\_col\_seq\_drop(int conn\_handle, char \*oid\_str, char \*col\_attr, int index, T\_CCI\_ERROR \*err\_buf)

sequence 속성 값에 index(base:1) 번째의 멤버를 drop시킨다. 다음은 seq 속성 값에서 첫 번째 값을 삭제하는 예이다.

```
cci_col_seq_drop(conn_handle, oid_str, seq_attr, 1, err_buf);
```

## 매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **index** – (IN) 인덱스
- **err\_buf** – (OUT) 에러 버퍼

## 반환:

## 에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_col\_seq\_insert

```
int cci_col_seq_insert(int conn_handle, char *oid_str, char *col_attr, int index, char *value,
T_CCI_ERROR *err_buf)
```

sequence 속성 값에서 index(base:1) 번째에 멤버를 추가시킨다. 다음은 seq 속성 값에서 1번에 값 'a'를 추가하는 예이다.

```
cci_col_seq_insert(conn_handle, oid_str, seq_attr, 1, "a",
err_buf);
```

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **index** – (IN) 인덱스
- **value** – (IN) 순차적 엘리먼트(스트링)
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_col\_seq\_put

```
int cci_col_seq_put(int conn_handle, char *oid_str, char *col_attr, int index, char *value,
T_CCI_ERROR *err_buf)
```

sequence 속성 값에 index(base:1) 번째의 멤버를 새로운 값으로 대체한다.. 다음은 seq 속성 값에서 1번 값을 'a'로 대체하는 예이다.



```
cci_col_seq_put(conn_handle, oid_str, seq_attr, 1, "a", err_buf);
```

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **index** – (IN) 인덱스
- **value** – (IN) 순차적 값
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_col\_set\_add

```
int cci_col_set_add(int conn_handle, char *oid_str, char *col_attr, char *value, T_CCI_ERROROR *err_buf)
```

set 속성 값에 member 하나를 추가한다. 다음은 set 속성 값에 'a'를 추가하는 예이다.

```
cci_col_set_add(conn_handle, oid_str, set_attr, "a", err_buf);
```

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름

- **value** – (IN) set 엘리먼트
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_col\_set\_drop

int cci\_col\_set\_drop(int conn\_handle, char \*oid\_str, char \*col\_attr, char \*value, T\_CCI\_ERROR \*err\_buf)

set 속성 값에서 멤버 하나를 drop시킨다. 다음은 set 속성 값에서 'a'를 삭제하는 예이다.

```
cci_col_set_drop(conn_handle, oid_str, set_attr, "a", err_buf);
```

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **value** – (IN) set 엘리먼트(스트링)
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_QUERY\_TIMEOUT**

- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_COMMUNICATION**

## cci\_col\_size

int cci\_col\_size(int conn\_handle, char \*oid\_str, char \*col\_attr, int \*col\_size, T\_CCI\_ERROR \*err\_buf)

set(seq) 속성의 개수를 가져온다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **col\_attr** – (IN) collection 속성 이름
- **col\_size** – (OUT) collection 크기 (-1 : NULL)
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0 : 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_connect

int cci\_connect(char \*ip, int port, char \*db\_name, char \*db\_user, char \*db\_password)

DB 서버에 대한 연결 핸들을 할당받고 해당 서버와 연결을 시도한다. 서버 연결에 성공하면 연결 핸들 ID를 반환하고, 실패하면 에러 코드를 반환한다.

매개변수:

- **ip** – (IN) 서버 IP 문자 스트링 (호스트 이름)

- **port** - (IN) 브로커 포트(**\$CUBRID/conf/cubrid\_broker.conf** 파일에 설정된 포트를 사용)
- **db\_name** - (IN) DB 이름
- **db\_user** - (IN) DB 사용자 이름
- **db\_passwd** - (IN) DB 사용자 암호

반환:

연결 핸들 ID(성공), 에러 코드(실패)

- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_HOSTNAME**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

## cci\_connect\_ex

```
int cci_connect_ex(char *ip, int port, char *db_name, char *db_user, char *db_password,
T_CCI_ERROR *err_buf)
```

**CCI\_ER\_DBMS** 에러를 반환하면 세부 에러 내용을 DB 에러 버퍼(*err\_buf*)를 통해 확인할 수 있다는 점만 `cci_connect()` 와 다르고 나머지는 동일하다.

매개변수:

- **ip** – (IN) 서버 IP 문자 스트링 (호스트 이름)
- **port** – (IN) 브로커 포트(**\$CUBRID/conf/cubrid\_broker.conf** 파일에 설정된 포트를 사용)
- **db\_name** – (IN) DB 이름
- **db\_user** – (IN) DB 사용자 이름
- **db\_passwd** – (IN) DB 사용자 암호
- **err\_buf** – (OUT) 에러 버퍼

반환:

연결 핸들 ID(성공), 에러 코드(실패)

- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_HOSTNAME**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

## cci\_connect\_with\_url

int cci\_connect\_with\_url(char \*url, char \*db\_user, char \*db\_password)

*url* 인자로 전달된 접속 정보를 이용하여 데이터베이스로 연결을 시도한다. CCI에서 브로커의 HA 기능을 사용하는 경우 이 함수의 *url* 인자 내의 altHosts 속성을 이용하여, 장애 발생 시 failover할 standby 브로커 서버의 연결 정보를 명시해야 한다. 서버 연결에 성공하면 연결 핸들 ID를 반환하고, 실패하면 에러 코드를 반환한다. 브로커의 HA 기능에 대한 자세한 내용은 [브로커 이중화](#)를 참고한다.

매개변수:

- **url** – (IN) 서버 연결 정보 문자 스트링

- **db\_user** - (IN) DB 사용자 이름. NULL이면 *url* 의 *<db\_user>* 가 사용된다. 이 값이 빈 문자열("")이거나 *url* 내의 *<db\_user>*가 정의되지 않은 경우 DB 사용자 이름은 **PUBLIC** 이 된다.
- **db\_passwd** - (IN) DB 사용자 암호. NULL이면 *url* 의 *<db\_password>*가 사용된다. *url* 내의 *<db\_password>*가 정의되지 않은 경우 암호는 빈 문자열("")이 된다.

반환:

연결 핸들 ID(성공), 에러 코드(실패)

- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_HOSTNAME**
- **CCI\_ER\_INVALID\_URL**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_LOGIN\_TIMEOUT**

```
<url> ::=
cci:CUBRID:<host>:<port>:<db_name>:<db_user>:<db_password>:[?
<properties>]

<properties> ::= <property> [&<property>]
<property> ::= altHosts=<alternative_hosts> [&rcTime=<time>]
[&loadBalance=true|false]
 |{login_timeout|loginTimeout}=<milli_sec>
 |{query_timeout|queryTimeout}=<milli_sec>
 |{disconnect_on_query_timeout|
disconnectOnQueryTimeout}=true|false
 | logFile=<file_name>
 | logBaseDir=<dir_name>
 | logSlowQueries=true|
false[&slowQueryThresholdMillis=<milli_sec>]
 | logTraceApi=true|false
 | logTraceNetwork=true|false
 | useSSL=true|false
```

```
<alternative_hosts> ::= <host>:<port> [,<host>:<port>]

<host> := HOSTNAME | IP_ADDR
<time> := SECOND
<milli_sec> := MILLI SECOND
```

연결 대상과 관련된 속성은 **altHosts** 이며, 타임아웃과 관련된 속성은 **loginTimeout**, **queryTimeout**, **disconnectOnQueryTimeout** 이다. 디버깅용 로그 정보 설정과 관련된 속성은 **logSlowQueries**, **logTraceApi**, **logTraceNetwork** 이다. *url* 인자에 입력하는 모든 속성 (property) 이름은 대소문자 구별을 하지 않는다.

- *host*: 마스터 데이터베이스의 호스트 이름 또는 IP 주소
- *port*: 포트 번호
- *db\_name*: 데이터베이스 이름
- *db\_user*: 데이터베이스 사용자 이름
- *db\_password*: 데이터베이스 사용자 암호. *url* 내의 암호에는 ':'를 포함할 수 없다.
- **altHosts** = *standby\_broker1\_host*, *standby\_broker2\_host*, ...: active 서버에 연결할 수 없는 경우, 그 다음으로 연결을 시도(failover)할 standby 서버의 브로커 정보를 나타낸다. failover 할 브로커를 여러 개 지정할 수 있고, **altHosts** 에 나열한 순서대로 연결을 시도한다.

#### 참고

메인 호스트와 **altHosts** 브로커들의 **ACCESS\_MODE** 설정에 **RW**와 **RO**가 섞여 있다 하더라도, 응용 프로그램은 **ACCESS\_MODE**와 무관하게 접속 대상 호스트를 결정한다. 따라서 사용자는 접속 대상 브로커의 **ACCESS\_MODE**를 감안해서 메인 호스트와 **altHosts**를 정해야 한다.

- **rcTime**: 첫 번째로 접속했던 브로커에 장애가 발생한 이후 **altHosts** 에 명시한 브로커로 접속한다(브로커 failover). 이후, **rcTime** 만큼 시간이 경과할 때마다 원래의 브로커에 재접속을 시도한다(기본값 600초).
- **loadBalance**: 이 값이 true면 응용 프로그램이 메인 호스트와 **altHosts**에 지정한 호스트들에 랜덤한 순서로 연결한다(기본값: false)
- **login\_timeout** | **loginTimeout**: 데이터베이스에 로그인 시 타임아웃 값 (단위: msec). 이 시간을 초과하면 **CCI\_ER\_LOGIN\_TIMEOUT** (-38) 에러를 반환한다. 기본값은 **30,000**(30초)이다. 이 값이 0인 경우 무한 대기를 의미한다. 이 값은 최초 접속 이후 내부적인 재접속이 발생하는 경우에도 적용된다.

- **query\_timeout | queryTimeout**: `cci_prepare()`, `cci_execute()` 등의 함수를 호출했을 때 이 값으로 설정한 시간이 지나면 서버로 보낸 질의 요청에 대한 취소 메시지를 보내고 호출된 함수는 **CCI\_ER\_QUERY\_TIMEOUT** (-39) 에러를 반환한다. 기본값은 0이며, 0인 경우 무한 대기를 의미한다. 질의를 수행한 함수에서 타임아웃 발생 시 함수의 반환 값은 **disconnect\_on\_query\_timeout**의 설정에 따라 달라질 수 있다. 자세한 내용은 다음의 **disconnect\_on\_query\_timeout**을 참고한다.

#### 참고

`cci_execute()` 에 `CCI_EXEC_QUERY_ALL` 플래그를 설정하거나 `cci_execute_batch()` 또는 `cci_execute_array()` 를 사용하여 여러 개의 질의를 한 번에 실행하는 경우, 질의 타임아웃은 질의 하나에 대해 적용되는 것이 아니라 함수 하나에 대해 적용된다. 즉, 함수 시작 이후 타임아웃이 발생하면 함수 수행이 중단된다.

- **disconnect\_on\_query\_timeout | disconnectOnQueryTimeout** : 질의 요청 타임아웃 시 즉시 소켓 연결 종료 여부. `cci_prepare()`, `cci_execute()` 등의 함수를 호출했을 때 **query\_timeout** 으로 설정한 시간이 지나면 질의 취소 요청 후 즉시 소켓 연결을 종료할 것인지, 아니면 질의 취소를 받아들이는다는 서버의 응답을 기다릴 것인지를 설정한다. 기본값은 **false** 로, 서버의 응답을 기다린다. 이 값이 **true** 이면 `cci_prepare()`, `cci_execute()` 등의 함수 호출 도중 질의 타임아웃이 발생할 때 서버에 질의 취소 메시지를 보낸 후, 소켓을 닫고 **CCI\_ER\_QUERY\_TIMEOUT** (-39) 에러를 반환한다. (브로커가 아닌 데이터베이스 서버 쪽에서 에러가 발생한 경우 -1을 반환한다. 상세 에러를 확인하고 싶으면 “데이터베이스 에러 버퍼”의 에러 코드를 확인한다. 데이터베이스 에러 버퍼에서 에러 코드를 확인하는 방법은 **CCI 에러 코드와 에러 메시지** 를 참고한다.) 응용 프로그램이 질의 취소 메시지를 보낸 후 에러를 반환했음에도 불구하고, 데이터베이스 서버는 그 메시지를 받지 못하고 해당 질의를 수행할 수 있음을 주의한다. **false** 이면 서버에 취소 메시지를 보낸 후, 서버의 질의 요청에 대한 응답이 올 때 까지 대기한다.
- **logFile**: 디버깅용 로그 파일 이름(기본값: `cci_<handle_id>.log`). `<handle_id>`는 이 함수가 반환하는 연결 핸들 ID이다.
- **logBaseDir**: 디버깅용 로그 파일이 생성되는 디렉터리. 경로를 포함한 파일 이름의 형식은 `logBaseDir/logFile`이 되며, 상대 경로로 지정할 수 있다.
- **logSlowQueries**: 디버깅용 슬로우 쿼리 로깅 여부(기본값: **false**)
- **slowQueryThresholdMillis**: 디버깅용 슬로우 쿼리 로깅 시 슬로우 쿼리 제한 시간(기본값: **60000**). 단위는 밀리 초이다.
- **logTraceApi**: CCI 함수 시작과 끝의 로깅 여부
- **logTraceNetwork**: CCI 함수 네트워크 데이터 전송 내용의 로깅 여부



- **useSSL**: 패킷 암호화 여부 (기본값: false)
  - 패킷 암호화: useSSL = true
  - 일반 평문: useSSL = false

## 예제

```
--connection URL string when a property(altHosts) is specified
for HA
URL=cci:CUBRID:192.168.0.1:33000:demodb:::?
altHosts=192.168.0.2:33000,192.168.0.3:33000

--connection URL string when properties(altHosts,rcTime) is
specified for HA
URL=cci:CUBRID:192.168.0.1:33000:demodb:::?
altHosts=192.168.0.2:33000,192.168.0.3:33000&rcTime=600

--connection URL string when
properties(logSlowQueries,slowQueryThresholdMills, logTraceApi,
logTraceNetwork) are specified for interface debugging
URL = "cci:cubrid:192.168.0.1:33000:demodb:::?
logSlowQueries=true&slowQueryThresholdMills=1000&logTraceApi=tru
e&logTraceNetwork=true"

--connection URL string when useSSL property specified for
encrypted connection
URL = "cci:cubrid:192.168.0.1:33000:demodb:::?useSSL=true"
```

### ⚠ 경고

- useSSL의 flag는 **브로커 모드와 일치해야 한다**. 아래와 같이 브로커의 암호화 모드와 다른 flag로 접속을 요청하는 경우 **연결되지 않는다**.
  - useSSL=true, 브로커 '일반 모드' 일 때 연결 불가 ( **cubrid\_broker.conf**: SSL = OFF)
  - useSSL=false, 브로커 '암호화 모드' 일때 연결 불가 (**cubrid\_broker.conf**: SSL = ON)

## cci\_connect\_with\_url\_ex

```
int cci_connect_with_url_ex(char *url, char *db_user, char *db_password, T_CCI_ERROR *err_buf)
```

**CCI\_ER\_DBMS** 에러를 반환하면 세부 에러 내용을 데이터베이스 에러 버퍼(*err\_buf*)를 통해 확인할 수 있다는 점만 `cci_connect_with_url()` 과 다르고 나머지는 동일하다.

매개변수:

- **err\_buf** – (OUT) 에러 버퍼

## cci\_cursor

`int cci_cursor(int req_handle, int offset, T_CCI_CURSOR_POS origin, T_CCI_ERROR *err_buf)`

`cci_execute()` 로 실행한 질의 결과 내의 특정 레코드에 접근하기 위하여 요청 핸들에 설정된 커서를 이동시킨다. 인자로 지정되는 *origin* 변수 값과 *offset* 값을 통해 커서의 위치가 이동되며, 이동할 커서의 위치가 유효하지 않을 경우 **CCI\_ER\_NO\_MORE\_DATA** 를 반환한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **offset** – (IN) 이동할 오프셋
- **origin** – (IN) 커서 위치를 나타내는 변수로서, 타입은 **T\_CCI\_CURSOR\_POS** 이다. **T\_CCI\_CURSOR\_POS** enum은 **CCI\_CURSOR\_FIRST**, **CCI\_CURSOR\_CURRENT**, **CCI\_CURSOR\_LAST** 의 세 가지 값으로 구성된다.
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드 (0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_NO\_MORE\_DATA**
- **CCI\_ER\_COMMUNICATION**

예제

```
//the cursor moves to the first record
cci_cursor(req, 1, CCI_CURSOR_FIRST, &err_buf);

//the cursor moves to the next record
cci_cursor(req, 1, CCI_CURSOR_CURRENT, &err_buf);
```

```
//the cursor moves to the last record
cci_cursor(req, 1, CCI_CURSOR_LAST, &err_buf);

//the cursor moves to the previous record
cci_cursor(req, -1, CCI_CURSOR_CURRENT, &err_buf);
```

## cci\_cursor\_update

int cci\_cursor\_update(int req\_handle, int cursor\_pos, int index, T\_CCI\_A\_TYPE a\_type, void \*value, T\_CCI\_ERROR \*err\_buf)

*cursor\_pos* 의 커서 위치에 대해서 *index* 번째의 칼럼 값을 *value* 값으로 update한다. 데이터 베이스에 **NULL** 로 update할 경우 *value* 를 **NULL** 로 한다. update할 수 있는 조건은 `cci_prepare()` 를 참조한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **cursor\_pos** – (IN) 커서 위치
- **index** – (IN) 칼럼 인덱스
- **a\_type** – (IN) *value* 타입
- **value** – (IN) 새로운 값
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드 (0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_TYPE\_CONVERSION**
- **CCI\_ER\_ATYPE**

*a\_type* 에 대한 *value* 의 데이터 타입은 다음과 같다.

| <b>a_type</b>            | <b>value 타입</b>                    |
|--------------------------|------------------------------------|
| <b>CCI_A_TYPE_STR</b>    | char *                             |
| <b>CCI_A_TYPE_INT</b>    | int *                              |
| <b>CCI_A_TYPE_FLOAT</b>  | float *                            |
| <b>CCI_A_TYPE_DOUBLE</b> | double *                           |
| <b>CCI_A_TYPE_BIT</b>    | <b>T_CCI_BIT *</b>                 |
| <b>CCI_A_TYPE_SET</b>    | <b>T_CCI_SET</b>                   |
| <b>CCI_A_TYPE_DATE</b>   | <b>T_CCI_DATE *</b>                |
| <b>CCI_A_TYPE_BIGINT</b> | int64_t * (For Windows: __int64 *) |
| <b>CCI_A_TYPE_BLOB</b>   | <b>T_CCI_BLOB</b>                  |
| <b>CCI_A_TYPE_CLOB</b>   | <b>T_CCI_CLOB</b>                  |

### cci\_datasource\_borrow

T\_CCI\_CONN cci\_datasource\_borrow(T\_CCI\_DATASOURCE \*datasource, T\_CCI\_ERROR \*err\_buf)  
**T\_CCI\_DATASOURCE** 구조체에서 사용할 CCI 연결을 획득한다.

매개변수:

- **datasource** – (IN) CCI 연결을 획득할 **T\_CCI\_DATASOURCE** 구조체 포인터
- **err\_buf** – (OUT) 에러 버퍼 (에러가 발생하면 에러 코드와 메시지를 반환)

반환:

CCI 연결 핸들 식별자 (성공), -1 (실패)

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_release(),
cci_datasource_change_property()
```

## cci\_datasource\_change\_property

int cci\_datasource\_change\_property(T\_CCI\_DATASOURCE \*datasource, const char \*key, const char \*val)

DATASOURCE의 속성(property) 이름은 *key*\*에 명시하고, 값을 *\*val*에 설정한다. 이 함수를 사용하여 변경한 속성 값은 *datasource* 내 모든 연결에 적용된다.

매개변수:

- **datasource** – (IN) CCI 연결을 획득할 T\_CCI\_DATASOURCE 구조체 포인터
- **key** – (IN) 속성 이름 문자열에 대한 포인터
- **val** – (IN) 속성 값 문자열에 대한 포인터

반환:

에러 코드(0: 성공)

- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_NO\_PROPERTY**
- **CCI\_ER\_PROPERTY\_TYPE**

변경 가능한 속성(property)의 이름 및 값은 다음과 같다.

| 속성 이름                | 타입     | 값                                         | 의미                                                                                                                  |
|----------------------|--------|-------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| default_autocommit   | bool   | true/false                                | autocommit 여부.<br>기본값은<br>cubrid_broker.conf<br>의<br>CCI_DEFAULT_AUTO<br>COMMIT이며, 이 값<br>의 기본값은<br>ON(true)임.      |
| default_lock_timeout | msec   | 숫자                                        | lock timeout                                                                                                        |
| default_isolation    | string | <code>cci_property_set()</code><br>의 표 참고 | isolation level. 기<br>본값은 cubrid.conf<br>의 isolation_level<br>이며, 이 값의 기본<br>값은<br>"READ_COMMITTED"<br>임.           |
| login_timeout        | msec   | 숫자                                        | login timeout. 기본<br>값은 0(무한대기)<br>임. prepare 또는<br>execute 함수 호출<br>시 내부적으로 재접<br>속이 발생할 수 있<br>으며, 이 때에도 사<br>용됨. |

### 예제

```

...
ps = cci_property_create ();
...
ds = cci_datasource_create (ps, &err);
...
cci_datasource_change_property(ds, "login_timeout", "5000");
cci_datasource_change_property(ds, "default_lock_timeout",

```

```
"2000");
cci_datasource_change_property(ds, "default_isolation",
"TRAN_REP_CLASS_COMMIT_INSTANCE");
cci_datasource_change_property(ds, "default_autocommit", "true");
...
```

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_borrow(), cci_datasource_destroy(),
cci_datasource_release()
```

## cci\_datasource\_create

T\_CCI\_DATASOURCE \*cci\_datasource\_create(T\_CCI\_PROPERTIES \*properties, T\_CCI\_ERROR \*err\_buf)

CCI의 DATASOURCE를 생성한다.

매개변수:

- **properties** – (IN) 설정이 저장된 **T\_CCI\_PROPERTIES** 구조체 포인터. `cci_property_set()` 으로 속성 값들을 설정한다.
- **err\_buf** – (OUT) 에러 버퍼 (에러가 발생하면 에러 코드와 메시지를 반환)

반환:

생성된 **T\_CCI\_DATASOURCE** 구조체 포인터 (성공), NULL (실패)

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_borrow(), cci_datasource_destroy(), cci_datasource_release(),
cci_datasource_change_property()
```

## cci\_datasource\_destroy

void cci\_datasource\_destroy(T\_CCI\_DATASOURCE \*datasource)

CCI의 DATASOURCE를 삭제한다.

매개변수:

- **datasource** – (IN) 삭제할 **T\_CCI\_DATASOURCE** 구조체 포인터

반환:

void

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_borrow(), cci_datasource_release(),
cci_datasource_change_property()
```

## cci\_datasource\_release

int cci\_datasource\_release(T\_CCI\_DATASOURCE \*datasource, T\_CCI\_CONN conn, T\_CCI\_ERROR \*err\_buf)

**T\_CCI\_DATASOURCE** 구조체에 사용을 끝낸 CCI 연결을 반환한다. 연결이 연결 풀에 반환된 이후 재사용하려면 반드시 `cci_datasource_borrow()` 함수를 재호출해야 한다.

매개변수:

- **datasource** – (IN) CCI 연결을 반환할 **T\_CCI\_DATASOURCE** 구조체 포인터
- **conn** – (IN) 사용을 끝낸 CCI 연결의 핸들 식별자
- **err\_buf** – (OUT) 에러 버퍼 (에러가 발생하면 에러 코드와 메시지를 반환)

반환:

1 (성공), 0 (실패)



### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_borrow(),
cci_datasource_change_property()
```

## cci\_disconnect

int cci\_disconnect(int conn\_handle, T\_CCI\_ERROR \*err\_buf)

*conn\_handle*에 대해 생성된 모든 요청 핸들을 삭제한다. 트랜잭션이 진행 중일 경우 `cci_end_tran()`을 실행한 다음 삭제된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0 : 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**

## cci\_end\_tran

int cci\_end\_tran(int conn\_handle, char type, T\_CCI\_ERROR \*err\_buf)

현재 진행 중인 트랜잭션에 대해서 커밋(commit)이나 롤백(rollback)을 수행한다. 이때, 열려 있는 요청 핸들은 모두 종료되고, 데이터베이스 서버와 연결이 해제된다. 단, 서버와 연결이 끊어진 후에도 해당 연결 핸들은 유효하며, 이는 `cci_connect()` 함수로 연결 핸들을 하나 할당 받은 경우와 동일한 상태다. *type* 이 **CCI\_TRAN\_COMMIT** 으로 지정되면 트랜잭션을 커밋하고, **CCI\_TRAN\_ROLLBACK** 으로 지정되면 트랜잭션을 롤백한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

- **type** – (IN) **CCI\_TRAN\_COMMIT** 또는 **CCI\_TRAN\_ROLLBACK**
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0 : 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_TRAN\_TYPE**

브로커 파라미터인 **CCI\_DEFAULT\_AUTOCOMMIT**으로 응용 프로그램 시작 시 자동 커밋 모드  
기본값을 설정할 수 있으며, 브로커 파라미터 설정을 생략하면 기본값은 **ON**이다. 응용 프로그  
램 내에서 자동 커밋 모드를 변경하려면 `cci_set_autocommit()` 함수를 이용하며, 자동 커밋 모드  
가 **OFF** 이면 `cci_end_tran()` 함수를 이용하여 명시적으로 트랜잭션을 커밋하거나 롤백해야 한  
다.

## cci\_escape\_string

long cci\_escape\_string(int conn\_handle, char \*to, const char \*from, unsigned long length,  
T\_CCI\_ERROR \*err\_buf)

입력 문자열을 CUBRID 질의문에서 사용할 수 있는 문자열로 변환한다. 이 함수의 인자로 연결  
핸들 또는 **no\_backslash\_escapes** 설정 값, 출력 문자열 포인터, 입력 문자열 포인터, 입력 문  
자열의 바이트 길이, 오류 정보를 담을 **T\_CCI\_ERROR** 구조체 변수의 주소가 지정된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들 또는 **no\_backslash\_escapes**  
설정 값. 연결 핸들이 주어지는 경우, 연결된 서버의  
**no\_backslash\_escapes** 파라미터 설정 값을 읽어서 변환 방  
법을 결정한다. 연결 핸들 대신  
**CCI\_NO\_BACKSLASH\_ESCAPES\_TRUE** 또는  
**CCI\_NO\_BACKSLASH\_ESCAPES\_FALSE** 설정 값을 전달하여  
변환 방법을 결정할 수 있다.
- **to** – (OUT) 결과 문자열
- **from** – (IN) 입력 문자열
- **length** – (IN) 입력 문자열의 최대 바이트 길이

- **err\_buf** – (OUT) 에러 버퍼

반환:

변경된 문자열의 바이트 길이(성공), 에러 코드(실패)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_COMMUNICATION**

시스템 파라미터 **no\_backslash\_escapes**의 값이 **yes**(기본값)이거나 연결 핸들 위치에 **CCI\_NO\_BACKSLASH\_ESCAPES\_TRUE** 값을 전달하는 경우, 변환되는 문자는 다음과 같다.

- ' (single quote) => \ ' (escaped single quote)

시스템 파라미터 **no\_backslash\_escapes**의 값이 **no**이거나 연결 핸들 위치에 **CCI\_NO\_BACKSLASH\_ESCAPES\_FALSE** 값을 전달하는 경우, 변환되는 문자는 다음과 같다.

- \n (new line character, ASCII 10) => \ + n (백슬래시 + 알파벳 n)
- \r (carriage return, ASCII 13) => \ + r (백슬래시 + 알파벳 r)
- \0 (ASCII 0) => \ + 0 (백슬래시 + 0(ASCII 48))
- \ (백슬래시) => \ + \

결과 문자열을 저장할 공간은 *length* 인자로 사용자가 직접 할당하며, 최대 입력 문자열의 바이트 길이 \* 2 + 1만큼 필요할 수 있다.

## cci\_execute

int cci\_execute(int req\_handle, char flag, int max\_col\_size, T\_CCI\_ERROR \*err\_buf)

**cci\_prepare()**를 수행한 SQL 문(prepared statement)을 실행한다. 이 함수의 인자로 요청 핸들, *flag*, fetch하는 칼럼의 문자열 최대 길이, 오류 정보를 담을 **T\_CCI\_ERROR** 구조체 변수의 주소가 지정된다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **flag** – (IN) exec flag ( **CCI\_EXEC\_QUERY\_ALL** )
- **max\_col\_size** – (IN) 문자열 타입인 경우 fetch하는 칼럼의 문자열 최대 길이(단위: 바이트). 이 값이 0이면 전체 길이를 fetch한다.

- **err\_buf** – (OUT) 에러 버퍼

반환:

- **SELECT** : 결과 행의 개수를 반환
- **INSERT, UPDATE** : 반영된 행의 개수
- 기타 질의 : 0
- 실패 : 에러 코드
  - **CCI\_ER\_REQ\_HANDLE**
  - **CCI\_ER\_BIND**
  - **CCI\_ER\_DBMS**
  - **CCI\_ER\_COMMUNICATION**
  - **CCI\_ER\_QUERY\_TIMEOUT**
  - **CCI\_ER\_LOGIN\_TIMEOUT**

*flag*를 통해 질의문 수행 방식을 모두 수행하게 하거나 첫번째 질의문만 수행하도록 지정할 수 있다.

#### 참고

2008 R4.4와 9.2 이상 버전에서 *flag* 설정 시 비동기 방식으로 결과를 가져오게 하는 **CCI\_EXEC\_ASYNC**를 더 이상 지원하지 않는다.

*flag*에 **CCI\_EXEC\_QUERY\_ALL**을 설정하면 prepare 시에 전달된 여러 개의 질의문(세미콜론으로 여러 개의 질의문을 구분)을 모두 수행하며, 그렇지 않은 경우 제일 앞에 있는 질의문만 수행한다.

*flag*에 **CCI\_EXEC\_QUERY\_ALL**을 설정하면 다음의 규칙이 적용된다.

- 리턴 값은 첫 번째 질의에 대한 결과이다.
- 어느 하나의 질의에서 에러가 발생할 경우 execute는 실패한 것으로 처리된다.
- q1; q2; q3와 같이 구성된 질의에 대해서 q1을 성공하고 q2에서 에러가 발생해도 q1의 수행 결과는 유효하다. 즉, 에러가 발생했을 때, 앞서 성공한 질의 수행에 대해서 롤백하지 않는다.

- 질의가 성공적으로 수행된 경우 두 번째 질의에 대한 결과는 `cci_next_result()` 를 통해서 얻을 수 있다.

`max_col_size` 는 prepared statement의 칼럼이 **CHAR, VARCHAR, BIT, VARBIT** 일 경우 클라이언트로 전송되는 칼럼의 문자열 최대 길이를 결정하기 위한 값이며, 이 값이 0이면 전체 길이를 fetch한다.

## cci\_execute\_array

int cci\_execute\_array(int req\_handle, T\_CCI\_QUERY\_RESULT \*\*query\_result, T\_CCI\_ERROR \*err\_buf)

prepared statement에 하나 이상의 값이 바인딩되는 경우, 바인딩되는 변수의 값을 배열(array)로 전달받아 각각의 값을 변수에 바인딩하여 질의를 실행한다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **query\_result** – (OUT) 질의 결과
- **err\_buf** – (OUT) 데이터베이스 에러 버퍼

반환:

- 성공 : 수행된 질의의 개수(질의 수행 성공/실패 여부와 상관없음)
- 실패 : 에러 코드
  - **CCI\_ER\_REQ\_HANDLE**
  - **CCI\_ER\_BIND**
  - **CCI\_ER\_DBMS**
  - **CCI\_ER\_COMMUNICATION**
  - **CCI\_ER\_QUERY\_TIMEOUT**
  - **CCI\_ER\_LOGIN\_TIMEOUT**

데이터를 바인딩하기 위해서는 `cci_bind_param_array_size()` 함수를 호출하여 배열의 크기를 지정한 후, `cci_bind_param_array()` 함수를 이용하여 각각의 값을 변수에 바인딩하고, `cci_execute_array()` 함수를 호출하여 질의를 실행한다. 질의 결과는 T\_CCI\_QUERY\_RESULT 구조체의 배열에 저장된다.

`cci_execute_array()` 함수는 `query_result` 변수에 질의 결과를 반환한다. 실행 결과에 대한 정보를 얻기 위해서는 아래와 같은 매크로를 이용할 수 있다. 매크로에서는 입력받는 각 인자에 대한 유효성 검사가 이루어지지 않으므로 주의한다.

`query_result` 변수의 사용이 끝나면 `cci_query_result_free()` 함수를 이용하여 질의 결과를 삭제해야 한다.

| 매크로                                     | 리턴 타입                                     | 의미                                               |
|-----------------------------------------|-------------------------------------------|--------------------------------------------------|
| <code>CCI_QUERY_RESULT_RESULT</code>    | int                                       | 영향을 끼친 행의 개수 또는 에러 식별자 (-1: CAS 에러, -2: DBMS 에러) |
| <code>CCI_QUERY_RESULT_ERR_NO</code>    | int                                       | 질의에 대한 에러 번호                                     |
| <code>CCI_QUERY_RESULT_ERR_MSG</code>   | char *                                    | 질의에 대한 에러 메시지                                    |
| <code>CCI_QUERY_RESULT_STMT_TYPE</code> | int( <code>T_CCI_CUBRID_STMT</code> enum) | 질의문의 타입                                          |

자동 커밋이 ON인 경우 배열 내의 각 질의가 수행될 때마다 커밋된다.

#### 참고

- 2008 R4.3 미만 버전에서 자동 커밋이 ON인 경우 배열 내의 모든 질의가 수행된 이후에 커밋되었으나, 2008 R4.3부터는 질의 하나가 수행될 때마다 커밋된다.
- 자동 커밋이 OFF일 때 질의문을 일괄 처리하는 `cci_execute_array` 함수에서 배열 내의 질의 일부에 일반적인 오류가 발생하는 경우, 이를 건너뛰고 다음 질의를 계속 수행한다. 그러나, 교착 상태가 발생하면 트랜잭션을 롤백하고 오류 처리한다.

```
char *query =
 "update participant set gold = ? where host_year = ? and
 nation_code = 'KOR'";
int gold[2];
char *host_year[2];
int null_ind[2];
T_CCI_QUERY_RESULT *result;
int n_executed;
```

```

...

req = cci_prepare (con, query, 0, &cci_error);
if (req < 0)
{
 printf ("prepare error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}

gold[0] = 20;
host_year[0] = "2004";

gold[1] = 15;
host_year[1] = "2008";

null_ind[0] = null_ind[1] = 0;
error = cci_bind_param_array_size (req, 2);
if (error < 0)
{
 printf ("bind_param_array_size error: %d\n", error);
 goto handle_error;
}

error =
 cci_bind_param_array (req, 1, CCI_A_TYPE_INT, gold,
null_ind, CCI_U_TYPE_INT);
if (error < 0)
{
 printf ("bind_param_array error: %d\n", error);
 goto handle_error;
}

error =
 cci_bind_param_array (req, 2, CCI_A_TYPE_STR, host_year,
null_ind, CCI_U_TYPE_INT);
if (error < 0)
{
 printf ("bind_param_array error: %d\n", error);
 goto handle_error;
}

n_executed = cci_execute_array (req, &result, &cci_error);
if (n_executed < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}
for (i = 1; i <= n_executed; i++)
{
 printf ("query %d\n", i);
 printf ("result count = %d\n", CCI_QUERY_RESULT_RESULT

```

```

(result, i));
 printf ("error message = %s\n", CCI_QUERY_RESULT_ERR_MSG
(result, i));
 printf ("statement type = %d\n",
 CCI_QUERY_RESULT_STMT_TYPE (result, i));
}
error = cci_query_result_free (result, n_executed);
if (error < 0)
{
 printf ("query_result_free: %d\n", error);
 goto handle_error;
}
error = cci_end_tran(con, CCI_TRAN_COMMIT, &cci_error);
if (error < 0)
{
 printf ("end_tran: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}

```

## cci\_execute\_batch

int cci\_execute\_batch(int conn\_handle, int num\_sql\_stmt, char \*\*sql\_stmt, T\_CCI\_QUERY\_RESULT \*\*query\_result, T\_CCI\_ERROR \*err\_buf)

CCI에서 **INSERT** / **UPDATE** / **DELETE** 와 같은 DML 질의를 사용하는 경우에는 여러 작업을 한 번에 처리할 수 있는데, 이러한 배치 작업을 위해서 `cci_execute_array()` 함수와 `cci_execute_batch()` 함수가 이용될 수 있다. 단, `cci_execute_batch()` 함수에서는 prepared statement를 사용할 수 없다. 질의 결과는 **T\_CCI\_QUERY\_RESULT** 구조체의 배열에 저장된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **num\_sql\_stmt** – (IN) *sql\_stmt* 의 개수
- **sql\_stmt** – (IN) SQL 문 array
- **query\_result** – (OUT) *sql\_stmt* 의 결과
- **err\_buf** – (OUT) 데이터베이스 에러 버퍼

반환:

- 성공 : 수행된 질의의 개수(질의 수행 성공/실패 여부와 상관없음)



- 실패 : 에러 코드
  - CCI\_ER\_CON\_HANDLE
  - CCI\_ER\_DBMS
  - CCI\_ER\_COMMUNICATION
  - CCI\_ER\_NO\_MORE\_MEMORY
  - CCI\_ER\_CONNECT
  - CCI\_ER\_QUERY\_TIMEOUT
  - CCI\_ER\_LOGIN\_TIMEOUT

인자로 지정된 *num\_sql\_stmt* 개의 *sql\_stmt* 를 수행하며, *query\_result* 변수로 수행된 질의 개수를 반환한다. 각각의 질의에 대한 결과는 `CCI_QUERY_RESULT_RESULT`, `CCI_QUERY_RESULT_ERR_NO`, `CCI_QUERY_RESULT_ERR_MSG`, `CCI_QUERY_RESULT_STMT_TYPE` 매크로를 이용할 수 있다. 전체 매크로에 대한 요약 설명은 `cci_execute_array()` 를 참고한다.

매크로에서는 입력받은 인자에 대한 유효성을 검사하지 않으므로 주의한다.

*query\_result* 변수의 사용이 끝나면 `cci_query_result_free()` 함수를 이용하여 질의 결과를 삭제해야 한다.

자동 커밋이 ON인 경우 배열 내의 각 질의가 수행될 때마다 커밋된다.

#### 참고

- 2008 R4.3 이전 버전에서 자동 커밋이 ON인 경우 배열 내의 모든 질의가 수행된 이후에 커밋되었으나, 2008 R4.3부터는 질의 하나가 수행될 때마다 커밋된다.
- 자동 커밋이 OFF일 때 질의문을 일괄 처리하는 `cci_execute_batch` 함수에서 배열 내의 질의 일부에 일반적인 오류가 발생하는 경우, 이를 건너뛰고 다음 질의를 계속 수행한다. 그러나, 교착 상태가 발생하면 트랜잭션을 롤백하고 오류 처리한다.

```
...
char **queries;
T_CCI_QUERY_RESULT *result;
int n_queries, n_executed;
...
count = 3;
queries = (char **) malloc (count * sizeof (char *));
queries[0] =
 "insert into athlete(name, gender, nation_code, event)
```

```

values('Ji-sung Park', 'M', 'KOR', 'Soccer');
queries[1] =
 "insert into athlete(name, gender, nation_code, event)
values('Joo-young Park', 'M', 'KOR', 'Soccer')";
queries[2] =
 "select * from athlete order by code desc limit 2";

//calling cci_execute_batch()
n_executed = cci_execute_batch (con, count, queries, &result,
&cci_error);
if (n_executed < 0)
{
 printf ("execute_batch: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}
printf ("%d statements were executed.\n", n_executed);

for (i = 1; i <= n_executed; i++)
{
 printf ("query %d\n", i);
 printf ("result count = %d\n", CCI_QUERY_RESULT_RESULT
(result, i));
 printf ("error message = %s\n", CCI_QUERY_RESULT_ERR_MSG
(result, i));
 printf ("statement type = %d\n",
 CCI_QUERY_RESULT_STMT_TYPE (result, i));
}

error = cci_query_result_free (result, n_executed);
if (error < 0)
{
 printf ("query_result_free: %d\n", error);
 goto handle_error;
}
...

```

## cci\_execute\_result

int cci\_execute\_result(int req\_handle, T\_CCI\_QUERY\_RESULT \*\*query\_result, T\_CCI\_ERROR \*err\_buf)

질의가 여러 개인 경우 수행 결과(statement type, result count)를 **T\_CCI\_QUERY\_RESULT** 구조체의 배열에 저장한다. 각각의 질의에 대한 결과는 **CCI\_QUERY\_RESULT\_RESULT**, **CCI\_QUERY\_RESULT\_ERR\_NO**, **CCI\_QUERY\_RESULT\_ERR\_MSG**, **CCI\_QUERY\_RESULT\_STMT\_TYPE** 매크로를 이용할 수 있다. 전체 매크로에 대한 요약 설명은 **cci\_execute\_array()** 를 참고한다.

매크로에서는 입력받은 인자에 대한 유효성을 검사하지 않으므로 주의한다.

사용된 질의 결과의 메모리는 `cci_query_result_free()` 를 통해 해제되어야 한다.

매개변수:

- **req\_handle** – (IN) prepared statement의 요청 핸들
- **query\_result** – (OUT) 쿼리 결과
- **err\_buf** – (OUT) 에러 버퍼

반환:

수행된 질의의 개수(성공), 에러 코드(실패)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_COMMUNICATION**

```
...
T_CCI_QUERY_RESULT *qr;
...

cci_execute(...);
res = cci_execute_result(req_h, &qr, &err_buf);
if (res < 0)
{
 /* error */
}
else
{
 for (i=1 ; i <= res ; i++)
 {
 result_count = CCI_QUERY_RESULT_RESULT(qr, i);
 stmt_type = CCI_QUERY_RESULT_STMT_TYPE(qr, i);
 }
 cci_query_result_free(qr, res);
}
...
```

## cci\_fetch

`int cci_fetch(int req_handle, T_CCI_ERROR *err_buf)`

`cci_execute()` 로 실행한 질의 결과를 서버 측 CAS로부터 fetch하여 클라이언트 버퍼에 저장한다. fetch된 질의 결과에서 특정 칼럼의 데이터는 `cci_get_data()` 함수를 이용해서 확인할 수 있다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CAS\_ER\_HOLDABLE\_NOT\_ALLOWED**
- **CCI\_ER\_NO\_MORE\_DATA**
- **CCI\_ER\_RESULT\_SET\_CLOSED**
- **CCI\_ER\_DELETED\_TUPLE**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_NO\_MORE\_MEMORY**

### cci\_fetch\_buffer\_clear

int cci\_fetch\_buffer\_clear(int req\_handle)

클라이언트 버퍼에 임시 저장된 레코드를 삭제한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들

반환:

에러 코드(0: 성공)

- **CCI\_ER\_REQ\_HANDLE**

### cci\_fetch\_sensitive

int cci\_fetch\_sensitive(int req\_handle, T\_CCI\_ERROR \*err\_buf)

서버에서 클라이언트로 **SELECT** 질의의 결과가 전송될 때 sensitive column에 대해서 변경된 값으로 전송되도록 한다. *req\_handle* 에 의한 결과가 sensitive result가 아닐 경우 `cci_fetch()` 와 동일하다. 리턴 값이 **CCI\_ER\_DELETED\_TUPLE**일 경우 해당 레코드는 삭제된 경우이다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **err\_buf** – (OUT) 데이터베이스 에러 버퍼

반환:

에러 코드 (0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_NO\_MORE\_DATA**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_DELETED\_TUPLE**

sensitive column이란 **SELECT** 리스트 항목 중 결과 재요청 시 업데이트된 값을 제공할 수 있는 항목을 말한다. 주로 어떠한 연산 없이, 예를 들면 집계 연산과 같은 과정이 없이 칼럼을 **SELECT** 리스트의 항목으로 그대로 쓰는 경우 그 칼럼을 sensitive column이라고 말할 수 있다.

질의 결과를 다시 fetch할 때, sensitive result는 클라이언트 버퍼에 저장된 레코드를 받지 않고, 서버로부터 변경된 값을 받는다.

## cci\_fetch\_size

int cci\_fetch\_size(int req\_handle, int fetch\_size)

이 함수는 더 이상 사용되지 않으며(deprecated), 제거될 예정이다. 호출되더라도 무시되어 동작에 어떠한 변화도 발생하지 않는다.

## cci\_get\_autocommit

CCI\_AUTOCOMMIT\_MODE cci\_get\_autocommit(int conn\_handle)

현재 설정한 자동 커밋 모드(autocommit mode)를 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

반환:

- **CCI\_AUTOCOMMIT\_TRUE**: 자동 커밋 모드 ON
- **CCI\_AUTOCOMMIT\_FALSE**: 자동 커밋 모드 OFF
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_get\_bind\_num

int cci\_get\_bind\_num(int req\_handle)

입력 바인딩(input binding) 개수를 가져온다. prepare 시 사용된 SQL 문이 여러 개의 질의로 구성되어 있을 경우, 전체 질의에서 사용된 입력 바인딩 개수를 나타낸다.

매개변수:

- **req\_handle** – (IN) prepared statement에 대한 요청 핸들

반환:

입력 바인딩 개수

- **CCI\_ER\_REQ\_HANDLE**

## cci\_get\_cas\_info

int cci\_get\_cas\_info(int conn\_handle, char \*info\_buf, int buf\_length, T\_CCI\_ERROR \*err\_buf)

conn\_handle에 연결되어 있는 CAS 정보를 조회한다. info\_buf에 아래와 같은 형식의 문자열이 리턴된다.

```
<host>:<port>,<cas id>,<cas process id>
```

출력 예는 다음과 같다.

```
127.0.0.1:33000,1,12916
```

CAS ID를 통해 해당 CAS의 SQL 로그 파일을 쉽게 확인할 수 있다.

보다 자세한 사항은 [SQL 로그 확인](#)을 참고한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **info\_buf** – (OUT) 연결 정보 버퍼
- **buf\_length** – (IN) 연결 정보 버퍼 길이
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_INVALID\_ARGS**
- **CCI\_ER\_CON\_HANDLE**

## cci\_get\_class\_num\_objs

```
int cci_get_class_num_objs(int conn_handle, char *class_name, int flag, int *num_objs, int
*num_pages, T_CCI_ERROR *err_buf)
```

*class\_name* 클래스의 객체 개수와 사용하고 있는 페이지 수를 가져온다. flag가 1일 경우 대략의 값을 가져오고, 0일 경우 정확한 값을 가져온다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **class\_name** – (IN) 클래스 이름
- **flag** – (IN) 0 또는 1
- **num\_objs** – (OUT) 객체 수
- **num\_pages** – (OUT) 페이지 수
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

## CCI\_GET\_COLLECTION\_DOMAIN

### CCI\_GET\_COLLECTION\_DOMAIN(*u\_type*)

*u\_type* 이 set, multiset, sequence type인 경우 set, multiset, sequence의 domain을 가져온다.  
*u\_type* 이 set type이 아닐 경우 리턴 값은 *u\_type* 과 같다.

매개변수:

- **u\_type** (Type) – (IN)

반환:

Type (CCI\_U\_TYPE)

## cci\_get\_cur\_oid

```
int cci_get_cur_oid(int req_handle, char *oid_str_buf)
```



Execute에서 **CCI\_INCLUDE\_OID** 가 설정된 경우 현재 fetch된 레코드의 OID를 가져온다. OID는 page, slot, volume에 의한 스트링으로 표현된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str\_buf** – (OUT) OID 스트링

반환:

에러 코드(0: 성공)

- **CCI\_ER\_REQ\_HANDLE**

## cci\_get\_data

int cci\_get\_data(int req\_handle, int col\_no, int type, void \*value, int \*indicator)

현재 fetch된 결과에 대해서 *col\_no* 번째의 값을 가져온다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **col\_no** – (IN) 칼럼 인덱스. 1부터 시작.
- **type** – (IN) *value* 변수의 데이터 타입(**T\_CCI\_A\_TYPE**에 정의된 타입을 사용)
- **value** – (OUT) 데이터를 저장할 변수의 주소. *type*이 **CCI\_A\_TYPE\_STR**, **CCI\_A\_TYPE\_SET**, **CCI\_A\_TYPE\_BLOB** 또는 **CCI\_A\_TYPE\_CLOB**이고 칼럼의 값이 NULL이면 *value*의 값도 NULL이다.
- **indicator** –  
(OUT) **NULL** indicator. (-1: **NULL**)
  - *type* 이 **CCI\_A\_TYPE\_STR** 인 경우: **NULL** 이면 -1을 반환하고, **NULL** 이 아니면 *value*에 저장된 문자열의 바이트 길이를 반환
  - *type* 이 **CCI\_A\_TYPE\_STR** 이 아닌 경우: **NULL** 이면 -1을 반환하고, **NULL** 이 아니면 0을 반환

반환:

에러 코드(0: 성공)

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_TYPE\_CONVERSION**
- **CCI\_ER\_COLUMN\_INDEX**
- **CCI\_ER\_ATYPE**

주어진 *type* 인자에 따라 *value* 변수의 타입이 결정되고, 이에 따라 *value* 변수로 값 또는 포인터가 복사된다. 값을 복사하는 경우 *value* 변수로 전달되는 주소에 대한 메모리가 할당되어 있어야 한다. 포인터 복사의 경우 응용 클라이언트 라이브러리 내의 포인터를 반환하는 것이므로, 다음 `cci_get_data()` 함수 호출 시 해당 값이 유효하지 않게 되므로 주의한다.

포인터 복사에 의해 반환된 포인터는 해제(`free`)하면 안 된다. 단, 타입이 **CCI\_A\_TYPE\_SET** 인 경우 **T\_CCI\_SET** 타입의 set 포인터를 메모리에 할당한 후 이를 반환하므로, set 포인터를 사용한 후에는 `cci_set_free()` 함수를 이용하여 할당된 메모리를 해제해야 한다. 아래는 *type* 인자와 그에 대응하는 *value*의 데이터 타입을 정리한 표이다.

| type              | value Type                         | Meaning                                   |
|-------------------|------------------------------------|-------------------------------------------|
| CCI_A_TYPE_STR    | char **                            | pointer copy                              |
| CCI_A_TYPE_INT    | int *                              | value copy                                |
| CCI_A_TYPE_FLOAT  | float *                            | value copy                                |
| CCI_A_TYPE_DOUBLE | double *                           | value copy                                |
| CCI_A_TYPE_BIT    | <b>T_CCI_BIT</b> *                 | value copy (pointer copy for each member) |
| CCI_A_TYPE_SET    | <b>T_CCI_SET</b> *                 | memory allocation and value copy          |
| CCI_A_TYPE_DATE   | <b>T_CCI_DATE</b> *                | value copy                                |
| CCI_A_TYPE_BIGINT | int64_t * (For Windows: __int64 *) | value copy                                |
| CCI_A_TYPE_BLOB   | <b>T_CCI_BLOB</b> *                | memory allocation and value copy          |
| CCI_A_TYPE_CLOB   | <b>T_CCI_CLOB</b> *                | memory allocation and value copy          |

#### 참고

- **LOB** 타입에 대해 `cci_get_data()` 를 호출하면 **LOB** 타입 칼럼의 메타 데이터(Locator)를 출력하며, **LOB** 타입 칼럼의 데이터를 인출하려면 `cci_blob_read()` 를 호출해야 한다.

다음 예제는 페치한 결과 값을 `cci_get_data()` 를 이용하여 출력하는 코드의 일부이다.

```

...

if ((res=cci_get_data(req, i, CCI_A_TYPE_INT, &buffer, &ind))<0)
{
 printf("%s(%d): cci_get_data fail\n", __FILE__, __LINE__);
 goto handle_error;
}
if (ind != -1)
 printf("%d \t|", buffer);
else
 printf("NULL \t|");
...

```

### cci\_get\_db\_parameter

int cci\_get\_db\_parameter(int conn\_handle, T\_CCI\_DB\_PARAM param\_name, void \*value, T\_CCI\_ERROR \*err\_buf)

데이터베이스에 설정된 파라미터 값을 가져온다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **param\_name** – (IN) 시스템 파라미터 이름
- **value** – (OUT) 파라미터 값
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_PARAM\_NAME**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

*param\_name* 에 대한 *value* 의 데이터 타입은 다음과 같다.

| param_name                         | value 타입 | note     |
|------------------------------------|----------|----------|
| <b>CCI_PARAM_ISOLATION_LEVEL</b>   | int *    | get/set  |
| <b>CCI_PARAM_LOCK_TIMEOUT</b>      | int *    | get/set  |
| <b>CCI_PARAM_MAX_STRING_LENGTH</b> | int *    | get only |
| <b>CCI_PARAM_AUTO_COMMIT</b>       | int *    | get only |

`cci_get_db_parameter()`, `cci_set_db_parameter()` 에서 **CCI\_PARAM\_LOCK\_TIMEOUT** 의 입출력 단위는 밀리초이다.

#### ⚠ 경고

CUBRID 9.0 미만 버전에서 **CCI\_PARAM\_LOCK\_TIMEOUT** 의 출력 단위는 초이므로 주의해야 한다.

**CCI\_PARAM\_MAX\_STRING\_LENGTH** 의 단위는 바이트이며, 브로커 파라미터 **MAX\_STRING\_LENGTH** 에 정의된 값을 가져온다.

## cci\_get\_db\_version

`int cci_get_db_version(int conn_handle, char *out_buf, int out_buf_size)`  
DBMS (Database Management System) 버전을 가져온다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **out\_buf** – (OUT) 결과 버퍼
- **out\_buf\_size** – (IN) *out\_buf* 크기

반환:

에러 코드(0: 성공)

- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_COMMUNICATION
- CCI\_ER\_CONNECT

## cci\_get\_err\_msg

int cci\_get\_err\_msg(int err\_code, char \*msg\_buf, int msg\_buf\_size)

에러 코드에 대응되는 에러 메시지를 에러 메시지 버퍼에 저장한다. 에러 코드와 에러 메시지에 대한 내용은 **CCI 에러 코드와 에러 메시지** 를 참고한다.

매개변수:

- **err\_code** – (IN) 에러 코드
- **msg\_buf** – (OUT) 에러 메시지 버퍼
- **msg\_buf\_size** – (IN) *msg\_buf* 크기

반환:

0 (성공), -1 (실패)

note:: CUBRID 9.1 부터는 err\_buf가 있는 모든 함수에서 err\_buf에 값이 저장되는 것을 보장하므로 err\_buf 인자가 있는 함수에서는 cci\_get\_err\_msg 함수를 사용할 필요가 없으며, `cci_get_error_msg()` 함수를 사용할 것을 권장한다.

```
req = cci_prepare (con, query, 0, &err_buf);
if (req < 0)
{
 printf ("error: %d, %s\n", err_buf.err_code,
err_buf.err_msg);
 goto handle_error;
}
```

9.1 미만 버전에서는 CCI\_ER\_DBMS 에러가 발생했을 때만 err\_buf에 에러 정보가 저장되므로 CCI\_ER\_DBMS에 따라 다음과 같이 분기하여 처리해야 했다.

```
req = cci_prepare (con, query, 0, &err_buf);
if (req < 0)
{
```

```

 if (req == CCI_ER_DBMS)
 {
 printf ("error: %s\n", err_buf.err_msg);
 }
 else
 {
 char msg_buf[1024];
 cci_get_err_msg(req, msg_buf, 1024);
 printf ("error: %s\n", msg_buf);
 }
 goto handle_error;
}

```

9.1 버전부터는 cci\_get\_error\_msg 함수를 사용하여 위의 분기 코드를 단순화할 수 있다.

```

req = cci_prepare (con, query, 0, &err_buf);
if (req < 0)
{
 char msg_buf[1024];
 cci_get_error_msg(req, err_buf, msg_buf, 1024);
 printf ("error: %s\n", msg_buf);
 goto handle_error;
}

```

## cci\_get\_error\_msg

int cci\_get\_error\_msg(int err\_code, T\_CCI\_ERROR \*err\_buf, char \*msg\_buf, int msg\_buf\_size)

CCI 에러 코드에 대응되는 에러 메시지를 에러 메시지 버퍼에 저장한다. CCI 에러 코드의 값이 **CCI\_ER\_DBMS** 이면 데이터베이스 서버에서 발생한 에러 메시지를 데이터베이스 에러 버퍼 (err\_buf)에서 전달받아 메시지 버퍼(msg\_buf)에 저장한다. 에러 코드와 에러 메시지에 대한 내용은 **CCI 에러 코드와 에러 메시지**를 참고한다.

매개변수:

- **err\_code** - (IN) 에러 코드
- **err\_buf** - (IN) 데이터베이스 에러 버퍼
- **msg\_buf** - (OUT) 에러 메시지 버퍼
- **msg\_buf\_size** - (IN) msg\_buf 크기

반환:

0 (성공), -1 (실패)

## cci\_get\_holdability

int cci\_get\_holdability(int conn\_handle)

연결 핸들에서 결과 셋에 대한 커서 유지(cursor holdability) 설정 값을 리턴한다. 값이 1이면 커밋 여부에 관계 없이 연결이 종료되거나 결과 셋을 의도적으로 닫기 전까지 커서를 유지(holdable)하고, 0이면 커밋될 때 결과 셋이 닫히면서 커서를 유지하지 않는다(not holdable). 커서 유지에 대한 자세한 설명은 CUBRID SQL 설명서 > 트랜잭션과 잠금 > 커서 유지를 참고한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들

반환:

0 (not holdable), 1 (holdable)

- **CCI\_ER\_CON\_HANDLE**

## cci\_get\_last\_insert\_id

int cci\_get\_last\_insert\_id(int conn\_handle, void \*value, T\_CCI\_ERROR \*err\_buf)

가장 마지막에 수행한 INSERT 문의 기본 키 값을 얻는다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **value** – (OUT) 결과 버퍼 포인터의 포인터(char \*\*). 가장 마지막에 수행한 INSERT 문의 기본 키 값을 저장. 이 포인터가 가리키는 메모리는 연결 핸들 내부의 고정된 버퍼로 별도로 해제할 필요가 없다.
- **err\_buf** – (OUT) 에러 버퍼

반환:



에러 코드(0: 성공)

- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_USED\_CONNECTION
- CCI\_ER\_INVALID\_ARGS

```
#include <stdio.h>
#include "cas_cci.h"

int main ()
{
 int con = 0;
 int req = 0;

 int error;
 T_CCI_ERROR cci_error;

 char *query = "insert into t1 values(NULL);";
 char *value = NULL;

 con = cci_connect ("localhost", 33000, "demodb", "dba", "");

 if (con < 0)
 {
 printf ("con error\n");
 return;
 }

 req = cci_prepare (con, query, 0, &cci_error);
 if (req < 0)
 {
 printf ("cci_prepare error: %d\n", req);
 printf ("cci_error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 return;
 }
 error = cci_execute (req, 0, 0, &cci_error);
 if (error < 0)
 {
 printf ("cci_execute error: %d\n", error);
 return;
 }

 error = cci_get_last_insert_id (con, &value, &cci_error);
 if (error < 0)
 {
 printf ("cci_get_last_insert_id error: %d\n", error);
 }
}
```

```

 return;
 }

 printf ("## last insert id: %s\n", value);

 error = cci_close_req_handle (req);
 error = cci_disconnect (con, &cci_error);
 return 0;
}

```

## cci\_get\_login\_timeout

int cci\_get\_login\_timeout(int conn\_handle, int \*timeout, T\_CCI\_ERROR \*err\_buf)

로그인 타임아웃 값을 *timeout*에 반환한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **timeout** – (OUT) 로그인 타임아웃 값(단위: 밀리 초)에 대한 포인터
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_INVALID\_ARGS**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_get\_query\_plan

int cci\_get\_query\_plan(int req\_handle, char \*\*out\_buf\_p)

cci\_prepare 함수가 리턴한 요청 핸들(req\_handle)에 대한 질의 계획을 결과 버퍼에 출력한다. cci\_execute 함수의 호출 여부와 상관 없이 cci\_get\_query\_plan 함수를 호출할 수 있다.

cci\_get\_query\_plan 함수 호출 후 결과 버퍼의 사용이 끝나면 `cci_query_info_free()` 함수를 이용하여 cci\_get\_query\_plan 함수에서 생성된 결과 버퍼를 해제해야 한다.

```

char *out_buf;
...
req = cci_prepare (con, query, 0, &cci_error);
...
ret = cci_get_query_plan(req, &out_buf);
...
printf("plan = %s", out_buf);
cci_query_info_free(out_buf);

```

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **out\_buf\_p** – (OUT) 결과 버퍼 포인터의 포인터

반환:

에러 코드

- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**

#### ❗ 더 보기

`cci_query_info_free()`

## cci\_query\_info\_free

**int cci\_query\_info\_free(char \*out\_buf)**

cci\_get\_query\_plan 함수에서 할당된 결과 버퍼 메모리를 해제한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들
- **out\_buf** – (OUT) 결과 버퍼 포인터

반환:

에러 코드

- **CCI\_ER\_NO\_MORE\_MEMORY**

#### ! 더 보기

`cci_get_query_plan()`

## cci\_get\_query\_timeout

`int cci_get_query_timeout(int req_handle)`

질의 수행에 대해 설정된 타임아웃 시간을 반환한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들

반환:

현재 요청 핸들에 설정된 제한 시간(timeout) 값. 단위는 msec

- **CCI\_ER\_REQ\_HANDLE**

## cci\_get\_result\_info

`T_CCI_COL_INFO *cci_get_result_info(int req_handle, T_CCI_CUBRID_STMT *stmt_type, int *num)`

prepared statement가 **SELECT** 일 경우, 이 함수를 이용하여 실행 결과에 대한 칼럼 정보가 저장되어 있는 **T\_CCI\_COL\_INFO** 구조체를 가져올 수 있다. **SELECT** 질의가 아닌 경우, **NULL**을 반환하고 *num* 값은 0이 된다.

매개변수:

- **req\_handle** – (IN) prepared statement에 대한 요청 핸들
- **stmt\_type** – (OUT) command 타입
- **num** – (OUT) **SELECT** 문의 칼럼 개수( *stmt\_type* 이 **CUBRID\_STMT\_SELECT** 일 경우)

반환:

result info 포인터 (성공), **NULL** (실패)

**T\_CCI\_COL\_INFO** 구조체에서 칼럼 정보를 가져오기 위해서 구조체에 직접 접근해도 되지만, 다음과 같이 정의된 매크로를 이용하여 정보를 가져올 수 있다. 각 매크로의 인자로 **T\_CCI\_COL\_INFO** 구조체의 주소와 칼럼 인덱스가 지정되며, 매크로는 **SELECT** 질의에 대해서만 호출할 수 있다. 매크로에서 입력받는 각 인자에 대한 유효성 검사가 이루어지지 않으므로 주의한다. 매크로 리턴 값의 타입이 char\*인 경우 메모리 포인터를 해제(free)하지 않아야 한다.

| 매크로                                          | 리턴 값 타입             | 의미            |
|----------------------------------------------|---------------------|---------------|
| <code>CCI_GET_RESULT_INFO_TYPE</code>        | <b>T_CCI_U_TYPE</b> | 칼럼의 type      |
| <code>CCI_GET_RESULT_INFO_SCALE</code>       | short               | 칼럼의 scale     |
| <code>CCI_GET_RESULT_INFO_PRECISION</code>   | int                 | 칼럼의 precision |
| <code>CCI_GET_RESULT_INFO_NAME</code>        | char *              | 칼럼의 이름        |
| <code>CCI_GET_RESULT_INFO_ATTR_NAME</code>   | char *              | 칼럼의 속성 이름     |
| <code>CCI_GET_RESULT_INFO_CLASS_NAME</code>  | char *              | 칼럼의 클래스 이름    |
| <code>CCI_GET_RESULT_INFO_IS_NON_NULL</code> | char (0 or 1)       | 칼럼이NULL 인지 여부 |

```
col_info = cci_get_result_info (req, &stmt_type, &col_count);
if (col_info == NULL)
{
 printf ("get_result_info error\n");
 goto handle_error;
}

for (i = 1; i <= col_count; i++)
{
 printf ("%12s = %d\n", "type", CCI_GET_RESULT_INFO_TYPE
(col_info, i));
 printf ("%12s = %d\n", "scale",
```

```

 CCI_GET_RESULT_INFO_SCALE (col_info, i));
 printf ("%12s = %d\n", "precision",
 CCI_GET_RESULT_INFO_PRECISION (col_info, i));
 printf ("%12s = %s\n", "name", CCI_GET_RESULT_INFO_NAME
(col_info, i));
 printf ("%12s = %s\n", "attr_name",
 CCI_GET_RESULT_INFO_ATTR_NAME (col_info, i));
 printf ("%12s = %s\n", "class_name",
 CCI_GET_RESULT_INFO_CLASS_NAME (col_info, i));
 printf ("%12s = %s\n", "is_non_null",
 CCI_GET_RESULT_INFO_IS_NON_NULL (col_info,i) ?
"true" : "false");
}

```

## CCI\_GET\_RESULT\_INFO\_ATTR\_NAME

### CCI\_GET\_RESULT\_INFO\_ATTR\_NAME(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 실제 속성 이름을 가져오는 매크로이다. 속성 이름이 없는 경우(상수값, 함수 등)는 빈 문자열 (empty string)을 반환한다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다. 반환된 메모리 포인터는 사용자가 **free()**를 통해 제거할 수 없다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

속성 이름 (char \*)

## CCI\_GET\_RESULT\_INFO\_CLASS\_NAME

### CCI\_GET\_RESULT\_INFO\_CLASS\_NAME(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 클래스 이름을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다. 반환된 메모리 포인터는 사용자가 **free()**를 통해 제거할 수 없다. 반환된 값은 **NULL**을 가질 수 있다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

클래스 이름 (char \*)

## CCI\_GET\_RESULT\_INFO\_IS\_NON\_NULL

### CCI\_GET\_RESULT\_INFO\_IS\_NON\_NULL(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼이 nullable인지에 대한 값을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

**SELECT** 리스트의 칼럼이 테이블의 칼럼이 아닌 표현식인 경우 **NON\_NULL** 여부를 알 수 없으므로 CCI\_GET\_RESULT\_INFO\_IS\_NON\_NULL 매크로는 일관되게 0을 반환한다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

0 : nullable, 1 : non **NULL**

## CCI\_GET\_RESULT\_INFO\_NAME

### CCI\_GET\_RESULT\_INFO\_NAME(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 이름을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다. 반환된 메모리 포인터는 사용자가 **free()**를 통해 제거할 수 없다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

칼럼 이름 (char \*)

## CCI\_GET\_RESULT\_INFO\_PRECISION

### CCI\_GET\_RESULT\_INFO\_PRECISION(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 precision을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

precision (int)

## CCI\_GET\_RESULT\_INFO\_SCALE

### CCI\_GET\_RESULT\_INFO\_SCALE(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 scale을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.



매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

scale (int)

## CCI\_GET\_RESULT\_INFO\_TYPE

### CCI\_GET\_RESULT\_INFO\_TYPE(res\_info, index)

prepare된 **SELECT** 리스트에서 *index* 번째 칼럼의 타입을 가져오는 매크로이다. 지정된 인자 *res\_info* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

어떤 칼럼의 SET 타입 여부를 확인하려면 `CCI_IS_SET_TYPE` 을 사용한다.

매개변수:

- **res\_info** (T\_CCI\_COL\_INFO\*) – (IN) `cci_get_result_info()` 에 의한 칼럼 정보 포인터
- **index** (int) – (IN) 칼럼 인덱스

반환:

칼럼 타입 (**T\_CCI\_U\_TYPE**)

## CCI\_IS\_SET\_TYPE

### CCI\_IS\_SET\_TYPE(u\_type)

*u\_type*이 set type인지를 검사한다.

매개변수:

· **u\_type** (Type) – (IN)

반환:

1 : set, 0 : not set

## CCI\_IS\_MULTISSET\_TYPE

### CCI\_IS\_MULTISSET\_TYPE(u\_type)

*u\_type*이 multiset type인지를 검사한다.

매개변수:

· **u\_type** (Type) – (IN)

반환:

1 : multiset, 0 : not multiset

## CCI\_IS\_SEQUENCE\_TYPE

### CCI\_IS\_SEQUENCE\_TYPE(u\_type)

*u\_type*이 sequence type인지를 검사한다.

매개변수:

· **u\_type** (Type) – (IN)

반환:

1 : sequence, 0 : not sequence

## CCI\_IS\_COLLECTION\_TYPE

### CCI\_IS\_COLLECTION\_TYPE(u\_type)

*u\_type*이 collection (set, multiset, sequence) type인지를 검사한다.

매개변수:

- **u\_type** (Type) – (IN)

반환:

1 : collection (set, multiset, sequence), 0 : not collection

## cci\_get\_version

int cci\_get\_version(int \*major, int \*minor, int \*patch)

CCI 라이브러리의 버전을 가져온다. 예를 들어 버전 스트링이 “9.2.0.0001” 인 경우, major 버전은 9, minor 버전은 2, patch 버전은 00이 된다.

매개변수:

- **major** – (OUT) major 버전
- **minor** – (OUT) minor 버전
- **patch** – (OUT) patch 버전

반환:

항상 0(성공)

### 참고

Linux용 CUBRID에서는 **strings** 명령을 이용하여 CCI 라이브러리 파일의 버전을 확인할 수 있다

```
$ strings /home/usr1/CUBRID/lib/libcascci.so | grep VERSION
VERSION=9.2.0.0001
```

## cci\_init

void cci\_init()

Windows용 CCI 응용 프로그램을 static linking library(.lib)로 컴파일하는 경우에는 반드시 호출해야 하고, 그 이외의 경우는 이 함수를 사용할 필요가 없다.

## cci\_is\_holdable

int cci\_is\_holdable(int req\_handle)

**cci\_is\_holdable** 함수는 요청 핸들의 연결 유지(holdable) 가능 여부를 리턴한다.

매개변수:

- **req\_handle** – (IN) prepared statement에 대한 요청 핸들

반환:

- 1: 연결 유지
- 0: 연결 유지 안 됨
- **CCI\_ER\_REQ\_HANDLE**

❗ 더 보기

`cci_prepare()`

## cci\_is\_updatable

int cci\_is\_updatable(int req\_handle)

`cci_prepare()` 를 수행한 SQL 문이 업데이트 가능한 결과 셋을 만들 수 있는 질의인지 (`cci_prepare()` 수행 시 *flag* 에 **CCI\_PREPARE\_UPDATABLE** 이 설정되었는지) 확인하는 함수이다. 업데이트 가능한 질의이면 1을 반환한다.

매개변수:

- **req\_handle** – (IN) prepared statement에 대한 요청 핸들

반환:

- 1: 업데이트 가능
- 0: 업데이트 불가능
- **CCI\_ER\_REQ\_HANDLE**

## cci\_next\_result

int cci\_next\_result(int req\_handle, T\_CCI\_ERROR \*err\_buf)

`cci_execute()` 수행 시 **CCI\_EXEC\_QUERY\_ALL** flag가 설정되면 여러 개의 질의를 순서대로 수행할 수 있게 된다. 이때, 첫 번째 질의 결과는 `cci_execute()` 함수를 호출한 뒤에 가져오고, 두 번째 질의부터는 **cci\_next\_result** 함수를 호출할 때마다 질의를 작성한 순서대로 결과를 가져온다.

이때 얻은 질의 결과에 대한 칼럼 정보는 `cci_execute()` 함수 또는 **cci\_next\_result** 함수를 호출할 때마다 `cci_get_result_info()` 함수를 호출하면 된다.

즉, Q1, Q2, Q3 질의 여러 개를 한 번의 `cci_prepare()` 호출을 통해 수행할 경우 Q1의 결과는 `cci_execute()` 함수를 호출할 때 가져오며, Q2, Q3의 결과는 **cci\_next\_result** 함수를 각각 호출하여 가져온다. Q1, Q2, Q3의 질의 결과에 대한 칼럼 정보는 매번 `cci_get_result_info()` 함수를 호출하여 가져온다.

```
sql = "SELECT * FROM athlete; SELECT * FROM nation; SELECT *
FROM game";
req = cci_prepare (con, sql, 0, &error);
...
ret = cci_execute (req, CCI_EXEC_QUERY_ALL, 0, &error);
res_col_info = cci_get_result_info (req, &cmd_type, &col_count);
...
ret = cci_next_result (req, &error);
res_col_info = cci_get_result_info (req, &cmd_type, &col_count);
...
ret = cci_next_result (req, &error);
res_col_info = cci_get_result_info (req, &cmd_type, &col_count);
```

매개변수:

- **req\_handle** – (IN) prepared statement에 대한 요청 핸들
- **err\_buf** – (OUT) 에러 버퍼

반환:

- **SELECT** : 결과 개수
- **INSERT, UPDATE** : 반영된 레코드 개수
- 기타 : 0
- 실패 : 에러 코드
  - **CCI\_ER\_REQ\_HANDLE**

- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**

에러 코드가 **CAS\_ER\_NO\_MORE\_RESULT\_SET** 일 경우 더 이상의 결과 셋이 존재하지 않는다는 것을 의미한다.

## cci\_oid

`int cci_oid(int conn_handle, T_CCI_OID_CMD cmd, char *oid_str, T_CCI_ERROR *err_buf)`

*cmd* 인자의 값에 따라 다음 동작을 수행한다.

- **CCI\_OID\_DROP** : 해당 oid를 삭제한다.
- **CCI\_OID\_IS\_INSTANCE** : 해당 oid가 instance oid인지를 검사한다.
- **CCI\_OID\_LOCK\_READ** : 해당 oid에 대해 read lock 을 설정한다.
- **CCI\_OID\_LOCK\_WRITE** : 해당 oid에 대해 write lock을 설정한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **cmd** – (IN) CCI\_OID\_DROP, CCI\_OID\_IS\_INSTANCE, CCI\_OID\_LOCK\_READ, CCI\_OID\_LOCK\_WRITE
- **oid\_str** – (IN) oid
- **err\_buf** – (OUT) 에러 버퍼

반환:

- *cmd*가 CCI\_OID\_IS\_INSTANCE인 경우
  - 0 : 인스턴스 아님
  - 1 : 인스턴스
  - < 0 : 에러
- *cmd*가 CCI\_OID\_DROP, CCI\_OID\_LOCK\_READ 또는 CCI\_OID\_LOCK\_WRITE인 경우

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**

- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OID\_CMD**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_oid\_get

int cci\_oid\_get(int conn\_handle, char \*oid\_str, char \*\*attr\_name, T\_CCI\_ERROR \*err\_buf)

해당 oid의 속성 값을 가져온다. *attr\_name* 은 속성의 array로서 마지막은 반드시 NULL로 끝나야 한다. *attr\_name* 이 NULL인 경우 모든 속성에 대한 정보를 가져온다. Request handle의 형태는 “SELECT attr\_name FROM oid\_class WHERE oid\_class = oid”의 SQL문을 실행했을 때와 동일한 형태이다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **attr\_name** – (IN) 속성 목록
- **err\_buf** – (OUT) 에러 버퍼

반환:

성공 : 요청 핸들, 실패 : 에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_CONNECT**

## cci\_oid\_get\_class\_name

int cci\_oid\_get\_class\_name(int conn\_handle, char \*oid\_str, char \*out\_buf, int out\_buf\_len, T\_CCI\_ERROR \*err\_buf)

해당 oid의 클래스 이름을 가져온다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **out\_buf** – (OUT) out 버퍼
- **out\_buf\_len** – (IN) *out\_buf* 길이
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_OBJECT**
- **CCI\_ER\_DBMS**

## cci\_oid\_put

```
int cci_oid_put(int conn_handle, char *oid_str, char **attr_name, char **new_val_str,
T_CCI_ERROR *err_buf)
```

해당 oid의 *attr\_name* 속성 값을 *new\_val\_str* 으로 설정한다. *attr\_name* 의 마지막은 반드시 NULL이어야 한다. 모든 타입의 값은 string으로 표현해야 하고, string으로 표현된 값은 서버에서 속성의 타입에 따라 변환되어 데이터베이스에 반영된다. NULL 값을 넣기 위해서는 *new\_val\_str* [i]의 값을 NULL로 한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oid\_str** – (IN) oid
- **attr\_name** – (IN) 속성 이름 목록
- **new\_val\_str** – (IN) 새 값의 목록
- **err\_buf** – (OUT) 에러 버퍼



반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**

## cci\_oid\_put2

```
int cci_oid_put2(int conn_handle, char *oidstr, char **attr_name, void **new_val, int *a_type,
T_CCI_ERROR *err_buf)
```

해당 oid의 *attr\_name* 속성 값을 *new\_val* 로 설정한다. *attr\_name* 의 마지막은 반드시 NULL이어야 한다. NULL 값을 넣기 위해서는 *new\_val* [i]의 값을 NULL로 지정한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **oidstr** – (IN) oid
- **attr\_name** – (IN) 속성 이름 목록
- **new\_val** – (IN) 새 값 배열
- **a\_type** – (IN) *new\_val* 타입 배열
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**

*a\_type* 에 대한 *new\_val* [i]의 타입은 다음 표와 같다.

**a\_type**에 대한 **new\_val[i]**의 타입

| Type              | value type                     |
|-------------------|--------------------------------|
| CCI_A_TYPE_STR    | char **                        |
| CCI_A_TYPE_INT    | int *                          |
| CCI_A_TYPE_FLOAT  | float *                        |
| CCI_A_TYPE_DOUBLE | double *                       |
| CCI_A_TYPE_BIT    | T_CCI_BIT *                    |
| CCI_A_TYPE_SET    | T_CCI_SET *                    |
| CCI_A_TYPE_DATE   | T_CCI_DATE *                   |
| CCI_A_TYPE_BIGINT | int64_t * (Windows는 __int64 *) |

```

char *attr_name[array_size]
void *attr_val[array_size]
int a_type[array_size]
int int_val

...

attr_name[0] = "attr_name0"
attr_val[0] = &int_val
a_type[0] = CCI_A_TYPE_INT
attr_name[1] = "attr_name1"
attr_val[1] = "attr_val1"
a_type[1] = CCI_A_TYPE_STR

...
attr_name[num_attr] = NULL

res = cci_put2(con_h, oid_str, attr_name, attr_val, a_type,
&error)

```

## cci\_prepare

int cci\_prepare(int conn\_handle, char \*sql\_stmt, char flag, T\_CCI\_ERROR \*err\_buf)

SQL 문에 관한 요청 핸들을 획득하여 SQL 실행을 준비한다. 단, SQL 문이 여러 개의 질의로 구성된 경우, 첫 번째 질의에 대해서만 실행을 준비한다. 이 함수의 인자로 연결 핸들, SQL문, *flag*, 오류 정보를 저장할 **T\_CCI\_ERROR** 구조체 변수의 주소가 지정된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **sql\_stmt** – (IN) SQL 문
- **flag** – (IN) prepare flag (CCI\_PREPARE\_UPDATABLE, CCI\_PREPARE\_INCLUDE\_OID, CCI\_PREPARE\_HOLDABLE 또는 CCI\_PREPARE\_CALL)
- **err\_buf** – (OUT) 에러 버퍼

반환:

성공 : 요청 핸들 ID, 실패 : 에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_STR\_PARAM**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_QUERY\_TIMEOUT**
- **CCI\_ER\_LOGIN\_TIMEOUT**

*flag* 에 **CCI\_PREPARE\_UPDATABLE**, **CCI\_PREPARE\_INCLUDE\_OID**, **CCI\_PREPARE\_HOLDABLE** 또는 **CCI\_PREPARE\_CALL** 이 설정될 수 있다. *flag*에 **CCI\_PREPARE\_UPDATABLE**이 설정되면 갱신 가능한 결과 셋(updatable resultset)을 만들 수 있으며, 이 경우 **CCI\_PREPARE\_INCLUDE\_OID**는 자동 설정된다. *flag*에 **CCI\_PREPARE\_UPDATABLE**과 **CCI\_PREPARE\_HOLDABLE**을 동시에 사용할 수 없다. 자바 저장 함수를 호출하고 싶으면 *flag*에 **CCI\_PREPARE\_CALL**을 설정한다. 자바 저장 함수와 관련된 예는 [cci\\_register\\_out\\_param\(\)](#) 함수를 참고한다.

커밋 이후 결과 셋 유지 여부에 대한 설정의 기본값은 커서 유지이다. 따라서 `cci_prepare()`의 `flag`에 **CCI\_PREPARE\_UPDATABLE**을 설정하고 싶으면 `cci_prepare()`를 호출하기 전에 `cci_set_holdability()`을 호출하여 커서를 유지하지 않도록 설정해야 한다.

**CCI\_PREPARE\_UPDATABLE**이 설정되더라도 모든 질의에 대해 갱신 가능한 결과를 만들 수 있는 것은 아니므로 SQL 문을 prepare한 후 `cci_is_updatable()` 함수를 이용하여 갱신 가능한 결과를 만들 수 있는 질의인지 확인해야 한다. 결과 셋을 갱신하려면 `cci_oid_put()` 함수 또는 `cci_oid_put2()` 함수를 사용할 수 있다.

갱신 가능한 질의의 조건은 다음과 같다.

- **SELECT** 질의여야 한다.
- 질의 결과에 OID가 포함될 수 있는 질의여야 한다.
- 갱신하고자 하는 칼럼이 **FROM** 절에 명시한 테이블에 속한 칼럼이어야 한다.

커밋 이후 결과 셋 유지에 대한 설정값이 기본값인 커서 유지이거나 **CCI\_PREPARE\_HOLDABLE**이 설정된 채로 prepare되었으면 해당 문장(statement)에 대해 결과 셋을 닫거나 연결을 종료하지 않는 한 커밋 이후에도 커서가 유지된다([커서 유지](#) 참고).

## cci\_prepare\_and\_execute

```
int cci_prepare_and_execute(int conn_handle, char *sql_stmt, int max_col_size, int *exec_retval,
T_CCI_ERROR *err_buf)
```

SQL 문을 즉시 실행하고 SQL 문에 대한 요청 핸들을 반환한다. 이 함수의 인자로는 연결 핸들, SQL 문, fetch하는 칼럼의 문자열 최대 길이, 에러 코드, 오류 정보를 저장할 **T\_CCI\_ERROR** 구조체 변수의 주소가 지정된다. `max_col_size`는 SQL 문의 칼럼이 **CHAR**, **VARCHAR**, **BIT**, **VARBIT**일 경우 클라이언트로 전송되는 칼럼의 문자열 최대 길이를 설정하기 위한 값이며, 이 값이 0이면 전체 길이를 fetch한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **sql\_stmt** – (IN) SQL 문
- **max\_col\_size** – (IN) 문자열 타입인 경우 fetch하는 칼럼의 문자열 최대 길이(단위: 바이트). 이 값이 0이면 전체 길이를 fetch한다.
- **exec\_retval** – (OUT) 성공: 영향을 받은 행의 개수, 실패: 에러 코드
- **err\_buf** – (OUT) 에러 버퍼

반환:

성공 : 요청 핸들 ID, 실패 : 에러 코드

- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_DBMS
- CCI\_ER\_COMMUNICATION
- CCI\_ER\_STR\_PARAM
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_CONNECT
- CCI\_ER\_QUERY\_TIMEOUT

## cci\_property\_create

T\_CCI\_PROPERTIES \*cci\_property\_create()

CCI의 DATASOURCE를 설정하기 위한 **T\_CCI\_PROPERTIES** 구조체를 생성한다.

반환:

성공: 메모리가 할당된 **T\_CCI\_PROPERTIES** 구조체 포인터, 실패 :

**NULL**

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_release()
```

## cci\_property\_destroy

void cci\_property\_destroy(T\_CCI\_PROPERTIES \*properties)

**T\_CCI\_PROPERTIES** 구조체를 삭제한다.

매개변수:

- **properties** – 삭제할 **T\_CCI\_PROPERTIES** 구조체 포인터

#### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_release()
```

## cci\_property\_get

char \*cci\_property\_get(T\_CCI\_PROPERTIES \*properties, char \*key)  
**T\_CCI\_PROPERTIES** 구조체에 설정된 속성값을 조회한다.

매개변수:

- **properties** – *key* 에 대응하는 *value*를 가져올 **T\_CCI\_PROPERTIES** 구조체 포인터
- **key** – 조회할 속성 이름(설정할 수 있는 속성의 이름과 의미는 `cci_property_set()` 함수 참고)

반환:

성공: *key* 에 대응하는 *value* 문자열의 포인터, 실패: NULL

#### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_release()
```

## cci\_property\_set

int cci\_property\_set(T\_CCI\_PROPERTIES \*properties, char \*key, char \*value)  
**T\_CCI\_PROPERTIES** 구조체에 속성 값을 설정한다.

매개변수:

- **properties** – *key* 와 *value* 를 저장할 **T\_CCI\_PROPERTIES** 구조체 포인터
- **key** – 속성 이름의 문자열 포인터
- **value** – 속성 값의 문자열 포인터

반환:

성공: 1, 실패: 0

구조체에 설정할 수 있는 속성의 이름 및 의미는 다음과 같다.

| 속성 이름         | 타입     | 기본값       | 의미                                                                  |
|---------------|--------|-----------|---------------------------------------------------------------------|
| user          | string |           | DB 사용자 이름                                                           |
| password      | string |           | DB 사용자 암호                                                           |
| url           | string |           | 연결 URL. 연결 URL 문자열을 정의하는 방법은 <code>cci_connect_with_url()</code> 참고 |
| pool_size     | int    | 10        | 연결 풀이 가질 수 있는 최대 연결 개수                                              |
| max_pool_size | int    | pool_size | 최초 datasource 생성 시 생성할 전체 연결 개수. 최대값은 INT_MAX                       |

| 속성 이름                       | 타입   | 기본값                                         | 의미                                                                                                                              |
|-----------------------------|------|---------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| max_wait                    | msec | 1000                                        | 연결을 가져오기 위해 대기하는 최대 시간                                                                                                          |
| pool_prepared_statement     | bool | false                                       | statement 풀링 가능 여부. true 또는 false                                                                                               |
| max_open_prepared_statement | int  | 1000                                        | statement pool에 유지할 prepared statement의 최대 개수                                                                                   |
| login_timeout               | msec | 0(무제한)                                      | <code>cci_datasource_create()</code> 함수로 datasource를 생성하거나 <code>prepare/execute</code> 함수에서 내부적인 재접속이 발생할 때마다 적용되는 로그인 타임아웃 시간 |
| query_timeout               | msec | 0(무제한)                                      | 질의 타임아웃 시간                                                                                                                      |
| disconnect_on_query_timeout | bool | no                                          | 질의 실행이 타임아웃 시간을 초과하여 실행이 취소될 때 연결의 종료 여부. yes 또는 no                                                                             |
| default_autocommit          | bool | cubridBroker.conf의 CCI_DEFAULT_AUTOCOMMIT 값 | <code>cci_datasource_create()</code> 함수로 datasource를 생성할 때 설정되는 자                                                               |



| 속성 이름                | 타입     | 기본값                            | 의미                                                                                  |
|----------------------|--------|--------------------------------|-------------------------------------------------------------------------------------|
|                      |        |                                | 동 커밋 모드. true 또는 false                                                              |
| default_isolation    | string | cubrid.conf의 isolation_level 값 | <code>cci_datasource_create()</code> 함수로 datasource를 생성할 때 설정되는 트랜잭션 격리 수준. 아래 표 참고 |
| default_lock_timeout | msec   | cubrid.conf의 lock_timeout 값    | <code>cci_datasource_create()</code> 함수로 datasource를 생성할 때 설정되는 잠금 타임아웃             |

prepared statement의 개수가 **max\_open\_prepared\_statement** 값을 초과하면 가장 오래된 prepared statement가 statement pool에서 해제되며, 추후 해제된 prepared statement가 재 사용되면 다시 statement pool에 추가된다.

하나의 연결 풀이 가질 수 있는 연결 개수는 필요 시 `cci_datasource_change_property()` 를 통해 값을 변경할 수 있지만, max\_pool\_size를 초과할 수 없다. 연결 개수의 제한을 조절하고자 할 때 이 값을 변경할 수 있다. 예를 들어 평소에는 max\_pool\_size보다 작게 설정해서 사용하다가 기대 이상으로 많은 연결이 필요해질 때 값을 높이고, 많은 연결이 필요하지 않으면 값을 다시 줄인다.

연결 풀이 가질 수 있는 연결 개수는 pool\_size까지 제한되는데, pool\_size는 최대 max\_pool\_size까지 변경될 수 있다.

**login\_timeout, default\_autocommit, default\_isolation, default\_lock\_timeout**의 값을 설정하면 `cci_datasource_borrow()` 를 호출할 때 이 설정 값을 가지고 연결을 반환한다.

**login\_timeout, default\_autocommit, default\_isolation, default\_lock\_timeout**은 `cci_datasource_borrow()` 함수 호출 이후에도 변경할 수 있다. `cci_set_login_timeout()`, `cci_set_autocommit()`, `cci_set_isolation_level()`, `cci_set_lock_timeout()` 을 참고한다. 이름이 **cci\_set\_**으로 시작하는 함수들에 의해 변경된 값은 변경한 연결 객체에만 적용되며, 연결이 해제된 이후에는 `cci_property_set()` 으로 설정한 값으로 복원된다. `cci_property_set()` 으로 설정한 값이 없을 경우에는 기본값으로 복원된다.

### 참고

- `cci_property_set()` 함수와 URL 문자열에서 동시에 속성 값을 명시하는 경우, `cci_property_set()` 함수에서 정의한 값으로 적용된다.
- **login\_timeout**은 DATASOURCE 객체를 생성하는 경우와 `cci_prepare()` 또는 `cci_execute()` 함수에서 내부적인 재접속이 발생하는 경우에 적용된다. 내부적인 재접속은 DATASOURCE에서 연결 객체를 얻은 경우와 `cci_connect()` / `cci_connect_with_url()` 함수로 연결 객체를 얻은 경우에서 모두 발생한다.
- DATASOURCE 객체를 생성하는 시간은 내부적인 재접속 시간보다 오래 걸릴 수 있으므로, 두 가지 경우의 **login\_timeout**을 다르게 적용하는 것을 감안할 수 있다. 예를 들어, 전자의 경우를 5000(5초), 후자의 경우를 2000(2초)로 설정하고 싶다면, 후자의 설정을 위해 DATASOURCE 생성 후 `cci_set_login_timeout()` 함수를 사용한다.

**default\_isolation**은 다음 값 중 하나의 설정값을 가지며, 격리 수준에 대한 자세한 내용은 [트랜잭션 격리 수준 설정](#)을 참조한다.

| isolation_level | Configuration Value   |
|-----------------|-----------------------|
| SERIALIZABLE    | "TRAN_SERIALIZABLE"   |
| REPEATABLE READ | "TRAN_REP_READ"       |
| READ COMMITTED  | "TRAN_READ_COMMITTED" |

DB 사용자 이름과 암호는 **user** 와 **password** 값을 직접 지정하거나 **url** 속성 내에서 **user** 와 **password** 값을 지정한다. 두 가지 경우가 같이 사용되는 경우 우선 순위는 다음과 같다.

The following shows how to work as the first priority if both are specified.

- 둘 다 입력되면 직접 지정하는 값이 우선한다.
- 둘 중 하나가 NULL이면 NULL이 아닌 값이 사용된다.
- 둘 다 NULL이면 NULL 값으로 사용된다.
- DB 사용자를 직접 지정한 값이 NULL이면 "public", 암호를 직접 지정한 값이 NULL이면 NULL로 설정된다.

- 암호를 직접 지정한 값이 NULL이면 URL의 설정을 따른다.

다음 예제에서 DB 사용자 이름은 “dba”, 암호는 “cubridpwd” 가 된다.

```
cci_property_set(ps, "user", "dba");
cci_property_set(ps, "password", "cubridpwd");
...
cci_property_set(ps, "url", "cci:cubrid:
192.168.0.1:33000:demodb:dba:mypwd:
logSlowQueries=true&slowQueryThresholdMillis=1000&logTraceApi=true&logTraceNetwork=true");
```

### ! 더 보기

```
cci_property_create(), cci_property_destroy(), cci_property_get(), cci_property_set(),
cci_datasource_create(), cci_datasource_destroy(), cci_datasource_release()
```

## cci\_query\_result\_free

int cci\_query\_result\_free(T\_CCI\_QUERY\_RESULT \*query\_result, int num\_query)

`cci_execute_batch()`, `cci_execute_array()` 또는 `cci_execute_result()` 함수에 의해 수행된 질의 결과를 메모리에서 해제한다.

매개변수:

- **query\_result** – (IN) 메모리에서 해제할 질의 결과
- **num\_query** – (IN) `query_result` 의 array 개수

반환:

0: 성공

```
T_CCI_QUERY_RESULT *qr;
char **sql_stmt;

...

res = cci_execute_array(conn, &qr, &err_buf);

...
```

```
cci_query_result_free(qr, res);
```

## CCI\_QUERY\_RESULT\_ERR\_NO

### CCI\_QUERY\_RESULT\_ERR\_NO(query\_result, index)

`cci_execute_batch()`, `cci_execute_array()`, 또는 `cci_execute_result()` 함수에 의해 수행된 질의 결과는 **T\_CCI\_QUERY\_RESULT** 타입의 배열로 저장되므로 배열의 항목 별로 질의 결과를 확인해야 한다.

**CCI\_QUERY\_RESULT\_ERR\_NO**는 *index*로 지정한 배열 항목에 대한 에러 번호를 가져오며, 에러가 아닌 경우 0을 반환한다.

매개변수:

- **query\_result** (T\_CCI\_QUERY\_RESULT\*) – (IN) 조회할 질의 결과
- **index** (int) – (IN) 결과 배열의 인덱스(base : 1). 결과 배열 중 특정 위치를 나타냄.

반환:

에러 번호

## CCI\_QUERY\_RESULT\_ERR\_MSG

### CCI\_QUERY\_RESULT\_ERR\_MSG(query\_result, index)

`cci_execute_batch()`, `cci_execute_array()` 또는 `cci_execute_result()` 함수에 의해 수행된 질의 결과에 대한 에러 메시지를 가져오며, 에러 메시지가 없을 경우 ""(empty string)을 반환하는 매크로이다. 지정된 인자 *query\_result* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

매개변수:

- **query\_result** (T\_CCI\_QUERY\_RESULT\*) – (IN) 조회할 질의 결과
- **index** (int) – (IN) 칼럼 인덱스(base : 1)

반환:

에러 메시지

## CCI\_QUERY\_RESULT\_RESULT

### CCI\_QUERY\_RESULT\_RESULT(query\_result, index)

`cci_execute_batch()`, `cci_execute_array()` 또는 `cci_execute_result()` 함수에 의해 수행된 질의 결과에 대한 result count를 가져오는 매크로이다. 지정된 인자 *query\_result* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

매개변수:

- **query\_result** (T\_CCI\_QUERY\_RESULT\*) – (IN) 조회할 질의 결과
- **index** (int) – (IN) 칼럼 인덱스(base : 1)

반환:

result count

## CCI\_QUERY\_RESULT\_STMT\_TYPE

### CCI\_QUERY\_RESULT\_STMT\_TYPE(query\_result, index)

`cci_execute_batch()`, `cci_execute_array()` 또는 `cci_execute_result()` 함수에 의해 수행된 질의 결과는 **T\_CCI\_QUERY\_RESULT** 타입의 배열로 저장되므로 배열의 항목 별로 질의 결과를 확인해야 한다.

**CCI\_QUERY\_RESULT\_STMT\_TYPE**은 index로 지정한 배열 항목에 대한 statement type을 가져오는 매크로이다.

지정된 인자 *query\_result* 가 **NULL** 인지, *index* 가 유효한지에 대한 검사는 하지 않는다.

매개변수:

- **query\_result** (T\_CCI\_QUERY\_RESULT\*) – (IN) 조회할 질의 결과
- **index** (int) – (IN) 칼럼 인덱스(base : 1)

반환:

statement type (**T\_CCI\_CUBRID\_STMT**)

## cci\_register\_out\_param

int cci\_register\_out\_param(int req\_handle, int index)

저장 프로시저에서 아웃 바인드로 정의한 파라미터를 바인딩하기 위해 사용하며, index는 1부터 시작한다. 이 함수를 호출하기 전에 cci\_prepare 함수의 플래그에 **CCI\_PREPARE\_CALL**을 설정해야 한다.

매개변수:

- **req\_handle** – (IN) 요청 핸들

반환:

에러 코드

- **CCI\_ER\_BIND\_INDEX**
- **CCI\_ER\_REQ\_HANDLE**
- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**

다음은 “Hello, CUBRID”라는 문자열을 출력하는 저장 프로시저의 예이다.

```
$ vi cubrid.conf

$ cubrid service start
$ cubrid server start demodb
```

저장 프로시저로 등록할 클래스를 구현하고 컴파일한다.

```
public class SpCubrid{
 public static void outTest(String[] o) {
 o[0] = "Hello, CUBRID";
 }
}

%javac SpCubrid.java
```

컴파일된 자바 클래스를 CUBRID로 로딩한다.

```
$ loadjava demodb SpCubrid.class
```

로딩한 Java 클래스를 등록한다.

```
-- csql>

CREATE PROCEDURE test_out(x OUT STRING)
AS LANGUAGE JAVA
NAME 'SpCubrid.outTest(java.lang.String[] o)';
```

CCI 응용 프로그램에서 cci\_prepare 함수의 플래그에 CCI\_PREPARE\_CALL을 지정하고, cci\_register\_out\_param 함수로 아웃 바인드 파라미터 위치를 지정한 후 실행 결과를 폐치하여 가져온다.

```
// On Linux, compile with "gcc -g -o jsp jsp.c -I$CUBRID/
include/ -L$CUBRID/lib/ -lcascci -lpthread"

#include <stdio.h>
#include <unistd.h>
#include <cas_cci.h>
char *cci_client_name = "test";

int
main (int argc, char *argv[])
{
 int con = 0, req = 0, res, ind, i, col_count;
 T_CCI_ERROR error;
 T_CCI_COL_INFO *res_col_info;
 T_CCI_CUBRID_STMT cmd_type;
 char *buffer, db_ver[16];

 if ((con = cci_connect ("localhost", 33000, "demodb", "dba",
"")) < 0)
 {
 printf ("%s(%d): cci_connect fail\n", __FILE__,
__LINE__);
```

```

 return -1;
 }

 if ((req = cci_prepare (con, "call test_out(?)",
CCI_PREPARE_CALL, &error)) < 0)
 {
 printf ("%s(%d): cci_prepare fail(%d)\n", __FILE__,
__LINE__,
 error.err_code);
 goto handle_error;
 }
 if ((res = cci_register_out_param (req, 1)) < 0)
 {
 printf ("%s(%d): cci_register_out_param fail(%d)\n",
__FILE__, __LINE__,
 error.err_code);
 goto handle_error;
 }
 if ((res = cci_execute (req, 0, 0, &error)) < 0)
 {
 printf ("%s(%d): cci_execute fail(%d)\n", __FILE__,
__LINE__,
 error.err_code);
 goto handle_error;
 }
 res = cci_cursor (req, 1, CCI_CURSOR_CURRENT, &error);
 if (res == CCI_ER_NO_MORE_DATA)
 {
 printf ("%s(%d): cci_cursor fail(%d)\n", __FILE__,
__LINE__,
 error.err_code);
 goto handle_error;
 }

 if ((res = cci_fetch (req, &error) < 0))
 {
 printf ("%s(%d): cci_fetch(%d, %s)\n", __FILE__,
__LINE__,
 error.err_code, error.err_msg);
 goto handle_error;
 }
 if ((res = cci_get_data (req, 1, CCI_A_TYPE_STR, &buffer,
&ind)) < 0)
 {
 printf ("%s(%d): cci_get_data fail\n", __FILE__,
__LINE__);
 goto handle_error;
 }
 // ind: string length, buffer: a string which came from the
 out binding parameter of test_out(?) Java SP.
 if (ind != -1)
 printf ("%d, (%s)\n", ind, buffer);

```



```

else // if ind== -1, then data is NULL
 printf ("NULL\n");

 if ((res = cci_close_query_result (req, &error)) < 0)
 {
 printf ("%s(%d): cci_close_query_result fail(%d)\n",
__FILE__, __LINE__, error.err_code);
 goto handle_error;
 }

 if ((res = cci_close_req_handle (req)) < 0)
 {
 printf ("%s(%d): cci_close_req_handle fail\n", __FILE__,
__LINE__);
 goto handle_error;
 }

 if ((res = cci_disconnect (con, &error)) < 0)
 {
 printf ("%s(%d): cci_disconnect fail(%d)\n", __FILE__,
__LINE__,
 error.err_code);
 goto handle_error;
 }
 printf ("Program ended!\n");
 return 0;

handle_error:
 if (req > 0)
 cci_close_req_handle (req);
 if (con > 0)
 cci_disconnect (con, &error);
 printf ("Program failed!\n");
 return -1;
}

```

## ❶ 더 보기

cci\_prepare()

## cci\_row\_count

int cci\_row\_count(int conn\_handle, int \*row\_count, T\_CCI\_ERROR \*err\_buf)

가장 최근 수행한 질의에 의해 영향을 받은 행의 개수를 얻는다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **row\_count** – (OUT) 가장 최근 수행한 질의에 의해 영향을 받은 행의 개수
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_LOGIN\_TIMEOUT**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_DBMS**

## cci\_savepoint

```
int cci_savepoint(int conn_handle, T_CCI_SAVEPOINT_CMD cmd, char *savepoint_name,
T_CCI_ERROR *err_buf)
```

세이브포인트를 설정하거나 설정된 세이브포인트로 트랜잭션 롤백을 수행한다. *cmd* 가 **CCI\_SP\_SET** 로 지정되면 세이브포인트를 설정하고, **CCI\_SP\_ROLLBACK** 인 경우 설정된 세이브 포인트까지 트랜잭션을 롤백한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **cmd** – (IN) **CCI\_SP\_SET** 또는 **CCI\_SP\_ROLLBACK**
- **savepoint\_name** – (IN) 세이브포인트 이름
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**

- CCI\_ER\_COMMUNICATION
- CCI\_ER\_QUERY\_TIMEOUT
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_DBMS

```
con = cci_connect(...);
... /* query execute */

/* "savepoint1"이란 이름의 세이브포인트 설정 */
cci_savepoint(con, CCI_SP_SET, "savepoint1", err_buf);

... /* query execute */

/* 설정된 세이브포인트 "savepoint1"로 롤백 */
cci_savepoint(con, CCI_SP_ROLLBACK, "savepoint1", err_buf);
```

## cci\_schema\_info

int cci\_schema\_info(int conn\_handle, T\_CCI\_SCHEMA\_TYPE type, char \*class\_name, char \*attr\_name, char flag, T\_CCI\_ERROR \*err\_buf)

스키마 정보를 읽어온다. 성공적으로 수행되었을 경우 결과는 request handle에 의해 관리되고, fetch, getdata를 통해서 결과를 가져올 수 있다. *class\_name*, *attr\_name* 을 **LIKE** 절의 패턴 매칭에 의해서 검색해야 할 경우 *flag* 를 설정한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **type** – (IN) 스키마 타입
- **class\_name** – (IN) 클래스 이름 또는 NULL
- **attr\_name** – (IN) 속성 이름 또는 NULL
- **flag** – (IN) 패턴 매칭  
flag(CCI\_CLASS\_NAME\_PATTERN\_MATCH 또는 CCI\_ATTR\_NAME\_PATTERN\_MATCH)
- **err\_buf** – (OUT) 에러 버퍼

반환:

성공 : 요청 핸들, 실패 : 에러 코드

- CCI\_ER\_CON\_HANDLE
- CCI\_ER\_DBMS
- CCI\_ER\_COMMUNICATION
- CCI\_ER\_NO\_MORE\_MEMORY
- CCI\_ER\_CONNECT

**CCI\_CLASS\_NAME\_PATTERN\_MATCH**, **CCI\_ATTR\_NAME\_PATTERN\_MATCH** 두 개의 *flag* 가 사용되며, OR 연산자( | )로 둘 다 설정할 수 있다. 패턴 매칭을 사용할 경우 **LIKE** 절을 이용하여 검색한다. 예를 들어, *class\_name* 이 “athlete”이고 *attr\_name* 이 “code”로 끝나는 칼럼에 대한 정보를 검색하고 싶다면, *attr\_name* 의 값을 “%code”로 하여 다음과 같이 입력할 수 있다.

```
cci_schema_info(conn, CCI_SCH_ATTRIBUTE, "athlete", "%code",
CCI_ATTR_NAME_PATTERN_MATCH, &error);
```

각 *type* 에 대한 레코드의 구성은 아래 표와 같다.

#### 타입에 대한 레코드 구성

| 타입             | 칼럼 순서 | 칼럼 이름   | 칼럼 타입                                                                                                                                  |
|----------------|-------|---------|----------------------------------------------------------------------------------------------------------------------------------------|
| CCI_SCH_CLASS  | 1     | NAME    | char *                                                                                                                                 |
|                | 2     | TYPE    | short <ul style="list-style-type: none"> <li>• 0: system class</li> <li>• 1: vclass</li> <li>• 2: class</li> <li>• 3: proxy</li> </ul> |
|                | 3     | REMARKS | char *                                                                                                                                 |
| CCI_SCH_VCLASS | 1     | NAME    | char *                                                                                                                                 |

| 타입                 | 칼럼 순서 | 칼럼 이름      | 칼럼 타입                                                                                   |
|--------------------|-------|------------|-----------------------------------------------------------------------------------------|
| CCI_SCH_QUERY_SPEC | 2     | TYPE       | short <ul style="list-style-type: none"> <li>• 1: vclass</li> <li>• 3: proxy</li> </ul> |
|                    | 3     | REMARKS    | char *                                                                                  |
|                    | 1     | QUERY_SPEC | char *                                                                                  |
|                    | 1     | NAME       | char *                                                                                  |
|                    | 2     | DOMAIN     | short                                                                                   |
|                    | 3     | SCALE      | short                                                                                   |
|                    | 4     | PRECISION  | int                                                                                     |
|                    | 5     | INDEXED    | short <ul style="list-style-type: none"> <li>1: indexed</li> </ul>                      |
| CCI_SCH_ATTRIBUTE  | 6     | NON_NULL   | short <ul style="list-style-type: none"> <li>1: non null</li> </ul>                     |
|                    | 7     | SHARED     | short <ul style="list-style-type: none"> <li>1: shared</li> </ul>                       |
|                    | 8     | UNIQUE     | short <ul style="list-style-type: none"> <li>1: unique</li> </ul>                       |

| 타입 | 칼럼 순서 | 칼럼 이름        | 칼럼 타입           |
|----|-------|--------------|-----------------|
|    | 9     | DEFAULT      | char *          |
|    | 10    | ATTR_ORDER   | int<br>base = 1 |
|    | 11    | CLASS_NAME   | char *          |
|    | 12    | SOURCE_CLASS | char *          |
|    | 13    | IS_KEY       | short<br>1: key |
|    | 14    | REMARKS      | char *          |

CCI\_SCH\_CLASS\_ATTR  
IBUTE

CCI\_SCH\_CLASS\_ATTR  
IBUTE 컬럼이  
INSTANCE 또는  
SHARED의 속성일 경  
우에  
순서와 이름 값은  
CCI\_SCH\_ATTRIBUTE  
의 컬럼과 동일하다.

CCI\_SCH\_CLASS\_MET  
HOD

|   |            |        |
|---|------------|--------|
| 1 | NAME       | char * |
| 2 | RET_DOMAIN | short  |
| 3 | ARG_DOMAIN | char * |

| 타입                   | 칼럼 순서 | 칼럼 이름       | 칼럼 타입                                                                                                                                            |
|----------------------|-------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| CCI_SCH_METHOD_FILE  | 1     | METHOD_FILE | char *                                                                                                                                           |
|                      | 2     | TYPE        | short                                                                                                                                            |
| CCI_SCH_SUPERCLASSES | 1     | CLASS_NAME  | char *                                                                                                                                           |
|                      | 2     | TYPE        | short                                                                                                                                            |
| CCI_SCH_SUBCLASS     | 1     | CLASS_NAME  | char *                                                                                                                                           |
|                      | 2     | TYPE        | short                                                                                                                                            |
| CCI_SCH_CONSTRAINT   | 1     | TYPE        | short <ul style="list-style-type: none"> <li>• 0: unique</li> <li>• 1: index</li> <li>• 2: reverse unique</li> <li>• 3: reverse index</li> </ul> |
|                      | 2     | NAME        | char *                                                                                                                                           |
|                      | 3     | ATTR_NAME   | char *                                                                                                                                           |
|                      | 4     | NUM_PAGES   | int                                                                                                                                              |
|                      | 5     | NUM_KEYS    | int                                                                                                                                              |
|                      | 6     | PRIMARY_KEY | short <ul style="list-style-type: none"> <li>• 1: primary key</li> </ul>                                                                         |
|                      |       |             |                                                                                                                                                  |
|                      |       |             |                                                                                                                                                  |

| 타입                          | 칼럼 순서 | 칼럼 이름          | 칼럼 타입             |
|-----------------------------|-------|----------------|-------------------|
| CCI_SCH_TRIGGER             | 7     | KEY_ORDER      | short<br>base = 1 |
|                             | 8     | ASC_DESC       | char *            |
|                             | 1     | NAME           | char *            |
|                             | 2     | STATUS         | char *            |
|                             | 3     | EVENT          | char *            |
|                             | 4     | TARGET_CLASS   | char *            |
|                             | 5     | TARGET_ATTR    | char *            |
|                             | 6     | ACTION_TIME    | char *            |
|                             | 7     | ACTION         | char *            |
|                             | 8     | PRIORITY       | float             |
|                             | 9     | CONDITION_TIME | char *            |
|                             | 10    | CONDITION      | char *            |
|                             | 11    | REMARKS        | char *            |
| CCI_SCH_CLASS_PRIV<br>ILEGE | 1     | CLASS_NAME     | char *            |



| 타입                                            | 칼럼 순서 | 칼럼 이름            | 칼럼 타입           |
|-----------------------------------------------|-------|------------------|-----------------|
|                                               | 2     | PRIVILEGE        | char *          |
|                                               | 3     | GRANTABLE        | char *          |
|                                               |       |                  |                 |
| CCI_SCH_ATTR_PRIVILEGE                        | 1     | ATTR_NAME        | char *          |
|                                               | 2     | PRIVILEGE        | char *          |
|                                               | 3     | GRANTABLE        | char *          |
| CCI_SCH_DIRECT_SUPER_CLASS                    | 1     | CLASS_NAME       | char *          |
|                                               | 2     | SUPER_CLASS_NAME | char *          |
| CCI_SCH_PRIMARY_KEY                           | 1     | CLASS_NAME       | char *          |
|                                               | 2     | ATTR_NAME        | char *          |
|                                               | 3     | KEY_SEQ          | int<br>base = 1 |
|                                               | 4     | KEY_NAME         | char *          |
| CCI_SCH_IMPORTED_KEYS                         | 1     | PKTABLE_NAME     | char *          |
| 주어진 테이블의 외래 키 칼럼들이 참조하고 있는 기본 키 칼럼들의 정보를 조회하며 |       |                  |                 |

| 타입                                                                                                                                                    | 칼럼 순서 | 칼럼 이름         | 칼럼 타입                                                                                                                                      |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 결과는<br>PKTABLE_NAME 및<br>KEY_SEQ 순서로 정렬<br>된다.<br>이 타입을 인자로 지<br>정하면, <i>class_name</i><br>에는 외래 키 테이블,<br><i>attr_name</i> 에는 <b>NULL</b><br>을 지정한다. | 2     | PKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                       | 3     | FKTABLE_NAME  | char *                                                                                                                                     |
|                                                                                                                                                       | 4     | FKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                       | 5     | KEY_SEQ       | short                                                                                                                                      |
|                                                                                                                                                       | 6     | UPDATE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                       | 7     | DELETE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                       | 8     | FK_NAME       | char *                                                                                                                                     |
|                                                                                                                                                       | 9     | PK_NAME       | char *                                                                                                                                     |
|                                                                                                                                                       |       |               |                                                                                                                                            |

CCI\_SCH\_EXPORTED\_  
KEYS

주어진 테이블의 기  
본 키 칼럼들을 참조

| 타입                                                                                                                                                        | 칼럼 순서 | 칼럼 이름         | 칼럼 타입                                                                                                                                      |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <p>하는 모든 외래 키 칼럼들의 정보를 조회하며, 결과는 FKTABLE_NAME 및 KEY_SEQ 순서로 정렬된다. 이 타입을 인자로 지정하면, <i>class_name</i>에는 기본 키 테이블, <i>attr_name</i>에는 <b>NULL</b>을 지정한다.</p> | 2     | PKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                           | 3     | FKTABLE_NAME  | char *                                                                                                                                     |
|                                                                                                                                                           | 4     | FKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                           | 5     | KEY_SEQ       | short                                                                                                                                      |
|                                                                                                                                                           | 6     | UPDATE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                           | 7     | DELETE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                           | 8     | FK_NAME       | char *                                                                                                                                     |
|                                                                                                                                                           | 9     | PK_NAME       | char *                                                                                                                                     |
| CCI_SCH_CROSS_REFERENCE                                                                                                                                   | 1     | PKTABLE_NAME  | char *                                                                                                                                     |
| 주어진 테이블의 기본 키와 주어진 테이블                                                                                                                                    |       |               |                                                                                                                                            |

| 타입                                                                                                                                               | 칼럼 순서 | 칼럼 이름         | 칼럼 타입                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| 불의 외래 키가 상호 참조하는 경우, 해당 외래 키 칼럼들의 정보를 조회하며, 결과는 FKTABLE_NAME 및 KEY_SEQ 순서로 정렬된다.<br>이 타입을 인자로 class_name에는 기본 키 테이블, attr_name에는 외래 키 테이블을 지정한다. | 2     | PKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                  | 3     | FKTABLE_NAME  | char *                                                                                                                                     |
|                                                                                                                                                  | 4     | FKCOLUMN_NAME | char *                                                                                                                                     |
|                                                                                                                                                  | 5     | KEY_SEQ       | short                                                                                                                                      |
|                                                                                                                                                  | 6     | UPDATE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                  | 7     | DELETE_ACTION | short <ul style="list-style-type: none"> <li>• 0: cascade</li> <li>• 1: restrict</li> <li>• 2: no action</li> <li>• 3: set null</li> </ul> |
|                                                                                                                                                  | 8     | FK_NAME       | char *                                                                                                                                     |
|                                                                                                                                                  | 9     | PK_NAME       | char *                                                                                                                                     |

`cci_schema_info()` 함수에서 인자 *type* 은 인자 *class\_name*, *attr\_name* 에 대해 **LIKE** 절의 패턴 매칭을 지원한다.

패턴 매칭을 지원하는 **type** 및 **class\_name**(테이블 이름), **attr\_name**(칼럼 이름) 입력

| type                                   | class_name  | attr_name   |
|----------------------------------------|-------------|-------------|
| CCI_SCH_CLASS (VCLASS)                 | 문자열 또는 NULL | 항상 NULL     |
| CCI_SCH_ATTRIBUTE (CLASS<br>ATTRIBUTE) | 문자열 또는 NULL | 문자열 또는 NULL |
| CCI_SCH_CLASS_PRIVILEGE                | 문자열 또는 NULL | 항상 NULL     |
| CCI_SCH_ATTR_PRIVILEGE                 | 항상 NULL     | 문자열 또는 NULL |
| CCI_SCH_PRIMARY_KEY                    | 문자열 또는 NULL | 항상 NULL     |
| CCI_SCH_TRIGGER                        | 문자열 또는 NULL | 항상 NULL     |

- class\_name이 “항상 NULL”인 type은 테이블 이름에 대해 패턴 매칭을 지원하지 않는다.
- attr\_name이 “항상 NULL”인 type은 칼럼 이름에 대해 패턴 매칭을 지원하지 않는다.
- 패턴 *flag* 가 설정되지 않을 경우 주어진 테이블 이름 또는 칼럼 이름은 동등 조건(=) 매칭을 사용하므로 **NULL** 이 주어지면 결과는 없다.
- 패턴 *flag* 가 설정되어 있을 경우 테이블 이름 또는 칼럼 이름이 **NULL**이면 **LIKE** 절의 “%”와 동일한 결과, 즉 모든 테이블 또는 칼럼에 대한 결과를 얻는다.

#### 참고

**CCI\_SCH\_CLASS, CCI\_SCH\_VCLASS** 의 TYPE 칼럼 : proxy 타입이 추가됨. OLEDB, ODBC, PHP 에서 사용할 경우 proxy와 vclass를 구분하지 않고 vclass로 보임.

```
// gcc -o schema_info schema_info.c -m64 -I${CUBRID}/include -
lnsl ${CUBRID}/lib/libcascci.so -lpthread

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "cas_cci.h"
```

```

int
main ()
{
 int conn = 0, req = 0, col_count = 0, res = 0, i, ind;
 char *data, query_result_buffer[1024];
 T_CCI_ERROR cci_error;
 T_CCI_COL_INFO *col_info;
 T_CCI_CUBRID_STMT cmd_type;

 conn = cci_connect ("localhost", 33000, "demodb", "dba", "");
 // get all columns' information of table "athlete"
 // req = cci_schema_info(conn, CCI_SCH_ATTRIBUTE, "athlete",
"code", 0, &cci_error);
 req =
 cci_schema_info (conn, CCI_SCH_ATTRIBUTE, "athlete", NULL,
 CCI_ATTR_NAME_PATTERN_MATCH, &cci_error);

 if (req < 0)
 {
 fprintf (stdout, "(%s, %d) ERROR : %s [%d] \n\n",
__FILE__, __LINE__,
 cci_error.err_msg, cci_error.err_code);
 goto _END;
 }
 col_info = cci_get_result_info (req, &cmd_type, &col_count);
 if (!col_info && col_count == 0)
 {
 fprintf (stdout, "(%s, %d) ERROR :
cci_get_result_info\n\n", __FILE__,
 __LINE__);
 goto _END;
 }
 res = cci_cursor (req, 1, CCI_CURSOR_FIRST, &cci_error);
 if (res == CCI_ER_NO_MORE_DATA)
 {
 goto _END;
 }
 if (res < 0)
 {
 fprintf (stdout, "(%s, %d) ERROR : %s [%d] \n\n",
__FILE__, __LINE__,
 cci_error.err_msg, cci_error.err_code);
 goto _END;
 }

 while (1)
 {
 res = cci_fetch (req, &cci_error);
 if (res < 0)

```

```

 {
 fprintf (stdout, "(%s, %d) ERROR : %s [%d] \n\n",
__FILE__,
 __LINE__, cci_error.err_msg,
cci_error.err_code);
 goto _END;
 }

 for (i = 1; i <= col_count; i++)
 {
 if ((res = cci_get_data (req, i, CCI_A_TYPE_STR,
&data, &ind)) < 0)
 {
 goto _END;
 }
 if (ind != -1)
 {
 strcat (query_result_buffer, data);
 strcat (query_result_buffer, "|");
 }
 else
 {
 strcat (query_result_buffer, "NULL|");
 }
 }
 strcat (query_result_buffer, "\n");

 res = cci_cursor (req, 1, CCI_CURSOR_CURRENT, &cci_error);
 if (res == CCI_ER_NO_MORE_DATA)
 {
 goto _END;
 }
 if (res < 0)
 {
 fprintf (stdout, "(%s, %d) ERROR : %s [%d] \n\n",
__FILE__,
 __LINE__, cci_error.err_msg,
cci_error.err_code);
 goto _END;
 }
 }

_END:
 if (req > 0)
 cci_close_req_handle (req);
 if (conn > 0)
 res = cci_disconnect (conn, &cci_error);
 if (res < 0)
 fprintf (stdout, "(%s, %d) ERROR : %s [%d] \n\n", __FILE__,
__LINE__,
 cci_error.err_msg, cci_error.err_code);

```

```

 fprintf (stdout, "Result : %s\n", query_result_buffer);

 return 0;
}

```

## cci\_set\_allocators

int cci\_set\_allocators(CCI\_MALLOC\_FUNCTION malloc\_func, CCI\_FREE\_FUNCTION free\_func, CCI\_REALLOC\_FUNCTION realloc\_func, CCI\_CALLOC\_FUNCTION calloc\_func)

사용자가 정의한 메모리 할당 및 해제 함수들을 등록하여 사용할 수 있게 한다. 이 함수를 실행하면 CCI API 내부에서 사용되는 모든 메모리 할당 및 해제 작업은 사용자가 정의한 메모리 할당 함수들을 사용하게 된다. 이를 사용하지 않으면 기본적인 시스템 함수들(malloc, free, realloc, calloc)이 사용된다.

### 참고

이 함수는 Linux 전용으로, Windows에서는 사용할 수 없다.

매개변수:

- **malloc\_func** – (IN) malloc에 해당하는 외부 정의 함수에 대한 포인터
- **free\_func** – (IN) free에 해당하는 외부 정의 함수에 대한 포인터
- **realloc\_func** – (IN) realloc에 해당하는 외부 정의 함수에 대한 포인터
- **calloc\_func** – calloc에 해당하는 외부 정의 함수에 대한 포인터

반환:

에러 코드(0: 성공)

- **CCI\_ER\_NOT\_IMPLEMENTED**

```

/*
 How to build: gcc -Wall -g -o test_cci test_cci.c -I$
 {CUBRID}/include -L${CUBRID}/lib -lcascci

```



```

*/

#include <stdio.h>
#include <stdlib.h>
#include "cas_cci.h"

void *my_malloc(size_t size)
{
 printf ("my malloc: size: %ld\n", size);
 return malloc (size);
}

void *my_calloc(size_t nm, size_t size)
{
 printf ("my calloc: nm: %ld, size: %ld\n", nm, size);
 return calloc (nm, size);
}

void *my_realloc(void *ptr, size_t size)
{
 printf ("my realloc: ptr: %p, size: %ld\n", ptr, size);
 return realloc (ptr, size);
}

void my_free(void *ptr)
{
 printf ("my free: ptr: %p\n", ptr);
 return free (ptr);
}

int test_simple (int con)
{
 int req = 0, col_count = 0, i, ind;
 int error;
 char *data;
 T_CCI_ERROR cci_error;
 T_CCI_COL_INFO *col_info;
 T_CCI_CUBRID_STMT stmt_type;
 char *query = "select * from db_class";

 //preparing the SQL statement
 req = cci_prepare (con, query, 0, &cci_error);
 if (req < 0)
 {
 printf ("prepare error: %d, %s\n", cci_error.err_code,
 cci_error.err_msg);
 goto handle_error;
 }

 //getting column information when the prepared statement is
 the SELECT query

```

```

col_info = cci_get_result_info (req, &stmt_type, &col_count);
if (col_info == NULL)
{
 printf ("get_result_info error\n");
 goto handle_error;
}

//Executing the prepared SQL statement
error = cci_execute (req, 0, 0, &cci_error);
if (error < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
}
while (1)
{
 //Moving the cursor to access a specific tuple of results
 error = cci_cursor (req, 1, CCI_CURSOR_CURRENT,
&cci_error);
 if (error == CCI_ER_NO_MORE_DATA)
 {
 break;
 }
 if (error < 0)
 {
 printf ("cursor error: %d, %s\n",
cci_error.err_code, cci_error.err_msg);
 goto handle_error;
 }

 //Fetching the query result into a client buffer
 error = cci_fetch (req, &cci_error);
 if (error < 0)
 {
 printf ("fetch error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
 goto handle_error;
 }
 for (i = 1; i <= col_count; i++)
 {
 //Getting data from the fetched result
 error = cci_get_data (req, i, CCI_A_TYPE_STR, &data,
&ind);
 if (error < 0)
 {
 printf ("get_data error: %d, %d\n", error, i);
 goto handle_error;
 }
 printf ("%s\t", data);
 }
}

```

```

 printf ("\n");
 }

 //Closing the query result
 error = cci_close_query_result(req, &cci_error);
 if (error < 0)
 {
 printf ("cci_close_query_result error: %d, %s\n",
cci_error.err_code, cci_error.err_msg);
 goto handle_error;
 }

 //Closing the request handle
 error = cci_close_req_handle(req);
 if (error < 0)
 {
 printf ("cci_close_req_handle error\n");
 goto handle_error;
 }

 //Disconnecting with the server
 error = cci_disconnect (con, &cci_error);
 if (error < 0)
 {
 printf ("cci_disconnect error: %d, %s\n",
cci_error.err_code, cci_error.err_msg);
 goto handle_error;
 }
 return 0;

handle_error:
 if (req > 0)
 cci_close_req_handle (req);
 if (con > 0)
 cci_disconnect (con, &cci_error);

 return 1;
}

int main()
{
 int con = 0;

 if (cci_set_allocators (my_malloc, my_free, my_realloc,
my_calloc) != 0)
 {
 printf ("cannot register allocators\n");
 return 1;
 };

 //getting a connection handle for a connection with a server
 con = cci_connect ("localhost", 33000, "demodb", "dba", "");

```

```

 if (con < 0)
 {
 printf ("cannot connect to database\n");
 return 1;
 }

 test_simple (con);
 return 0;
}

```

## cci\_set\_autocommit

int cci\_set\_autocommit(int conn\_handle, CCI\_AUTOCOMMIT\_MODE autocommit\_mode)

현재 데이터베이스 연결의 자동 커밋 모드 여부를 설정한다. 이 함수는 자동 커밋 모드를 설정하거나 해제하는 데에만 사용되며, 이 함수를 호출하면 진행 중이던 트랜잭션은 모드 설정과 상관없이 커밋된다.

### 참고

**cubrid\_broker.conf**의 **CCI\_DEFAULT\_AUTOCOMMIT**은 프로그램 시작 시의 기본 자동 커밋 모드를 설정한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **autocommit\_mode** – (IN) 자동 커밋모드 설정.  
CCI\_AUTOCOMMIT\_FALSE 또는 CCI\_AUTOCOMMIT\_TRUE 중 하나의 값을 가진다.

반환:

에러 코드(0: 성공)

### 참고

**CCI\_DEFAULT\_AUTOCOMMIT**, a broker parameter configured in the **cubrid\_broker.conf** file, determines whether it is in auto-commit mode upon program startup.

## cci\_set\_db\_parameter

```
int cci_set_db_parameter(int conn_handle, T_CCI_DB_PARAM param_name, void *value,
T_CCI_ERROR *err_buf)
```

시스템 파라미터를 설정한다. 설정할 수 있는 *param\_name* 은 `cci_get_db_parameter()` 를 참조한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **param\_name** – (IN) 시스템 파라미터 이름
- **value** – (IN) 파라미터 값
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_PARAM\_NAME**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

## cci\_set\_element\_type

```
int cci_set_element_type(T_CCI_SET set)
T_CCI_SET 타입 값의 엘리먼트 타입을 가져온다.
```

매개변수:

- **set** – (IN) cci set 포인터

반환:

타입

## cci\_set\_free

void cci\_set\_free(T\_CCI\_SET set)

`cci_get_data()`에 대해 **CCI\_A\_TYPE\_SET**에 의해 가져온 **T\_CCI\_SET** 타입 값에 할당된 메모리를 제거한다. **T\_CCI\_SET** 타입 값은 `cci_get_data()` 함수를 통해 가져오거나 `cci_set_make()` 함수를 통해 생성할 수 있다.

매개변수:

- **set** – (IN) cci set 포인터

## cci\_set\_get

int cci\_set\_get(T\_CCI\_SET set, int index, T\_CCI\_A\_TYPE a\_type, void \*value, int \*indicator)

**T\_CCI\_SET** 타입 값에 대해 *index* 번째의 데이터를 가져온다.

매개변수:

- **set** – (IN) cci set 포인터
- **index** – (IN) set index (base: 1)
- **a\_type** – (IN) 타입
- **value** – (OUT) 결과 버퍼
- **indicator** – (OUT) null 표시(indicator)

반환:

에러 코드

- **CCI\_ER\_SET\_INDEX**
- **CCI\_ER\_TYPE\_CONVERSION**
- **CCI\_ER\_NO\_MORE\_MEMORY**
- **CCI\_ER\_COMMUNICATION**

*a\_type*에 대한 *value*의 타입은 다음과 같다.

| a_type            | value 타입                       |
|-------------------|--------------------------------|
| CCI_A_TYPE_STR    | char **                        |
| CCI_A_TYPE_INT    | int *                          |
| CCI_A_TYPE_FLOAT  | float *                        |
| CCI_A_TYPE_DOUBLE | double *                       |
| CCI_A_TYPE_BIT    | T_CCI_BIT *                    |
| CCI_A_TYPE_DATE   | T_CCI_DATE *                   |
| CCI_A_TYPE_BIGINT | int64_t * (Windows는 __int64 *) |

## cci\_set\_holdability

int cci\_set\_holdability(int conn\_handle, int holdable)

연결 수준에서 결과 셋의 커서 유지(cursor holdability) 여부를 설정한다. 값이 1이면 커밋 여부에 관계 없이 연결이 종료되거나 결과 셋을 의도적으로 닫기 전까지 커서를 유지(holdable)하고, 0이면 커밋될 때 결과 셋이 닫히면서 커서를 유지하지 않는다(not holdable). 커서 유지에 대한 자세한 설명은 [커서 유지](#)를 참고한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **holdable** – (IN) 커서 유지 여부 설정 값(0: not holdable, 1: holdable)

반환:

에러 코드

- **CCI\_ER\_INVALID\_HOLDABILITY**

## cci\_set\_isolation\_level

```
int cci_set_isolation_level(int conn_handle, T_CCI_TRAN_ISOLATION new_isolation_level,
T_CCI_ERROR *err_buf)
```

연결에 대한 트랜잭션 격리 수준(transaction isolation level)을 설정한다. 이후 주어진 연결에 대한 모든 트랜잭션은 *new\_isolation\_level* 로서 동작한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **new\_isolation\_level** – (IN) 격리 수준(isolation level)
- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_CONNECT**
- **CCI\_ER\_ISOLATION\_LEVEL**
- **CCI\_ER\_DBMS**

cci\_set\_db\_parameter()에 의해 트랜잭션 격리 수준이 설정된 경우에는 현재의 트랜잭션에 대해서만 영향을 주고 트랜잭션이 끝나면 CAS에서 설정된 트랜잭션 격리 수준으로 복귀된다. 연



결 전체에 대해서 트랜잭션 격리 수준을 설정할 경우는 반드시 `cci_set_isolation_level()`을 사용해야 한다.

## cci\_set\_lock\_timeout

`int cci_set_lock_timeout(int conn_handle, int locktimeout, T_CCI_ERROR *err_buf)`

연결에 대한 잠금 타임아웃을 밀리초 단위로 설정한다. `cci_set_db_parameter(conn_id, CCI_PARAM_LOCK_TIMEOUT, &val, err_buf)`를 호출한 것과 동일하며, `cci_set_db_parameter()` 함수를 참고한다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **locktimeout** – (IN) 잠금 타임아웃 시간(단위: 밀리 초).

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_PARAM\_NAME**
- **CCI\_ER\_DBMS**
- **CCI\_ER\_COMMUNICATION**
- **CCI\_ER\_CONNECT**

## cci\_set\_login\_timeout

`int cci_set_login_timeout(int conn_handle, int timeout, T_CCI_ERROR *err_buf)`

로그인 타임아웃을 밀리초 단위로 설정한다. `cci_prepare()` 함수나 `cci_execute()` 함수를 호출하면서 내부적으로 재접속이 발생할 때 로그인 타임 아웃이 적용된다. 이 변경은 현재의 연결에만 적용된다.

매개변수:

- **conn\_handle** – (IN) 연결 핸들
- **timeout** – (IN) 로그인 타임아웃(단위: 밀리초)

- **err\_buf** – (OUT) 에러 버퍼

반환:

에러 코드(0: 성공)

- **CCI\_ER\_CON\_HANDLE**
- **CCI\_ER\_USED\_CONNECTION**

## cci\_set\_make

int cci\_set\_make(T\_CCI\_SET \*set, T\_CCI\_U\_TYPE u\_type, int size, void \*value, int \*indicator)

새로운 **CCI\_A\_TYPE\_SET** 타입의 set을 만든다. 만들어진 set은 `cci_bind_param()` 을 통해 **CCI\_A\_TYPE\_SET** 으로 서버에 전달된다. `cci_set_make()` 에 의해 만들어진 set은 반드시 `cci_set_free()` 를 통해 사용된 메모리를 제거해야 한다. `u_type` 에 따른 `value` 의 타입은 `cci_bind_param()` 을 참고한다.

매개변수:

- **set** – (OUT) cci set 포인터
- **u\_type** – (IN) 엘리먼트 타입
- **size** – (IN) set 크기
- **value** – (IN) set 엘리먼트
- **indicator** – (IN) null 표시 배열(indicator array)

반환:

에러 코드

## cci\_set\_max\_row

int cci\_set\_max\_row(int req\_handle, int max)

`cci_execute()` 에 의해 수행되는 **SELECT** 문의 결과에 대해 최대 레코드 수를 지정한다. `max` 값이 0일 경우 지정하지 않은 것과 동일하다.

매개변수:

- **req\_handle** - (IN) 요청 핸들
- **max** - (IN) 최대 행

반환:

에러 코드

```
req = cci_prepare(...);
cci_set_max_row(req, 1);
cci_execute(...);
```

## cci\_set\_query\_timeout

int cci\_set\_query\_timeout(int req\_handle, int milli\_sec)

질의 수행의 타임아웃 시간을 설정한다.

매개변수:

- **req\_handle** - (IN) 요청 핸들
- **milli\_sec** - 타임아웃(timeout) 시간, 단위는 msec.

반환:

성공 : 요청 핸들 ID, 실패 : 에러 코드

- **CCI\_ER\_REQ\_HANDLE**

**cci\_set\_query\_timeout** 으로 설정된 타임아웃 시간은 `cci_prepare()`, `cci_execute()`, `cci_execute_array()`, `cci_execute_batch()` 함수들에 영향을 미친다. 각 함수에서 타임아웃이 발생했을 때 `cci_connect_with_url()` 연결 URL에 설정한 **disconnect\_on\_query\_timeout** 의 값이 yes이면 **CCI\_ER\_QUERY\_TIMEOUT** 에러를 반환한다.

위 함수들은 `cci_connect_with_url()` 함수의 인자인 연결 URL에 **login\_timeout** 이 설정되어 있는 경우에도 **CCI\_ER\_LOGIN\_TIMEOUT** 에러를 반환할 수 있는데, 이는 응용 클라이언트와 브로커 응용 서버(CAS) 간 재연결 과정에서 로그인 타임아웃이 발생한 경우이다.

응용 클라이언트와 CAS 간 재연결 과정은 CAS가 재시작하거나 재스케줄되는 경우에 발생한다. 재스케줄이란 CAS가 트랜잭션 단위로 응용 클라이언트를 선택하여 연결을 시작하고 종료

하는 과정을 의미하는데, 브로커 파라미터인 **KEEP\_CONNECTION** 이 AUTO이면 상황에 따라 발생한다. 보다 자세한 사항은 **브로커별 파라미터**의 **KEEP\_CONNECTION** 설명을 참고한다.

## cci\_set\_size

int cci\_set\_size(T\_CCI\_SET set)

**T\_CCI\_SET** 타입 값에 대한 엘리먼트 개수를 가져온다.

매개변수:

- **set** – (IN) cci set 포인터

반환:

크기

## PHP 드라이버

CUBRID PHP 드라이버는 PHP로 작성한 응용 프로그램에서 CUBRID 데이터베이스를 사용할 수 있는 API를 제공한다. CUBRID PHP 드라이버가 제공하는 모든 함수는 앞에 **cubrid\_** 가 붙는다 (예: cubrid\_connect(), cubrid\_connect\_with\_url()).

공식 CUBRID PHP 드라이버는 PECL 패키지로 제공한다. PECL은 PHP 확장 저장소로, PHP 확장 개발 및 다운로드를 위한 편의 기능을 제공한다. PECL에 대한 자세한 내용은 <https://pecl.php.net/> 을 참고한다.

CUBRID PHP 드라이버는 CCI API를 기반으로 작성되었으므로, CCI API 및 CCI에 적용되는 **CCI\_DEFAULT\_AUTOCOMMIT** 과 같은 설정 파라미터에 영향을 받는다.

## PHP 설치 및 설정

가장 쉽고 빠르게 응용 프로그램을 시스템에 설치하려면 Ubuntu에 CUBRID와 Apache, PHP를 설치한다.

### Linux

#### 기본 환경

- 운영체제: Linux: 32 비트 또는 64비트
- 웹 서버: Apache

- PHP: 5.2 또는 5.3( <https://www.php.net/downloads.php> )
- PHP: 5.x, 또는 7.x (<https://www.php.net/downloads.php>)
- 편의상 최신 PHP 5.6.x, 7.1.x 및 7.4.x를 참조 바랍니다.

## PECL을 이용한 설치

**PECL** 이 설치되어 있다면, **PECL** 이 소스코드 다운로드 및 컴파일을 수행하므로 다음과 같이 간단하게 CUBRID PHP 드라이버를 설치할 수 있다.

1. 다음과 같은 명령어를 입력하여 CUBRID PHP 드라이버 최신 버전을 설치한다.

```
sudo pecl install cubrid
```

하위 버전의 드라이버가 필요하면 다음과 같이 설치할 버전을 지정할 수 있다.

```
sudo pecl install cubrid-10.1.0.0002
```

설치가 진행되는 중에 **CUBRID base install dir autodetect :** 라는 프롬프트가 표시된다. 설치를 원활하게 진행하기 위해서 CUBRID를 설치한 디렉터리의 전체 경로를 입력한다. 예를 들어 CUBRID가 **/home/cubridtest/CUBRID** 디렉터리에 설치되었다면, **/home/cubridtest/CUBRID** 를 입력한다.

2. 설정 파일을 수정한다.

CentOS 6.0 이상 버전이나 Fedora 15 이상 버전을 사용한다면 **cubrid.ini** 파일을 생성하고 내용에 **extension=cubrid.so** 를 입력하여 **/etc/php.d** 디렉터리에 저장한다.

다른 운영체제를 사용한다면 **php.ini** 파일 끝에 다음 두 줄의 내용을 추가한다. **php.ini** 파일의 기본 위치는 **/etc/php5/apache2** 또는 **/etc** 이다.

```
[CUBRID]
extension=cubrid.so
```

3. 변경된 내용을 반영하려면 웹 서버를 재시작한다.

## apt-get을 이용하여 Ubuntu에 설치

1. PHP가 설치되어 있지 않다면, 다음 명령어로 PHP를 설치한다.

```
sudo apt-get install php5

or for PHP version 7
sudo add-apt-repository ppa:ondrej/php
```

```
sudo apt-get update
sudo apt-get install php7.1
```

2. **apt-get** 를 이용하여 CUBRID PHP 드라이버를 설치하려면, Ubuntu가 패키지 다운로드 위치를 알고 인덱스를 업데이트하도록 CUBRID 저장소를 추가해야 한다.

```
sudo add-apt-repository ppa:cubrid/cubrid
sudo apt-get update
```

3. 다음과 같이 드라이버를 설치한다.

```
sudo apt-get install php5-cubrid
```

최신 버전보다 하위 버전을 설치하려면 다음과 같이 버전을 명시한다.

```
sudo apt-get install php5-cubrid-8.3.1
```

위 명령어는 **cubrid.so** 드라이버를 **/usr/local/lib/php/\*** 디렉터리에 복사하고 다음과 같은 설정을 **/etc/php.ini** 파일에 추가한다.

```
[PHP_CUBRID]
extension=cubrid.so
```

4. PHP가 모듈을 읽어들이도록 Apache 서버를 재시작한다.

```
service apache2 restart
```

## Windows

### 기본 환경

- CUBRID: 9.3.x 이상
- 운영체제: Windows 32 비트 또는 64비트
- 웹 서버: Apache 또는 IIS
- PHP: 5.6.x, 7.1.x 또는 7.4.x(<https://windows.php.net/download/>)
- PHP 7.1.x 또는 7.4.x 의 경우 32bit 또는 64bit용 Microsoft Visual C ++ 2015 재배포 가능 패키지를 설치해야 한다.

### CUBRID PHP API Installer를 사용한 설치

CUBRID PHP API Installer는 자동으로 CUBRID와 PHP의 버전을 인식하여 해당 버전에 맞는 드라이버를 설치하는 Windows 설치 관리자이다. 드라이버를 기본 PHP 확장 디렉터리(**C:\Program Files\PHP\ext**)에 복사하고 **php.ini** 파일을 수정한다. 여기에서는 CUBRID PHP API Installer를 이용하여 Windows에 CUBRID PHP 확장을 설치하는 방법을 설명한다.

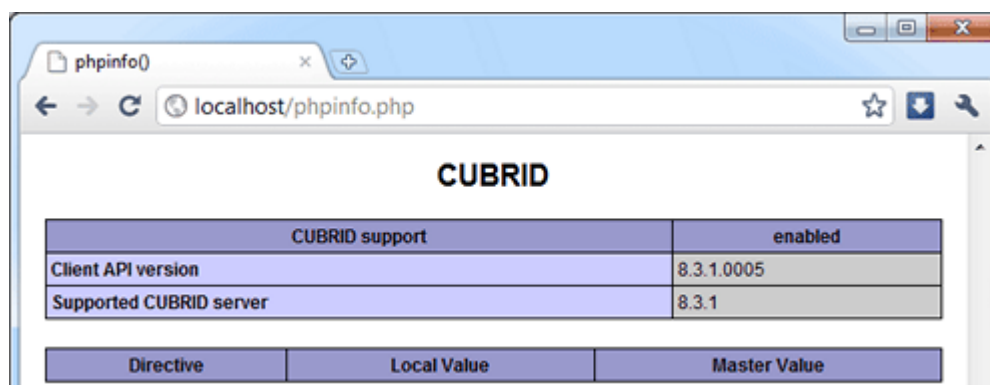
CUBRID PHP 드라이버를 제거하려면 CUBRID PHP API Installer를 다시 실행하여 프로그램 제거를 선택한다. 이 방법으로 CUBRID PHP 드라이버를 제거하면 설치할 때 발생한 모든 변경 사항이 복구된다.

CUBRID PHP 드라이버를 설치하기 전에 PHP와 CUBRID의 경로가 시스템 변수의 **Path**에 추가되어 있어야 한다.

1. 다음 주소에서 CUBRID PHP API Installer를 다운로드한다. 아래 주소에서는 모든 CUBRID 버전에 대한 CUBRID PHP 드라이버를 제공한다.

<https://www.cubrid.org/downloads#php>

2. CUBRID PHP API Installer를 실행하고 [다음]을 클릭하여 설치를 진행한다.
3. BSD 라이선스 조항에 동의하고 [다음]을 클릭한다.
4. CUBRID PHP API Installer를 설치할 경로를 지정하고 [다음]을 클릭한다. PHP를 설치한 경로가 아니라 예를 들면 **C:\Program Files\CUBRID PHP API**와 같은 새로운 경로를 입력해야 한다.
5. Windows [시작] 메뉴의 폴더 이름을 지정하고 [설치]를 클릭한다. 설치에 실패하면 아래의 **환경 변수 설정**을 참고한다.
6. 설치를 마치면 [마침]을 클릭한다.
7. 변경 내용을 반영하기 위해서 웹 서버를 재시작한다. 제대로 설치되었는지 확인하려면 `phpinfo()`를 실행한다.



## 시스템 환경 변수 설정

설치 중에 오류가 발생하면 시스템 환경 변수가 제대로 설정되었는지 확인해야 한다. CUBRID를 설치하면 자동으로 설치 경로가 시스템 환경 변수 **Path**에 추가된다. 시스템 환경 변수가 제대로 설치되

었는지 확인하려면, Windows의 [시작] > [모든 프로그램] > [보조프로그램] > [명령 프롬프트]를 실행하고 다음 작업을 수행한다.

1. 다음 명령을 입력한다.

```
php --version
```

시스템 환경 변수가 제대로 설정되었다면 아래와 같이 PHP 버전을 확인할 수 있다.

```
PHP 5.6.30 (cli) (built: Jun 13 2017 16:16:30)
또는 7.1.x
PHP 7.1.7 (cli) (built: Aug 3 2017 10:59:35) (NTS)

C:\Users\Administrator>php --version
PHP 5.6.30 (cli) (built: Jan 18 2017 19:47:28)
```

2. 다음 명령을 입력한다.

```
cubrid --version
```

시스템 환경 변수가 제대로 설정되었다면 아래와 같이 CUBRID 버전을 확인할 수 있다.

```
C:\Users\Administrator>cubrid --version
cubrid.exe (CUBRID utilities)
CUBRID 9.3 (9.3.8.0003) (64bit release build for Windows_NT) (Apr
11 2017 11:54:08)
```

위와 같은 결과가 출력되지 않는다면 PHP와 CUBRID가 설치되지 않았을 가능성이 높으므로 PHP와 CUBRID를 다시 설치한다. 만약 다시 설치해도 시스템 환경 변수가 제대로 설정되지 않는다면, 다음과 같이 수동으로 시스템 환경 변수를 설정한다.

1. [내 컴퓨터]를 마우스 오른쪽 버튼으로 클릭하여 [속성]을 선택하면 [시스템 속성] 대화 상자가 나타난다.
2. [고급] 탭을 선택하고 [환경 변수]를 클릭한다.
3. [시스템 변수]에서 **Path** 를 선택하고 [편집]을 클릭한다.
4. 변수 값에 CUBRID와 PHP의 설치 경로를 추가한다. 각 경로는 세미콜론(;)으로 구분한다. 만약 PHP를 **C:\Program Files\PHP** 디렉터리에 설치하고 CUBRID를 **C:\CUBRID\bin** 디렉터리에 설치했다면, 변수 값의 끝에 **C:\CUBRID\bin;C:\Program Files\PHP** 를 덧붙인다.
5. [확인]을 클릭한다.
6. 앞에서 설명한 방법으로 시스템 환경 변수가 제대로 설정되었는지 확인한다.



## 빌드된 드라이버 다운로드 및 설치

운영체제와 PHP 버전에 맞는 Windows용 CUBRID PHP/PDO 드라이버를 <https://www.cubrid.org/downloads#php> 에서 다운로드한다.

PHP 드라이버를 다운로드하면 **php\_cubrid.dll** 파일을 볼 수 있으며, PDO 드라이버를 다운로드하면 **php\_pdo\_cubrid.dll** 파일을 볼 수 있다. 드라이버를 설치하는 방법은 다음과 같다.

1. 드라이버 파일을 기본 PHP 확장 디렉터리( **C:\Program Files\PHP\ext** )에 복사한다.
2. 시스템 환경 변수를 설정한다. 시스템 환경 변수 **PHPRC** 의 값으로 **C:\Program Files\PHP** 가 설정되고, **Path** 에 **%PHPRC%** 와 **%PHPRC%\ext** 가 추가되었는지 확인한다.
3. **php.ini** ( **C:\Program Files\PHP\php.ini** ) 파일을 열어 끝에 다음 두 줄을 추가한다.

```
[PHP_CUBRID]
extension=php_cubrid.dll
```

PDO 드라이버의 경우에는 다음 내용을 추가한다.

```
[PHP_PDO_CUBRID]
extension = php_pdo_cubrid.dll
```

4. 웹 서버를 재시작한다.

## PHP 드라이버 빌드

### Linux

여기에서는 Linux에서 CUBRID PHP 드라이버를 빌드하는 방법을 설명한다.

### 환경 설정

- CUBRID: CUBRID를 설치한다. 시스템에 환경 변수 **%CUBRID%** 가 정의되어 있는지 확인한다.
- PHP 5.6.x , 7.1.x 또는 7.4.x 소스코드: PHP 5.3 소스코드를 다음 주소에서 다운로드한다. <https://www.php.net/downloads.php>
- Apache 2: PHP 테스트에 Apache 2를 사용할 수 있다.
- CUBRID PHP 드라이버 소스코드: <https://www.cubrid.org/downloads#php> 에서 CUBRID 버전에 맞는 CUBRID PHP 드라이버의 소스코드를 다운로드한다.

- Linux 또는 Mac에서는 CCI 드라이버 빌드를 하려면 GNU Developer Toolset 8 또는 그 이상이 필요하다.

## CUBRID PHP 드라이브 빌드

1. PHP 소스코드를 압축 해제하여 해당 디렉터리로 이동한다.

```
$> tar zxvf php-<version>.tar.gz (or tar jxvf php-<version>.tar.bz2)
$> cd php-<version>/ext
```

2. phpize를 실행한다. phpize에 대한 내용은 [참고 사항](#) 을 참고한다.

```
cubrid-php> /usr/bin/phpize
```

3. 프로젝트를 설정한다. 설정을 실행하기 전에 먼저 **./configure -h** 를 실행하여 설정 옵션을 확인하는 것을 권장한다. 설정 방법은 다음과 같다(Apache 2가 **/usr/local** 에 설치되어 있다고 가정한다).

```
cubrid-php>./configure --with-cubrid --with-php-config=/usr/local/
bin/php-config
```

- --with-cubrid=shared: CUBRID 지원을 포함한다.
- --with-php-config=PATH: 절대 경로를 포함한 php-config의 파일 이름을 입력한다.

4. 프로젝트를 빌드한다. 프로젝트가 성공적으로 빌드되면 **/modules** 디렉터리에 **cubrid.so** 파일이 생성된다.

5. **cubrid.so** 파일을 **/usr/local/php/lib/php/extensions** 디렉터리에 복사한다.

```
cubrid-php> mkdir /usr/local/php/lib/php/extensions
cubrid-php> cp modules/cubrid.so /usr/local/php/lib/php/extensions
```

6. **php.ini** 파일에 **extension\_dir** 변수에 PHP 확장의 경로를 입력하고 **extension** 변수에 CUBRID PHP 드라이버 파일 이름을 입력한다.

```
extension_dir = "/usr/local/php/lib/php/extension/no-debug-zts-xxx"
extension = cubrid.so
```

## CUBRID PHP 드라이버 설치 확인

1. 다음과 같은 내용의 **test.php** 파일을 생성한다.

```
<?php phpinfo(); ?>
```

2. 웹 브라우저로 `http://localhost/test.php` 에 접속하여 다음 내용이 보이는지 확인한다. 다음 내용이 보이면 설치가 완료된 것이다.

| CUBRID  | Value       |
|---------|-------------|
| Version | 10.1.0.XXXX |

## 참고 사항

phpize는 PHP 확장의 컴파일을 준비하는 셸 스크립트로, 일반적으로 PHP를 설치할 때 자동으로 설치된다. 만약 phpize가 설치되어 있지 않으면 다음과 같은 방법으로 설치할 수 있다.

1. PHP 소스코드를 다운로드한다. PHP 확장을 사용할 버전과 일치하는 버전을 다운로드해야 한다. 다운로드한 PHP 소스코드를 압축 해제하고 소스코드의 최상위 디렉터리로 이동한다.

```
$> tar zxvf php-<version>.tar.gz (or tar jxvf php-<version>.tar.bz2)
$> cd php-<version>
```

2. 프로젝트를 설정하고, 빌드한 후 설치한다. **--prefix** 옵션으로 PHP를 설치할 디렉터리를 지정할 수 있다.

```
php-root> ./configure --prefix=prefix_dir; make; make install
```

3. phpize는 **prefix\_dir/bin** 디렉터리에 위치한다.

## Windows

여기에서는 Windows에서 CUBRID PHP 드라이버를 빌드하는 방법을 설명한다. 어떤 버전을 선택해야 할지 알 수 없는 경우 다음 내용을 참고한다.

Apache.org에서 Apache 빌드시 PHP를 모듈로 사용하는 경우(권장되지 않음) Visual Studio 6 컴파일러로 컴파일 된 VC6 PHP버전을 사용해야 한다. Apache.org 바이너리와 함께 VC11+ 버전의 PHP를 사용하면 안된다.

Apache에서는 PHP의 Thread Safe(TS) 버전을 사용해야 한다.

- PHP 버전 5.5.x 이상을 사용하는 경우 VC11 버전을 사용해야 한다. (Visual Studio 2012)
- PHP 버전 7.1.x 이상을 사용하는 경우 VC14 버전을 사용해야 한다. (Visual Studio 2015)

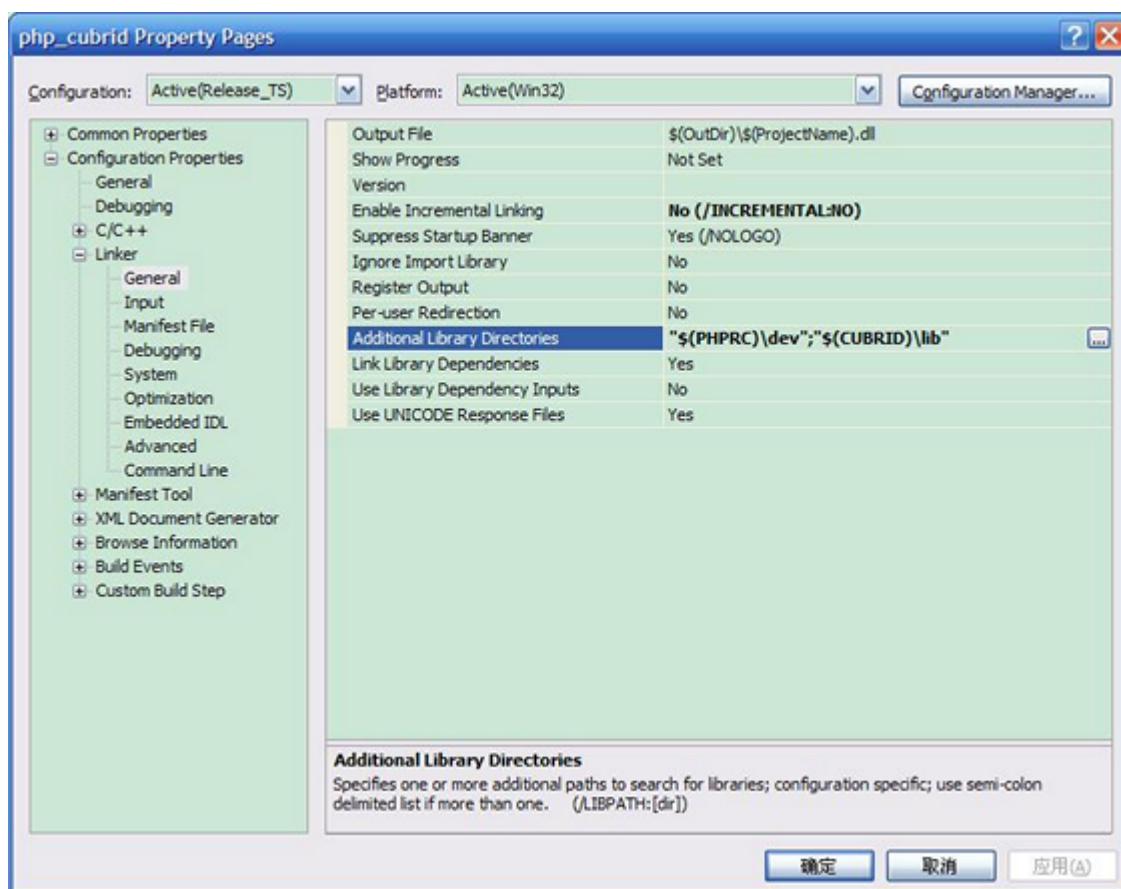
VC11 및 VC14 버전은 각각 Visual Studio 2012 및 2015 컴파일러로 컴파일된다. VC11 또는 VC14 버전은 성능과 안정성이 더욱 향상되었다.

PHP의 최신 버전은 VC11, VC14 (각각 Visual Studio 2012 또는 2015 컴파일러)로 빌드되며 성능 및 안정성이 향상되었다. \* VC11 에서 빌드를 하기 위해서는 Visual C ++ Redistributable for Visual Studio 2012 x86 또는 x64가 설치되어 있어야 한다. \* VC14 에서 빌드를 하기 위해서는 Visual C ++ Redistributable for Visual Studio 2015 x86 또는 x64가 설치되어 있어야 한다.

## VC11를 이용하여 PHP 5.6.x CUBRID PHP 드라이버 빌드

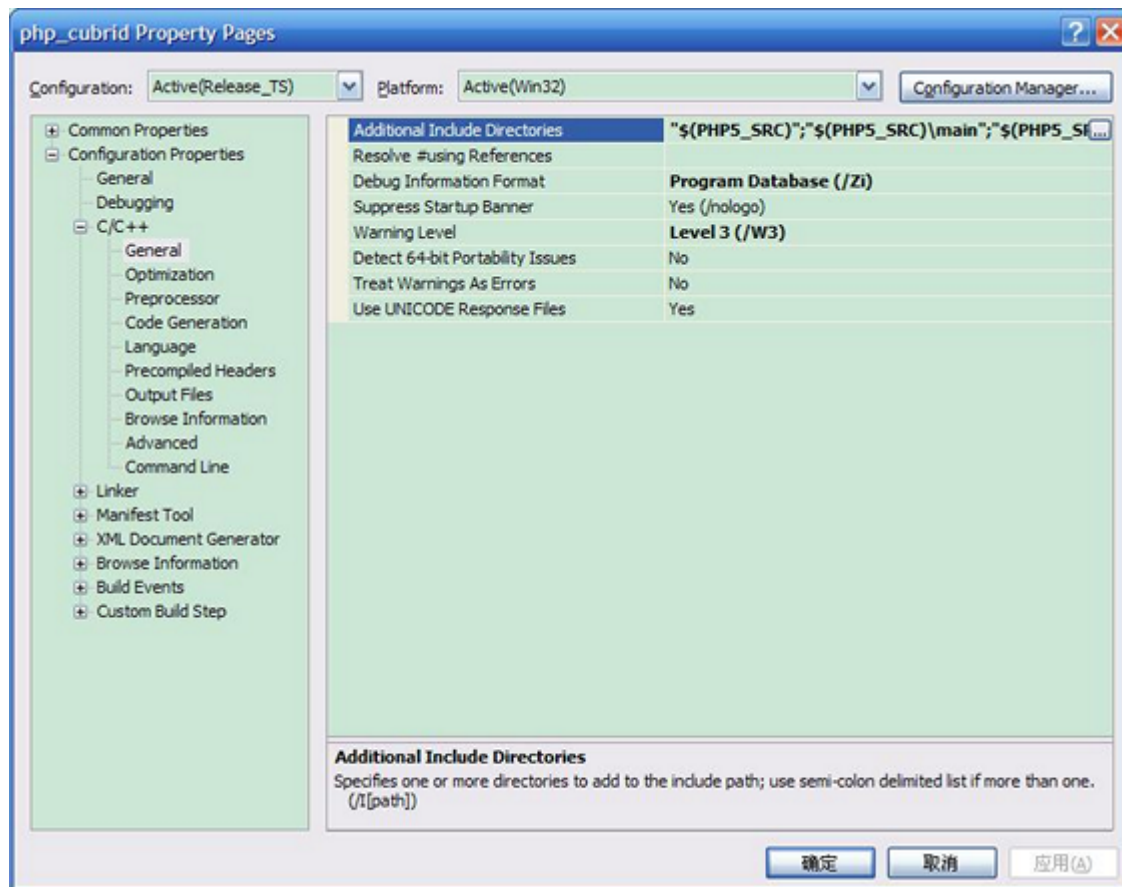
### 환경 설정

- CUBRID: CUBRID를 설치한다. 시스템에 환경 변수 **%CUBRID%** 가 정의되어 있는지 확인한다.
- Visual Studio 2012: makefile을 잘 다룰 수 있는 사용자라면, Visual Studio 대신에 무료인 Visual C++ Express Edition이나 Windows SDK 에 포함된 VC++ 11 컴파일러를 사용할 수 있다. Windows에서 CUBRID PHP VC11 드라이버를 사용하려면 Visual C++ Redistributable Package가 설치되어 있어야 한다.
- PHP 5.6.x 바이너리: VC11 x86 Non Thread Safe 또는 VC11 x86 Thread Safe를 사용할 수 있다. 시스템 환경 변수 **%PHPRC%** 가 제대로 정의되어 있어야 한다. VC11 프로젝트 속성에서 [Linker] > [General]을 선택하면 [Additional Library Directories]에서 **\$(PHPRC)** 가 사용되는 것을 볼 수 있다.



- PHP 5.6.x 소스코드: 바이너리 버전에 맞는 소스코드를 다운로드해야 한다. PHP 5.6.x 소스코드를 다운로드한 후 압축 해제하고, 시스템 환경 변수 **%PHP5\_SRC%** 를 추가하여 PHP 5.6.x 소스코드의

경로를 값으로 설정한다. VC11 프로젝트 속성에서 [C/C++] > [General]을 선택하면 [Additional Library Directories]에서 **\$(PHP5\_SRC)**가 사용되는 것을 볼 수 있다.



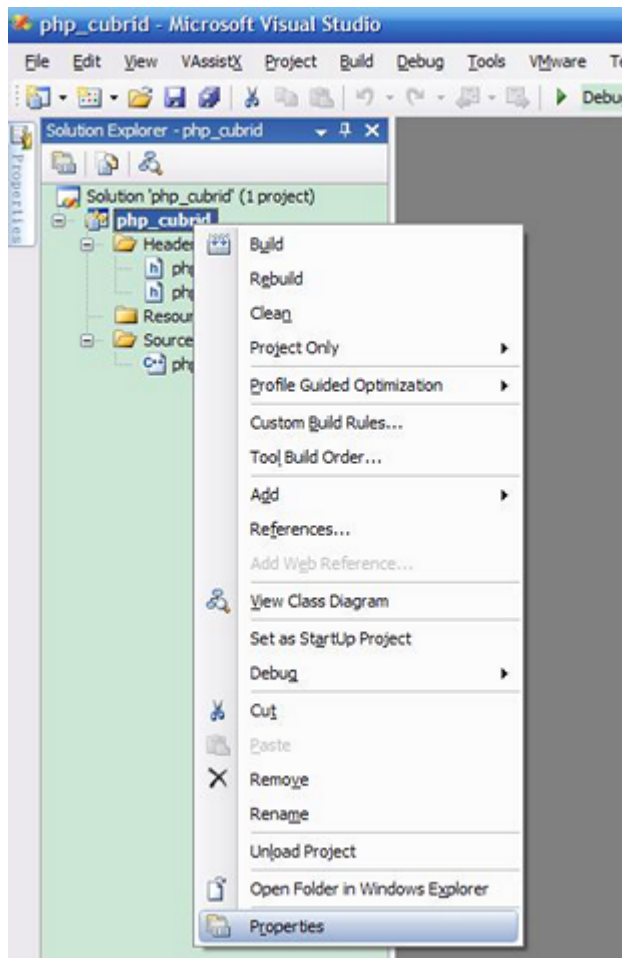
- CUBRID PHP 드라이버 소스코드: <https://www.cubrid.org/downloads#php> 에서 CUBRID 버전에 맞는 CUBRID PHP 드라이버의 소스코드를 다운로드한다.

## 참고

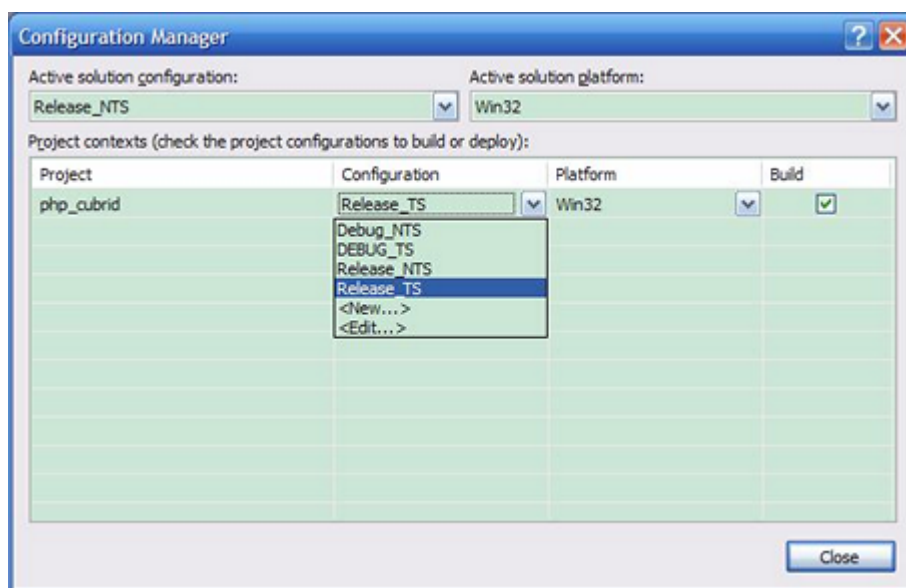
PHP 5.6.x을 소스코드에서 빌드할 필요는 없지만 PHP 5.6.x 프로젝트를 설정해야 한다. PHP 5.6.x 프로젝트를 설정하지 않으면 VC11에서 config.w32.h 헤더 파일을 찾을 수 없다는 메시지가 출력된다. 설정 방법은 다음 주소를 참고한다. <https://wiki.php.net/internals/windows/stepbystepbuild>

## CUBRID PHP 드라이버 빌드

1. 다운로드한 CUBRID PHP 드라이버 소스코드의 **\win** 디렉터리에 있는 **php\_cubrid.vcproj** 파일을 열고, 왼쪽의 [Solution Explorer] 창에서 **php\_cubrid**를 마우스 오른쪽 버튼으로 클릭하여 [Properties]를 선택한다.



2. [Property Page] 대화 상자에서 [Configuration Manager]을 클릭한다. [Project context]의 [Configuration]에서 네 가지 설정(Release\_TS, Release\_NTS, Debug\_TS and Debug\_NTS) 중 원하는 값을 선택하고 [닫기]를 클릭한다.



3. 설정을 마친 후에는 [OK]를 클릭한 후, <F7> 키를 눌러 컴파일한다.

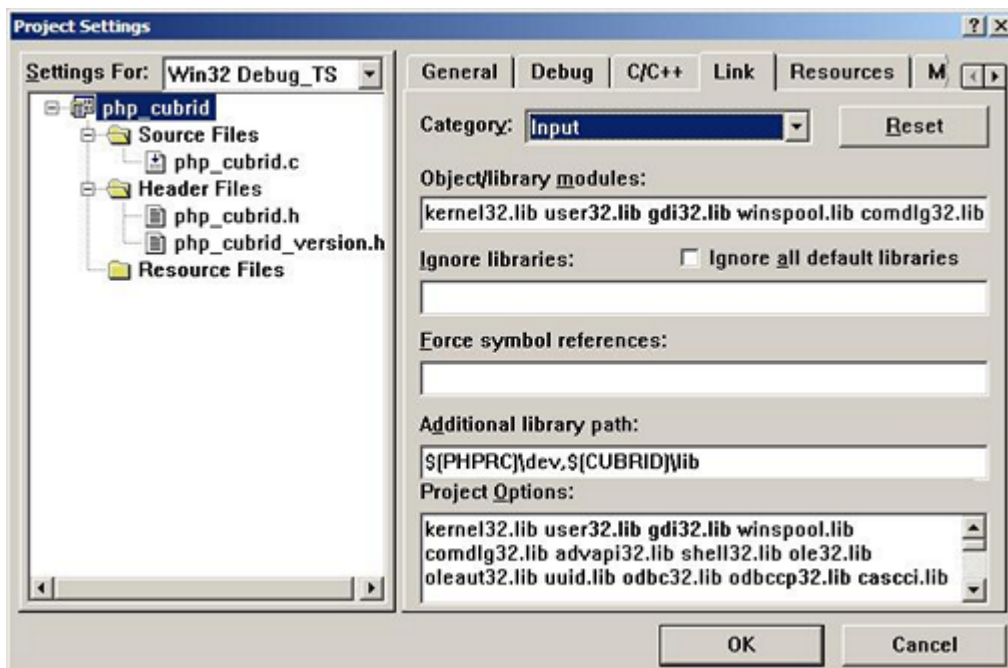
4. **php\_cubrid.dll** 파일을 빌드한 후에는 PHP가 **php\_cubrid.dll** 파일을 PHP 확장으로 인식하도록 다음 작업을 수행한다.

- PHP를 설치한 폴더에 **cubrid** 폴더를 생성하고 해당 폴더에 **php\_cubrid.dll** 파일을 복사한다. **%PHPRC%\ext** 디렉터리가 있다면 이 디렉터리에 **php\_cubrid.dll** 파일을 복사해도 된다.
- In **php.ini** 파일의 **extension\_dir** 변수의 값으로 **php\_cubrid.dll** 파일의 경로를 입력하고, **extension** 변수의 값으로 **php\_cubrid.dll** 을 입력한다.

## VC14을 이용하여 PHP 7.1.x용 CUBRID PHP 드라이버 빌드

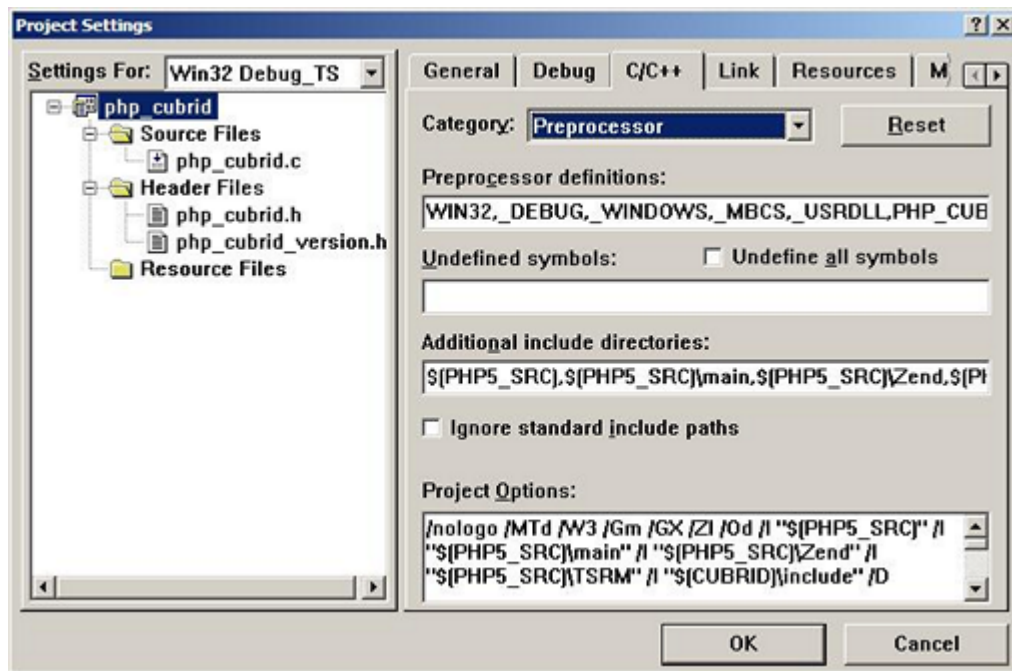
### 환경 설정

- CUBRID: CUBRID를 설치한다. 시스템에 환경 변수 **%CUBRID%** 가 정의되어 있는지 확인한다.
- Windows SDK에 포함 된 무료 Visual C++ Express Edition 또는 Visual C++ 14 컴파일러를 모두 사용할 수 있다. CUBRID PHP VC14 드라이버를 사용하려면 시스템에 Microsoft Visual C++ Redistributable Package가 설치되어 있어야 한다.
- PHP 7.1.x 바이너리: VC14 x86 Non Thread Safe 또는 VC6 x86 Thread Safe를 사용할 수 있다. 시스템 환경 변수 **%PHPRC%** 가 제대로 정의되어 있어야 한다. VC14 프로젝트의 [Project Settings]을 열면 [Link] 탭의 [Additional library path]에서 **\$(PHPRC)** 가 사용되는 것을 볼 수 있다.



- PHP 7.1.x 소스코드: 바이너리 버전에 맞는 소스코드를 다운로드해야 한다. PHP 소스코드를 다운로드한 후 압축 해제하고, 시스템 환경 변수 **%PHP7\_SRC%** 를 추가하여 PHP 소스코드의 경로를 값으로 설정한다. VC11 프로젝트의 [Project Settings]을 열면 [C/C++] 탭의 [Additional include directories]에서 **\$(PHP7\_SRC)** 가 사용되는 것을 볼 수 있다.





- CUBRID PHP 드라이버 소스코드: <https://www.cubrid.org/downloads#php> 에서 CUBRID 버전에 맞는 CUBRID PHP 드라이버의 소스코드를 다운로드한다.

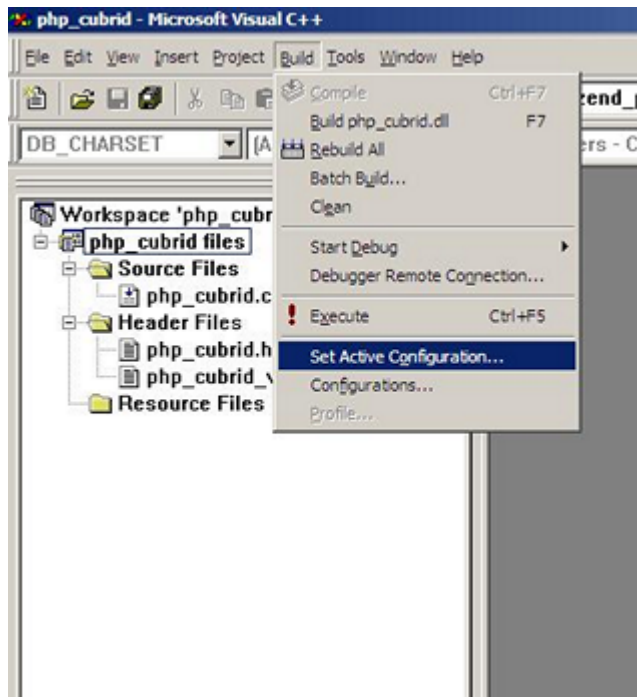
### 참고

PHP 7.1.x 소스코드로 CUBRID PHP 드라이버를 빌드한다면, Windows에서 PHP 7.1.x를 설정해야 한다. PHP 7.1.x 프로젝트를 설정하지 않으면 VC9에서 config.w32.h 헤더 파일을 찾을 수 없다는 메시지가 출력된다. 설정 방법은 다음 주소를 참고한다. <https://wiki.php.net/internals/windows/stepbystepbuild>

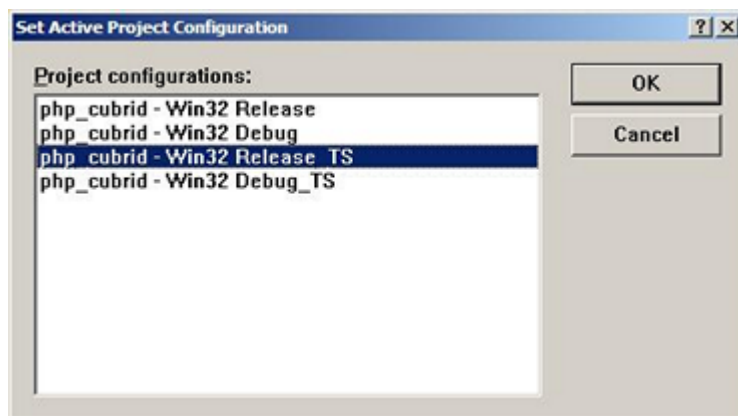
## CUBRID PHP 드라이버 빌드

1. 다운로드한 CUBRID PHP 드라이버 소스코드에서 **php\_cubrid.dsp** 파일을 열고, 메뉴에서 [Build] > [Set Active Configuration]를 선택한다. There are four configurations (Win32 Release\_TS, Win32 Release, Win32 Debug\_TS and Win32 Debug). Choose what you want, then close the [Set Active Project Configuration].





2. 네 가지 프로젝트 설정(Win32 Release\_TS, Win32 Release, Win32 Debug\_TS and Win32 Debug) 중에서 원하는 설정을 선택하고 [OK]를 클릭한다.

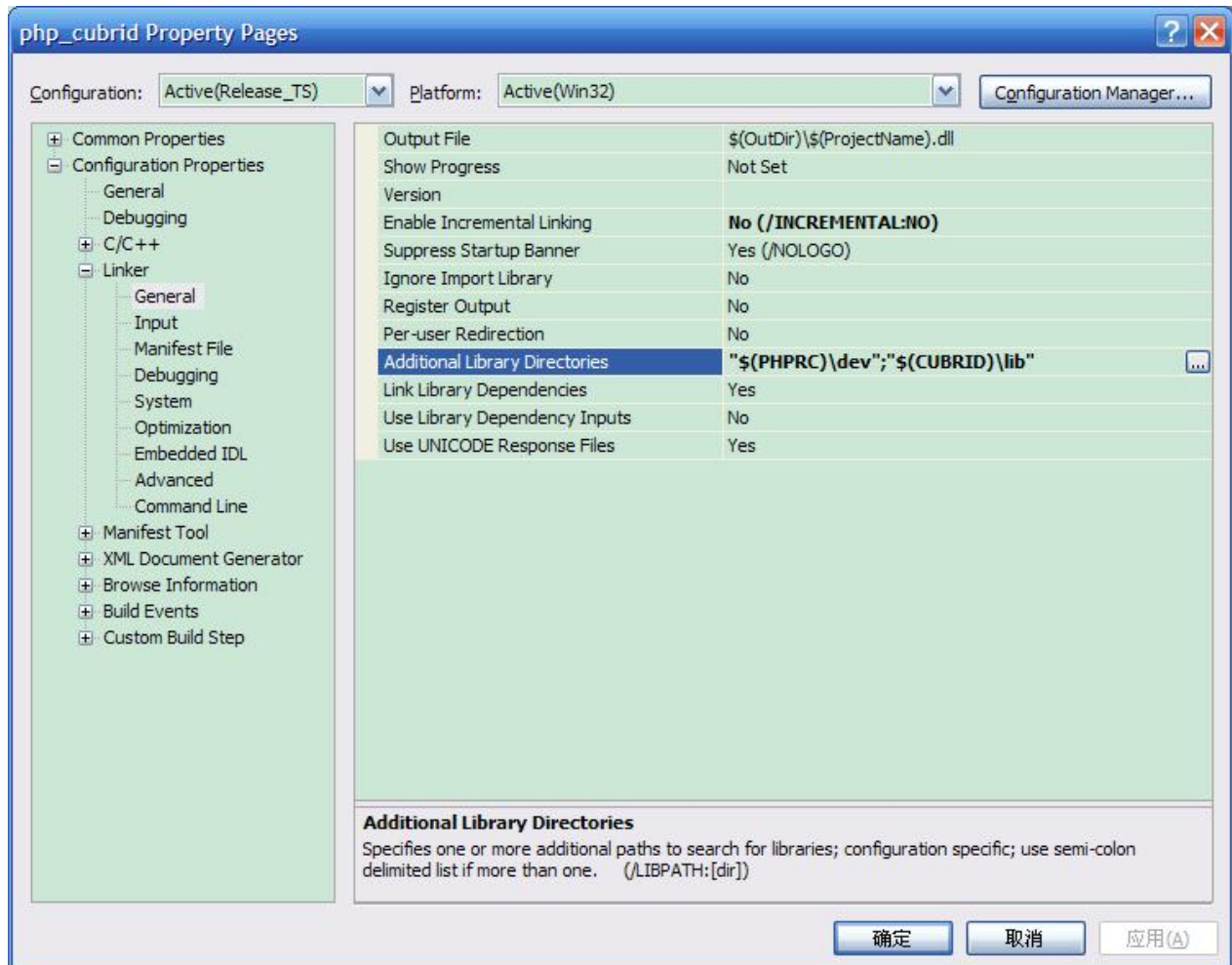


3. <F7> 키를 눌러 소스코드를 컴파일한다.
4. **php\_cubrid.dll** 파일을 빌드한 후에는 PHP가 **php\_cubrid.dll** 파일을 PHP 확장으로 인식하도록 다음 작업을 수행한다.
- PHP를 설치한 폴더에 **cubrid** 폴더를 생성하고 해당 폴더에 **php\_cubrid.dll** 파일을 복사한다. %PHPRC%\ext 디렉터리가 있다면 이 디렉터리에 **php\_cubrid.dll** 파일을 복사해도 된다.
  - In **php.ini** 파일의 **extension\_dir** 변수의 값으로 **php\_cubrid.dll** 파일의 경로를 입력하고, **extension** 변수의 값으로 **php\_cubrid.dll** 을 입력한다.

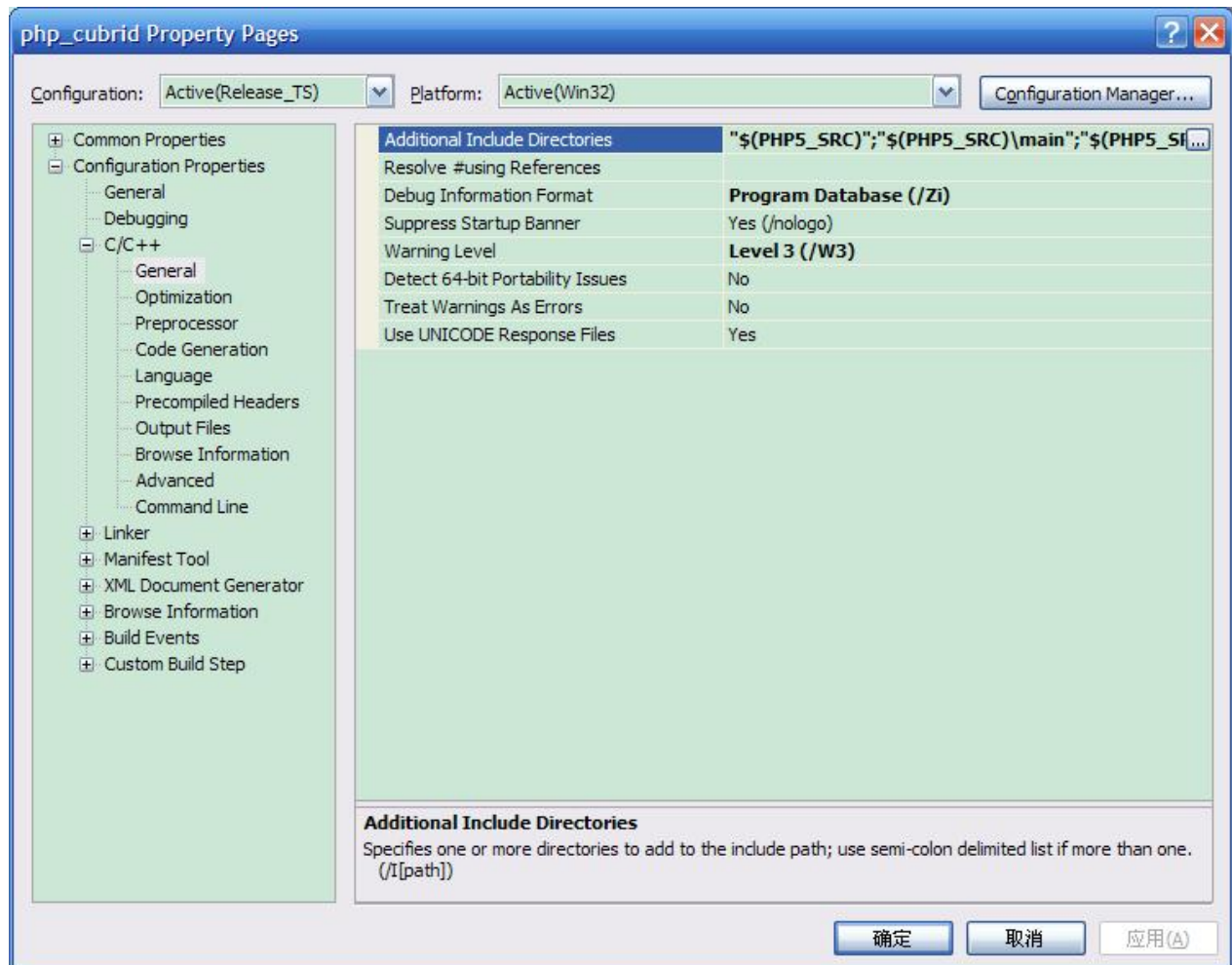
## Windows x64 CUBRID PHP 드라이버 빌드

## x64 PHP

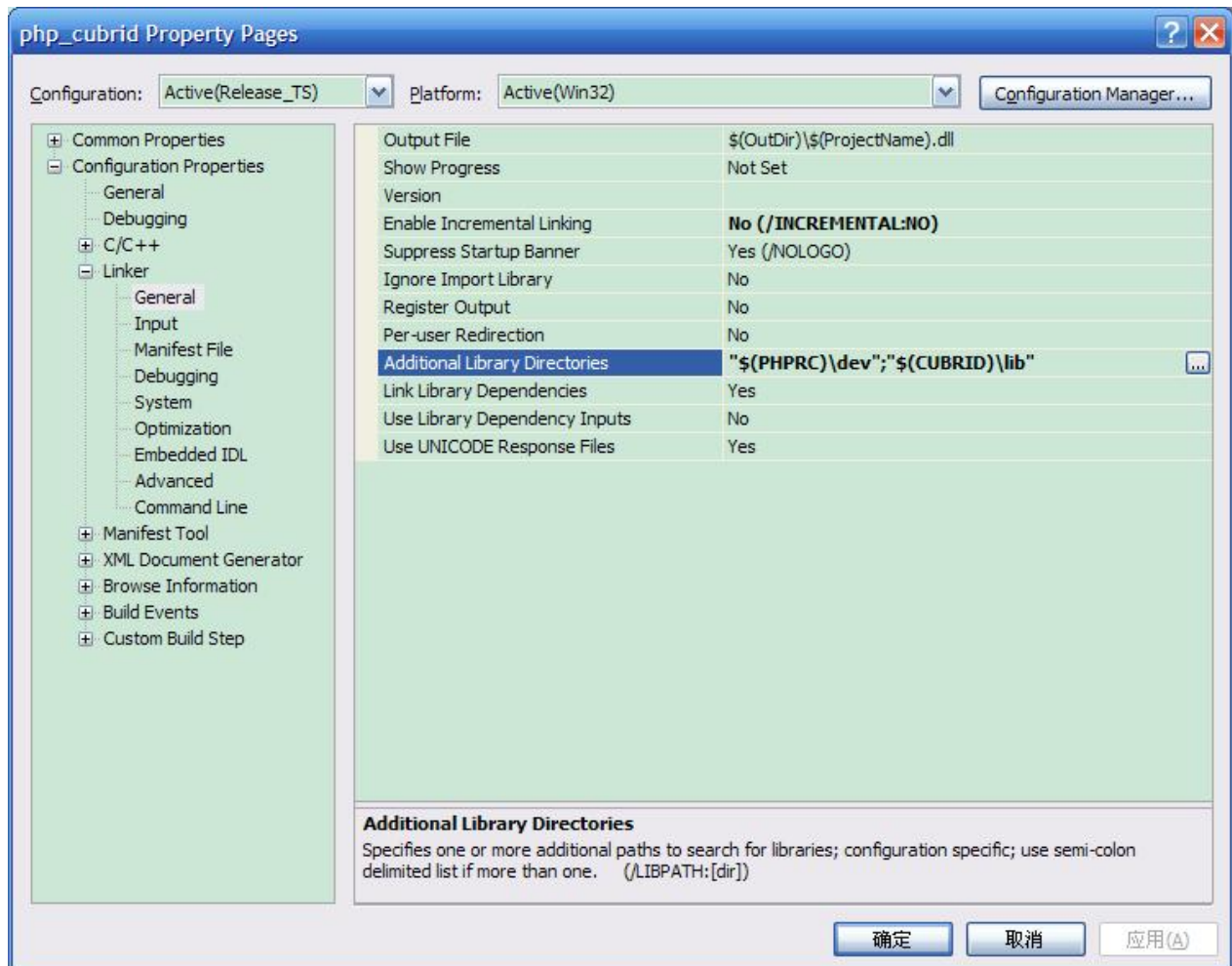
- PHP 5.6.x 바이너리 : VC11 x64 Non Thread Safe 또는 VC11 x64 Thread Safe를 설치할 수 있다. 시스템 환경 변수 **%PHPRC%** 가 제대로 정의되어 있어야 한다. [Property Pages] 대화 상자의 [Linker]에서 [General]을 선택한다. [Additional Library Directories]에서 **\$(PHPRC)**를 볼 수 있다.



- PHP 5.6.x 소스 코드 : 바이너리 버전과 일치하는 소스 코드를 가져와야 한다. PHP 5.6.x 소스 코드를 압축 해제한 후 시스템 환경 변수 **%PHPRC%**를 추가하고 해당 값에 PHP 5.6.x 소스 코드의 경로로 설정해야 한다. [Property Pages] 대화 상자의 [C/C++]에서 [General]을 선택한다. [Additional Library Directories]에서 **\$(PHP5\_SRC)**를 볼 수 있다..

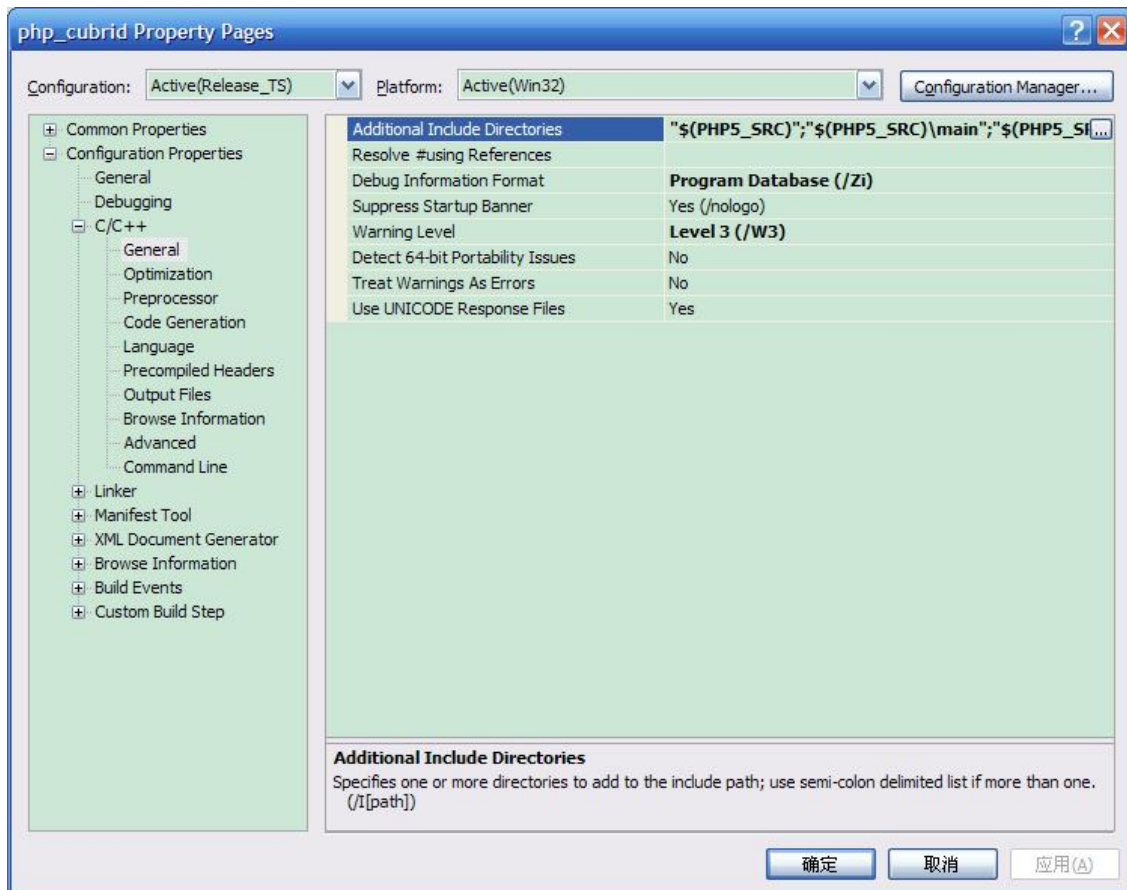


- PHP 7.1.x 바이너리 : VC14 x64 Non Thread Safe 또는 VC14 x64 Thread Safe를 설치할 수 있다. 시스템 환경 변수 **%PHPRC%** 가 올바르게 설정되어 있는지 확인해야 한다. [Property Pages] 대화 상자의 [Linker] 에서 [General]을 선택한다. [Additional Library Directories]에서 **\$(PHPRC)** 를 볼 수 있다.



- PHP 7.1.x 소스 코드 : 바이너리 버전과 일치하는 소스 코드를 가져와야 한다. PHP 7.1.x 소스 코드를 압축을 해제한 후 시스템 환경 변수에 **%PHP7\_SRC%** 를 추가하고 해당 값에 PHP 7.1.x 소스 코드의 경로로 설정해야 한다. [Property Pages] 대화 상자의 [C/C++] 에서 [General]을 선택한다. [Additional Library Directories]에서 **\$(PHP7\_SRC)** 를 볼 수 있다.

Windows에서 PHP 빌드를 지원하는 컴파일러 목록은 <https://wiki.php.net/internals/windows/compiler> 에서 제공하며, x64 PHP를 빌드할 때에는 Visual C++ 8(2005)와 Visual C++ 9(2008 SP1 only) 을 사용할 수 있다는 것을 확인할 수 있다. Visual C++ 2005 미만 버전에서 x64 PHP를 빌드하려면 Windows Server Feb. 2003 SDK를 사용해야 한다.



- <https://wiki.php.net/internals/windows/compiler>에는 Windows용 PHP 빌드를 지원하는 컴파일러를 찾을 수 있다. Visual C++ 11 (2012)과 Visual C++ 14 (2015)를 모두 사용하여 64bit PHP를 빌드할 수 있음을 알 수 있다.

## x64 Apache

- Apache Lounge는 64bit 버전을 포함한 최신 Windows 바이너리를 제공한다. 다음 링크에서 최신 Apache 2.2.34 64bit 버전을 다운로드 할 수 있다.

<https://www.apachelounge.com/download/win64/binaries/httpd-2.2.34-win64.zip>

## 환경 설정

- CUBRID x64 버전: CUBRID x64의 최신 버전을 설치한다.시스템에 환경 변수 **%CUBRID%** 가 정의되어 있는지 확인한다.
- Visual Studio 2012 or 2015: makefile을 잘 다룰 수 있는 사용자라면, Visual Studio 2008 대신에 무료인 Visual C++ Express Edition이나 Windows SDK v6.1에 포함된 VC++ 9 컴파일러를 사용할 수 있다. Windows에서 CUBRID PHP VC9 드라이버를 사용하려면 Visual C++ 2008 Redistributable Package가 설치되어 있어야 한다.
- 64-bit Windows 용 PHP 5.6.x 또는 7.1.x 바이너리 : VC11 또는 VC14 x64 이용하여 PHP를 빌드 할 수 있다. x64 Non Thread Safe와 x64 Thread Safe를 모두 사용할 수 있다. 설치 한 후 시스템 환경 변수 **%PHPRC%** 의 값이 올바르게 설정되어 있는지 확인 한다.



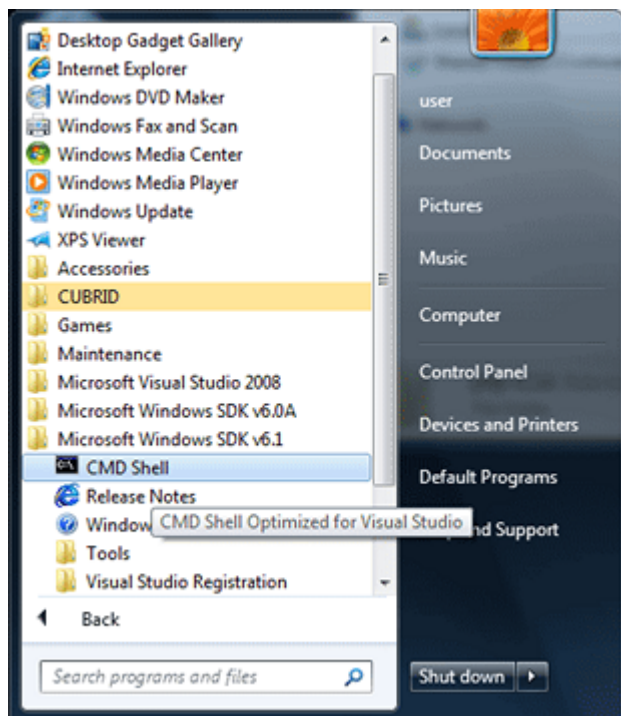
- PHP 5.6.x 소스: 바이너리 버전에 맞는 소스코드를 다운로드해야 한다. PHP 5.6.x 소스를 압축 해제한 후 시스템 환경 변수에 **%PHP5\_SRC%** 를 추가하고 해당 값에 PHP 5.6.x 소스 코드의 경로로 설정해야 한다. VC11 [Property Pages] 대화 상자의 [C/C++] 에서 [General]을 선택한다. [Additional Include Directories]에서 **\$(PHP5\_SRC)** 를 볼 수 있다.
- PHP 7.1.x 소스코드: 바이너리 버전에 맞는 소스코드를 다운로드해야 한다. PHP 7.1.x 소스코드를 다운로드한 후 압축 해제하고, 시스템 환경 변수 **%PHP7\_SRC%** 를 추가하여 PHP 7.1.x 소스코드의 경로를 값으로 설정한다. VC14 프로젝트 속성에서 [C/C++] > [General]을 선택하면 [Additional Library Directories]에서 **\$(PHP7\_SRC)** 가 사용되는 것을 볼 수 있다.
- CUBRID PHP 드라이버 소스코드: <https://www.cubrid.org/downloads#php> 에서 CUBRID 버전에 맞는 CUBRID PHP 드라이버의 소스코드를 다운로드한다.

### 참고

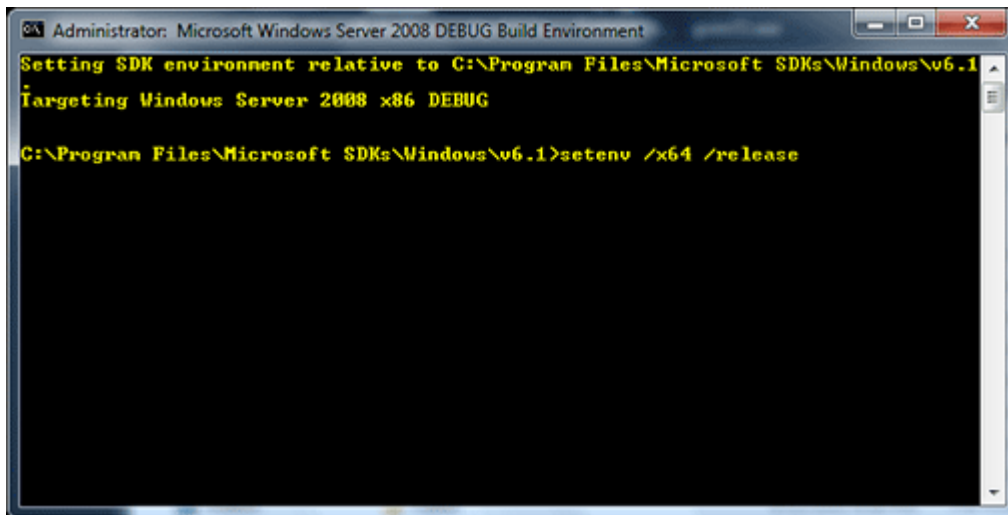
PHP 7.1.x을 소스코드에서 빌드할 필요는 없지만 PHP 7.1.x 프로젝트를 설정해야 한다. PHP 7.1.x 프로젝트를 설정하지 않으면 VC14에서 config.w32.h 헤더 파일을 찾을 수 없다는 메시지가 출력된다. 설정 방법은 다음 주소를 참고한다. <https://wiki.php.net/internals/windows/stepbystepbuild>

## PHP 5.6.x 또는 7.1.x 설정

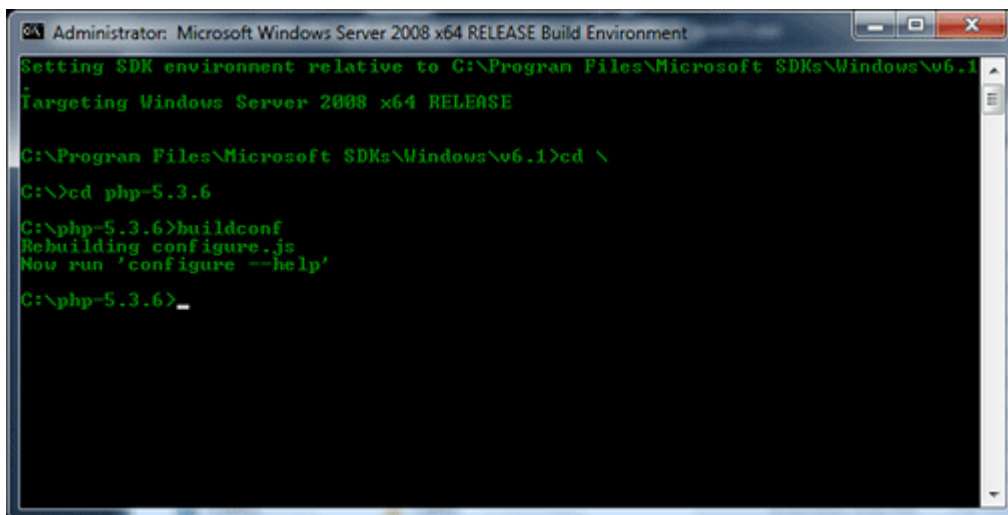
1. SDK 6.1 또는 8.1 이상을 설치한 후에는 Windows [시작] 메뉴에서 [Microsoft Windows SDK v.x] > [CMD Shell]을 선택하여 명령 셸을 시작한다.



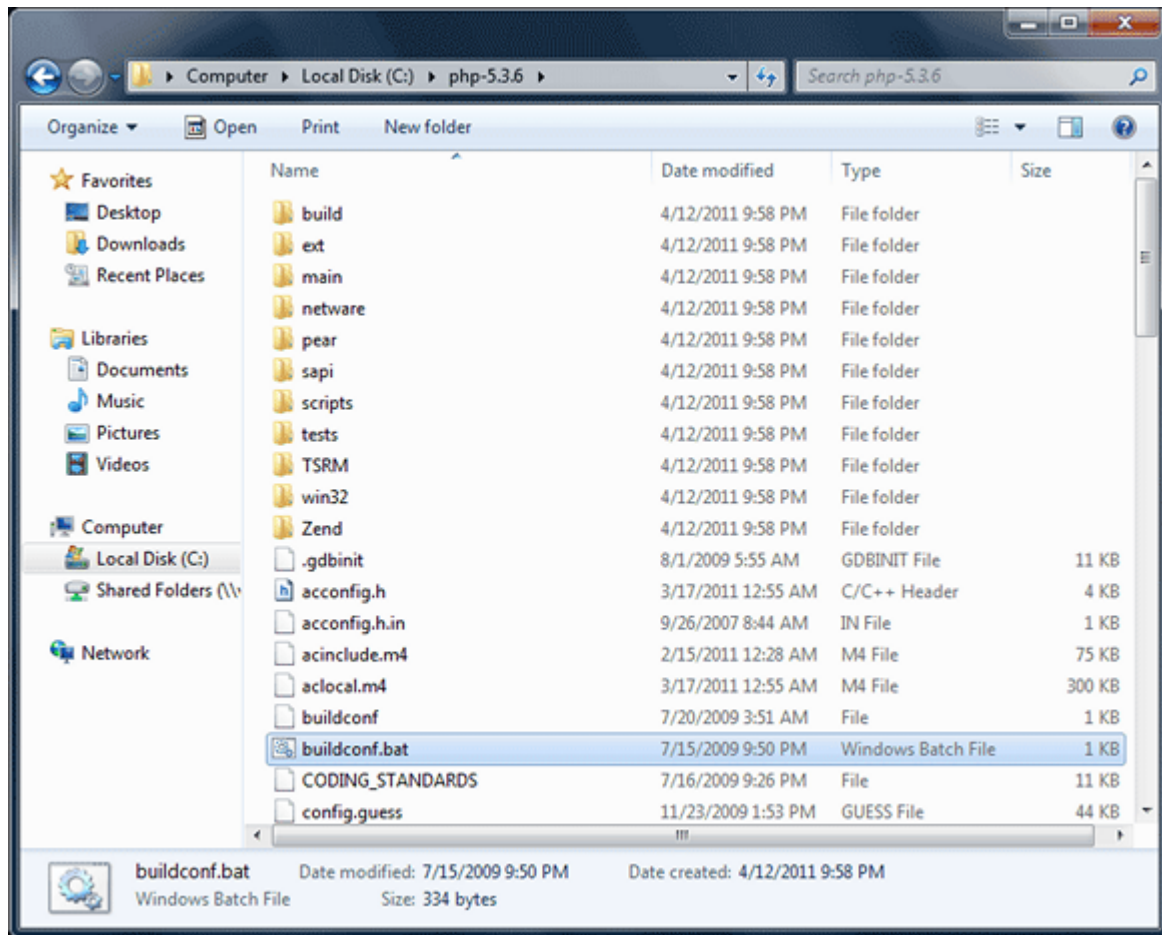
2. **setenv /x64 /release** 을 실행한다.



3. PHP 5.3 소스코드 디렉터리로 이동한 후 **buildconf** 을 실행하여 **configure.js** 파일을 생성한다.

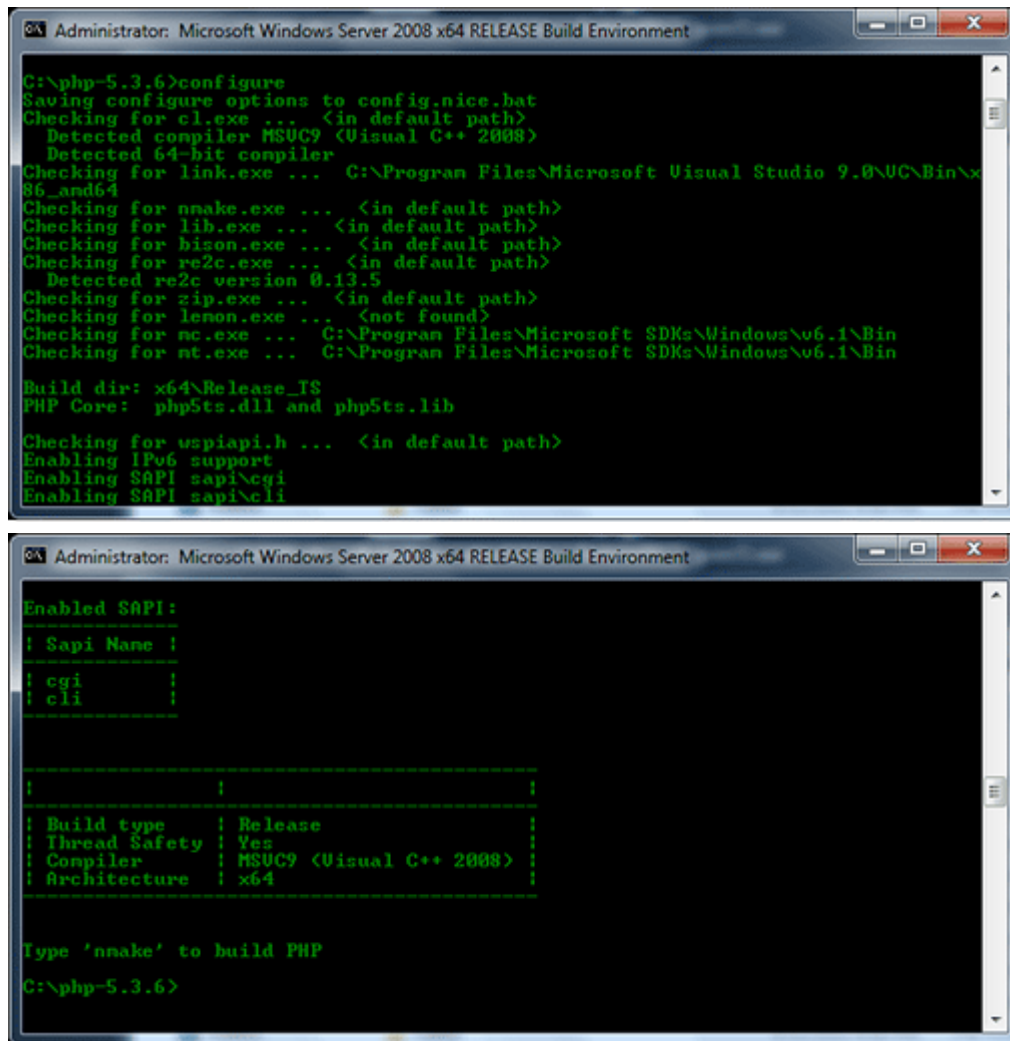


또는 PHP 5.3 소스코드에서 **buildconf.bat** 파일을 실행해도 같은 동작을 수행한다.



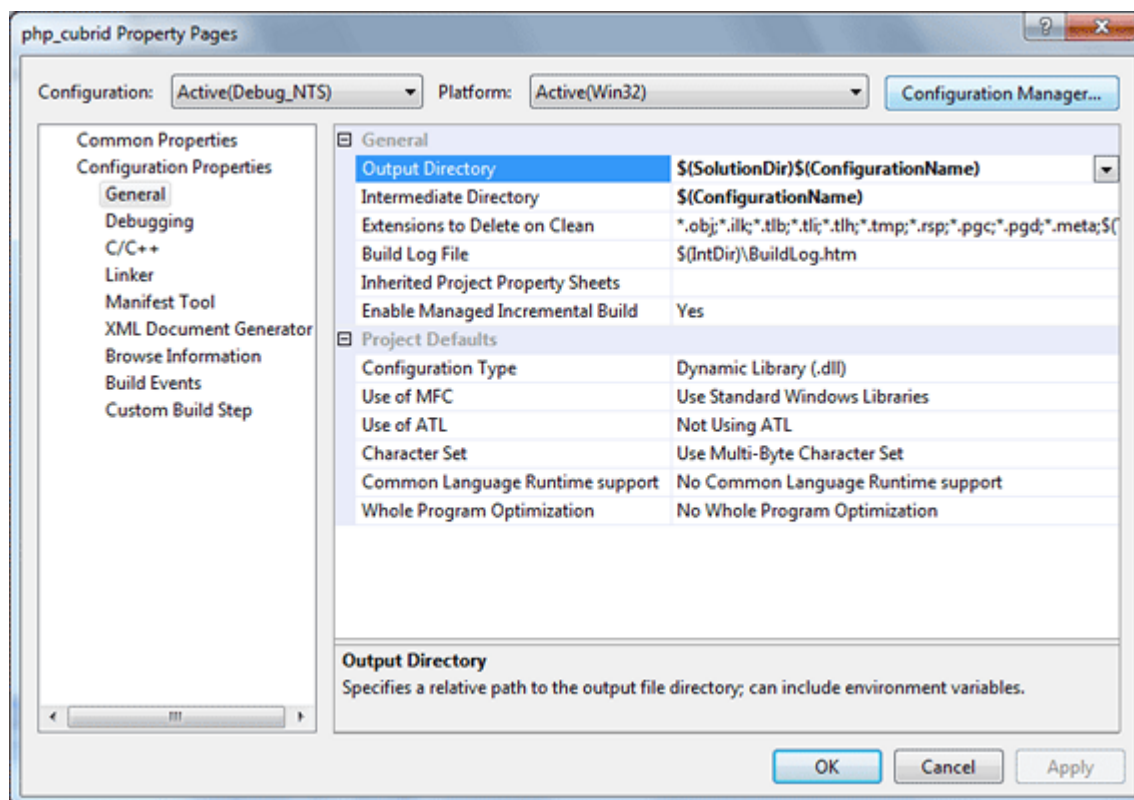
4. PHP 프로젝트를 설정하기 위해서 **configure** 를 실행한다.



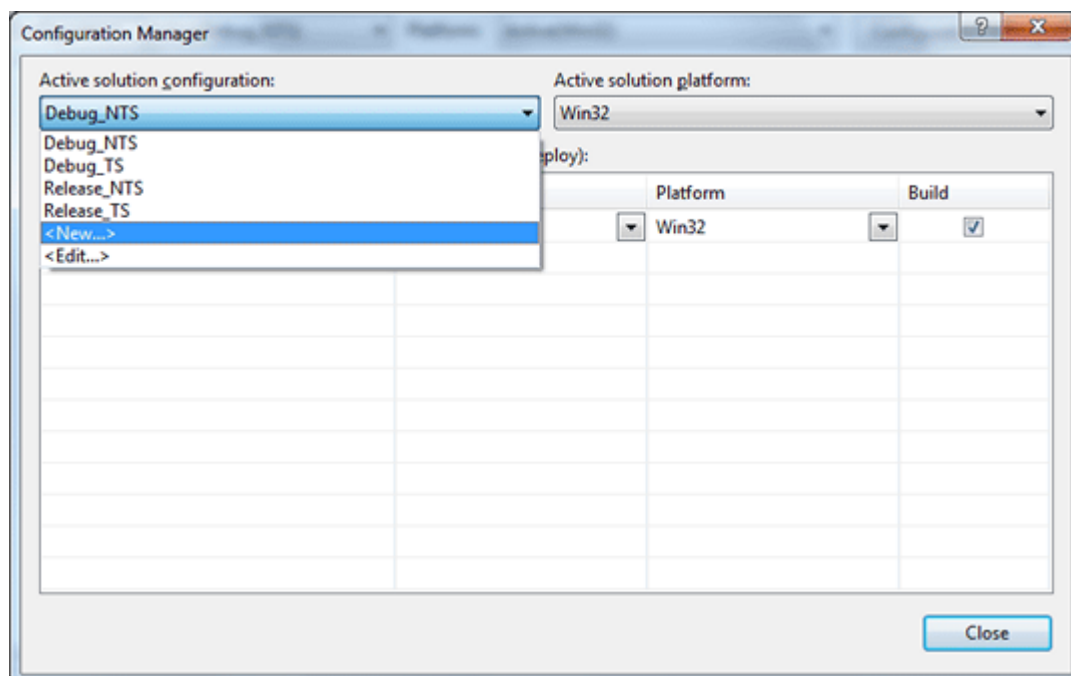


## CUBRID PHP 드라이버 빌드

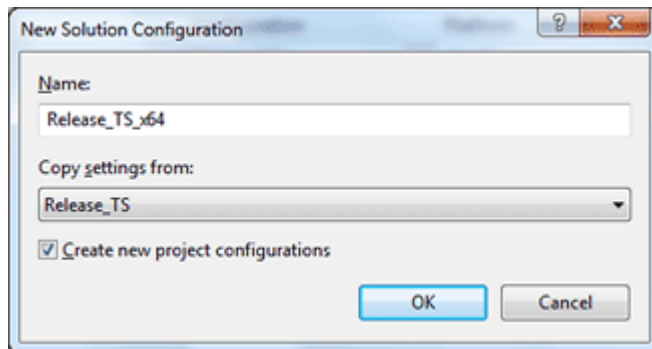
1. 다운로드한 CUBRID PHP 드라이버 소스코드의 \win 디렉터리에 있는 **php\_cubrid.vcproj** 파일을 열고, 왼쪽의 [Solution Explorer] 창에서 **php\_cubrid** 를 마우스 오른쪽 버튼으로 클릭하여 [Properties]를 선택한다.
2. [Property Page] 대화 상자에서 [Configuration Manager]을 클릭한다.



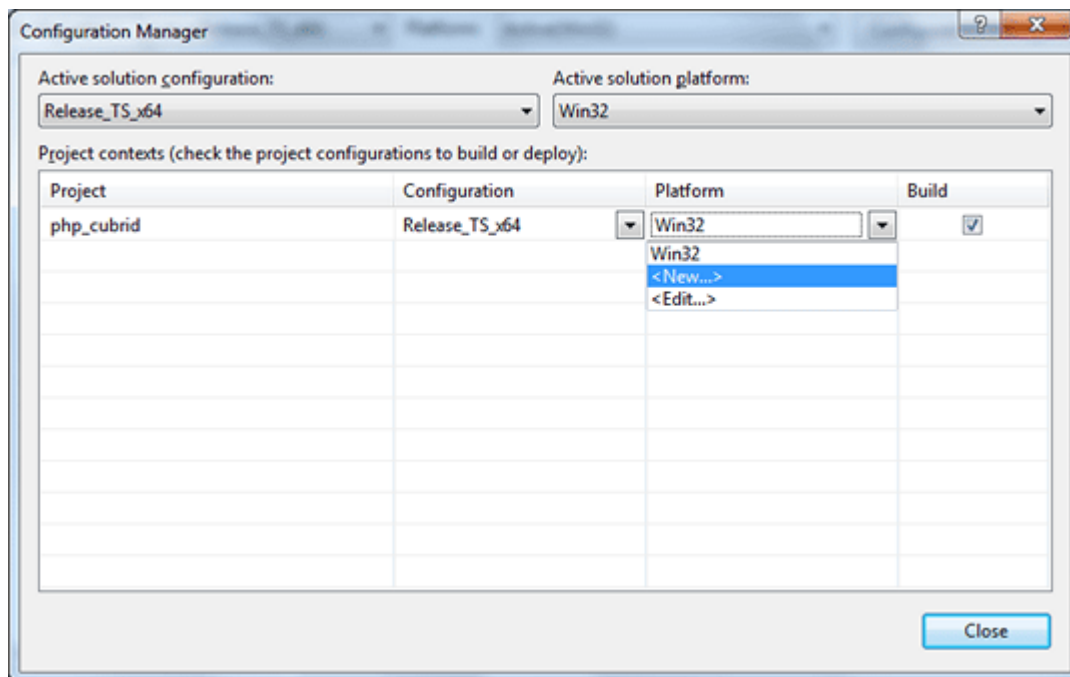
3. [Configuration Manager] 대화 상자의 [Active solution configuration]에는 네 가지 설정 (Release\_TS, Release\_NTS, Debug\_TS and Debug\_NTS)만 보인다. x64 CUBRID PHP 드라이버를 빌드하려면 새로운 설정을 생성해야 하므로 **New** 를 선택한다.



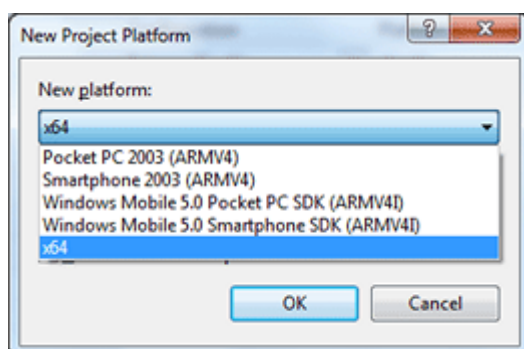
4. [New Solution Configuration] 대화상자에서 새로운 설정의 이름(예: Release\_TS\_x64)을 입력하고 [Copy settings from]에서 사용할 PHP와 같은 설정을 선택한다. 여기에서는 **Release\_TS** 를 선택했다. 선택한 후에 [OK]를 클릭한다.



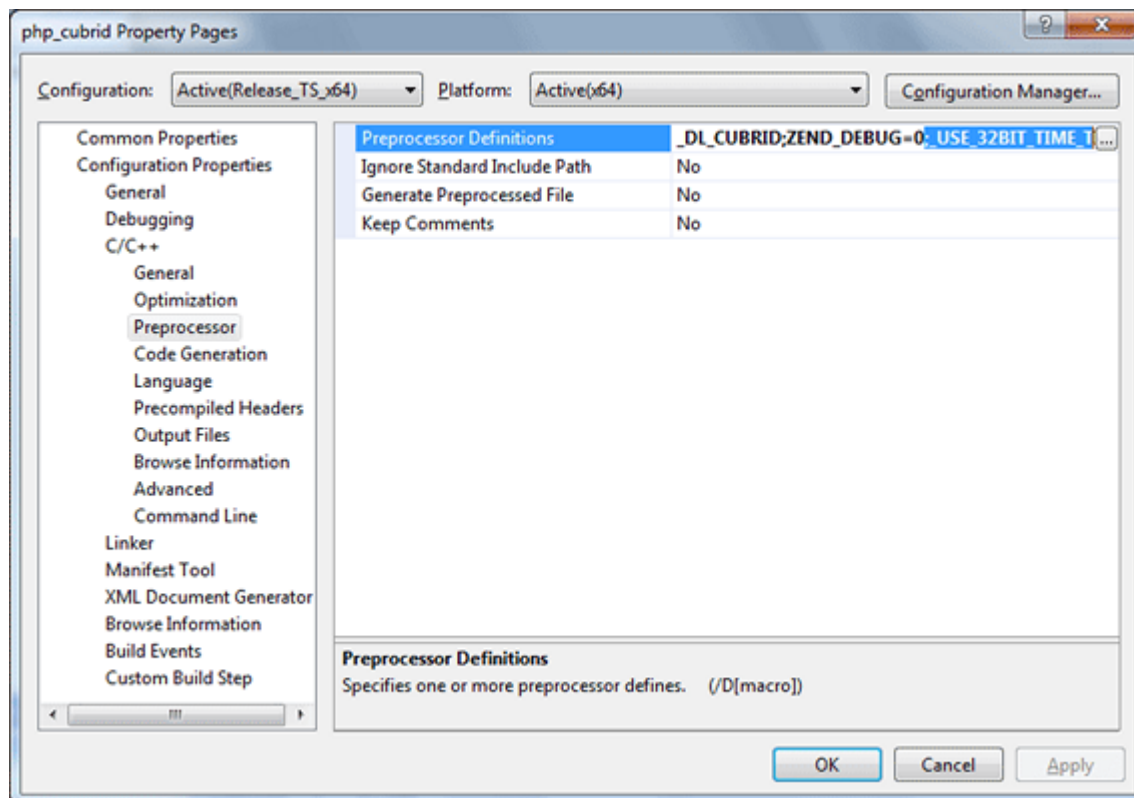
5. [Configuration Manager] 대화 상자에서 해당 프로젝트의 [Platform] 항목을 열어서 **x64** 가 있다면 **x64** 를 선택하고, 없으면 **New** 를 선택한다.



**New** 를 선택하면 [New Project Platform] 대화 상자가 나타난다. **x64** 를 선택하고 [OK]를 클릭한다.



6. [php\_cubrid Property Pages] 대화 상자에서 [C/C++] > [Preprocessor]를 선택하고, [Preprocessor Definitions]에서 **\_USE\_32BIT\_TIME\_T** 를 삭제한 후 [OK]를 클릭한다.



7. <F7> 키를 눌러 소스코드를 컴파일하면 x64 PHP 드라이버 파일이 생성된다.

## PHP 프로그래밍

### 데이터베이스 연결

데이터베이스 응용에서 첫 단계는 `cubrid_connect()` 함수 또는 `cubrid_connect_with_url()` 함수를 사용하는 것으로 데이터베이스 연결을 제공한다. `cubrid_connect` 함수 또는 `cubrid_connect_with_url()` 함수가 성공적으로 수행되면, 데이터베이스를 사용할 수 있는 모든 함수를 사용할 수 있다. 응용을 완전히 끝내기 전에 `cubrid_disconnect()` 함수를 호출하는 것은 매우 중요하다. `cubrid_disconnect()` 함수는 현재 발생한 트랜잭션을 끝나치고 `cubrid_connect()` 함수에 의해 생성된 연결 핸들과 모든 요청 핸들을 종료한다.

#### 참고

- 스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.
- 자동 커밋 모드에서 SELECT 문 수행 이후 모든 결과 셋이 fetch되지 않으면 커밋이 되지 않는다. 따라서, 자동 커밋 모드라 하더라도 프로그램 내에서 결과 셋에 대한 fetch 도중 어떠한 오류가 발생한다면 반드시 커밋 또는 롤백을 수행하여 트랜잭션을 종료 처리하도록 한다.

## 트랜잭션과 자동 커밋

CUBRID PHP는 트랜잭션과 자동 커밋 모드를 지원한다. 자동 커밋 모드에서는 하나의 질의마다 하나의 트랜잭션이 이루어진다. `cubrid_get_autocommit()` 함수를 사용하면 현재 연결의 자동 커밋 모드 여부를 확인할 수 있다. `cubrid_set_autocommit()` 함수를 사용하면 현재 연결의 자동 커밋 모드 여부를 설정할 수 있으며, 진행 중이던 트랜잭션은 모드 설정과 상관없이 커밋된다.

응용 프로그램 시작 시 자동 커밋 모드의 기본값은 브로커 파라미터인 **CCI\_DEFAULT\_AUTOCOMMIT** 으로 설정한다. 브로커 파라미터 설정을 생략하면 기본값은 **ON** 이다.

`cubrid_set_autocommit()` 함수에서 자동 커밋 모드를 OFF로 설정하면 커밋 또는 롤백을 명시하여 트랜잭션을 처리할 수 있다. 트랜잭션을 커밋하려면 `cubrid_commit()` 함수를 사용하고 트랜잭션을 롤백하려면 `cubrid_rollback()` 함수를 사용한다. `cubrid_disconnect()` 함수는 트랜잭션을 종료하고 커밋되지 않은 작업을 롤백한다.

## 질의 처리

### 질의 실행

다음은 질의 실행을 위한 기본 단계이다.

- 연결 핸들 생성
- SQL 질의 요청에 대한 요청 핸들 생성
- 결과 가져오기
- 요청 핸들 종료

```
$con = cubrid_connect("192.168.0.10", 33000, "demodb");
if($con) {
 $req = cubrid_execute($con, "select * from code");
 if($req) {
 while ($row = cubrid_fetch($req)) {
 echo $row["s_name"];
 echo $row["f_name"];
 }
 cubrid_close_request($req);
 }
 cubrid_disconnect($con);
}
```

### 질의 결과의 열 타입과 이름

`cubrid_column_types()` 함수를 사용하여 열 타입이 들어있는 배열을 얻을 수 있고, `cubrid_column_names()` 함수를 사용하여 열의 이름이 들어있는 배열을 얻을 수 있다.

```

$req = cubrid_execute($con, "select host_year, host_city from
olympic");
if($req) {
 $col_types = cubrid_column_types($req);
 $col_names = cubrid_column_names($req);

 while (list($key, $col_type) = each($col_types)) {
 echo $col_type;
 }
 while (list($key, $col_name) = each($col_names))
 echo $col_name;
 }
 cubrid_close_request($req);
}

```

## 커서 조정

질의 결과의 위치를 설정할 수 있다. `cubrid_move_cursor()` 함수를 사용하여 커서를 세 가지 포인트 (질의 결과의 처음, 현재 커서 위치, 질의 결과의 끝) 중 한 포인트로부터 일정한 위치로 이동할 수 있다.

```

$req = cubrid_execute($con, "select host_year, host_city from
olympic order by host_year");
if($req) {
 cubrid_move_cursor($req, 20, CUBRID_CURSOR_CURRENT)
 while ($row = cubrid_fetch($req, CUBRID_ASSOC)) {
 echo $row["host_year"]." ";
 echo $row["host_city"]."\n";
 }
}

```

## 결과 배열 타입

`cubrid_fetch()` 함수의 결과에는 세가지 종류의 배열 타입 중 하나가 사용된다. `cubrid_fetch()` 함수가 호출될 때 배열의 타입을 결정할 수 있다. 그 중 하나인 연관배열은 문자열 색인을 사용한다. 두 번째로 수치배열은 숫자 순서 색인을 사용한다. 마지막 배열은 연관배열과 수치배열을 둘 다 포함한다.

### · 수치배열

```

while (list($id, $name) = cubrid_fetch($req, CUBRID_NUM)) {
 echo $id;
 echo $name;
}

```

### · 연관배열

```
while ($row = cubrid_fetch($req, CUBRID_ASSOC)) {
 echo $row["id"];
 echo $row["name"];
}
```

## 카탈로그 연산

클래스, 가상 클래스, 속성, 메서드, 트리거, 제약 조건 등 데이터베이스의 스키마 정보는 `cubrid_schema()` 함수를 호출하여 얻을 수 있다. `cubrid_schema()` 함수의 리턴 값은 2차원 배열이다.

```
$pk = cubrid_schema($con, CUBRID_SCH_PRIMARY_KEY, "game");
if ($pk) {
 print_r($pk);
}

$fk = cubrid_schema($con, CUBRID_SCH_IMPORTED_KEYS, "game");
if ($fk) {
 print_r($fk);
}
```

## 에러 처리

에러가 발생하면 대부분의 PHP 인터페이스 함수는 에러 메시지를 출력하고 `false`나 `-1`을 반환한다. `cubrid_error_msg()`, `cubrid_error_code()` 그리고 `cubrid_error_code_facility()` 함수를 사용하면 각각 에러 메시지, 에러 코드, 에러 기능 코드를 확인할 수 있다.

`cubrid_error_code_facility()` 함수의 결과 값은 **CUBRID\_FACILITY\_DBMS** (DBMS 에러), **CUBRID\_FACILITY\_CAS** (CAS 서버 에러), **CUBRID\_FACILITY\_CCI** (CCI 에러), **CUBRID\_FACILITY\_CLIENT** (PHP 모듈 에러) 중 하나이다.

## OID 사용

`cubrid_execute()` 함수에서 `CUBRID_INCLUDE_OID` 옵션을 업데이트할 수 있는 질의를 함께 사용하면 `cubrid_current_oid` 함수를 통해 업데이트된 현재 f 레코드의 OID 값을 가져올 수 있다.

```
$req = cubrid_execute($con, "select * from person where id = 1",
 CUBRID_INCLUDE_OID);
if ($req) {
 while ($row = cubrid_fetch($req)) {
 echo cubrid_current_oid($req);
 echo $row["id"];
 echo $row["name"];
 }
 cubrid_close_request($req);
}
```

OID를 사용하여 인스턴스의 모든 속성, 지정한 속성 또는 한 속성의 값을 얻을 수 있다.

만약 `cubrid_get()` 함수에 속성을 명시하지 않으면 모든 속성의 값을 반환한다(a). 만약 배열 데이터 타입으로 속성을 명시하면 지정한 속성 값이 들어있는 배열은 연관배열로 반환된다(b). 만약 문자열 타입으로 한 속성을 명시하면 속성의 값이 반환된다(c).

```
$attrarray = cubrid_get ($con, $oid); // (a)
$attrarray = cubrid_get ($con, $oid, array("id", "name")); // (b)
$attrarray = cubrid_get ($con, $oid, "id"); // (c)
```

OID를 사용하여 인스턴스의 속성 값을 갱신할 수도 있다. 하나의 속성의 값을 갱신하려면 속성 이름을 문자열 타입으로 명시하고 값을 명시한다(a). 다중 속성의 값을 설정하려면 속성 명과 값을 연관 배열로 명시해야 한다(b).

```
$cubrid_put ($con, $oid, "id", 1); // (a)
$cubrid_put ($con, $oid, array("id"=>1, "name"=>"Tomas")); // (b)
```

## 컬렉션 사용

컬렉션 데이터 타입은 PHP 배열 데이터 타입을 통해 사용할 수 있고 배열 데이터 타입을 지원하는 PHP 함수를 사용할 수 있다. 다음은 `cubrid_fetch()` 함수를 사용하여 질의 결과를 가져오는 예제이다.

```
$row = cubrid_fetch ($req);
$col = $row["customer"];
while (list ($key, $cust) = each ($col)) {
 echo $cust;
}
```

컬렉션 속성의 값도 얻을 수 있다. 다음은 `cubrid_col_get()` 함수를 사용하여 컬렉션 속성 값을 가져오는 예제이다.

```
$tels = cubrid_col_get ($con, $oid, "tels");
while (list ($key, $tel) = each ($tels)) {
 echo $tel."\n";
}
```

`cubrid_set_add()` 함수와 `cubrid_set_drop()` 함수를 사용하면 컬렉션 타입의 값을 직접적으로 갱신할 수 있다.

```
$tels = cubrid_col_get ($con, $oid, "tels");
while (list ($key, $tel) = each ($tels)) {
 $res = cubrid_set_drop ($con, $oid, "tel", $tel);
}
```



```
cubrid_commit ($con);
```

### 참고

칼럼에서 정의한 크기보다 큰 문자열을 **INSERT** / **UPDATE** 하면 문자열이 잘려서 입력된다.

## PHP API

[http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/)를 참고한다.

## PDO 드라이버

공식 CUBRID PDO(PHP Data Objects) 드라이버는 PECL 패키지로 제공되며, PDO에서 CUBRID 데이터베이스에 접근할 수 있도록 PDO 인터페이스를 제공한다. PDO는 PHP 5.1과 함께 배포되며, PHP 5.0에서는 PECL 확장으로 사용할 수 있다. PDO를 사용하려면 PHP 5 코어의 OO 기능이 필요하므로, PHP 5.0 미만 버전에서는 사용할 수 없다.

PDO는 어떤 데이터베이스를 사용하든 같은 함수를 사용할 수 있게 하는 데이터 액세스 추상화 계층(data-access abstraction layer)을 제공하며, 데이터베이스 추상화를 제공하는 것은 아니다. PDO를 데이터베이스 인터페이스 계층으로 사용하면 PHP 데이터베이스 드라이버를 바로 사용하는 경우에 비해 다음과 같은 이점이 있다.

- 작성한 PHP 코드를 다양한 데이터베이스와 함께 사용할 수 있다.
- SQL 파라미터와 바인딩을 지원한다.
- SQL을 더 안전하게 사용할 수 있다(구문 검사, 이스케이프, SQL 인젝션 방지 등).
- 프로그래밍 모델이 간결해진다.

따라서 CUBRID PDO 드라이버를 사용하면, 데이터베이스 인터페이스 계층으로 PDO를 사용하는 모든 응용 프로그램은 CUBRID와 함께 사용할 수 있다.

CUBRID PDO 드라이버는 CCI API를 기반으로 작성되었으므로, CCI API 및 CCI에 적용되는 **CCI\_DEFAULT\_AUTOCOMMIT**과 같은 설정 파라미터에 영향을 받는다.

## PDO 설치 및 설정

### Linux

#### 기본 환경

- 운영체제: Linux: 32 비트 또는 64비트
- 웹 서버: Apache
- PHP: 5.6.x, 7.1.x, 또는 7.4.x (<https://www.php.net/downloads.php>)

#### PECL을 이용한 설치

**PECL**이 설치되어 있다면, **PECL**이 소스코드 다운로드 및 컴파일을 수행하므로 다음과 같이 간단하게 CUBRID PDO 드라이버를 설치할 수 있다.

1. 다음과 같은 명령어를 입력하여 CUBRID PDO 드라이버 최신 버전을 설치한다.

```
sudo pecl install pdo_cubrid
```

하위 버전의 드라이버가 필요하면 다음과 같이 설치할 버전을 지정할 수 있다.

```
sudo pecl install pdo_cubrid-8.3.1.0003
```

설치가 진행되는 중에 **CUBRID base install dir autodetect** :라는 프롬프트가 표시된다. 설치를 원활하게 진행하기 위해서 CUBRID를 설치한 디렉터리의 전체 경로를 입력한다. 예를 들어 CUBRID가 **/home/cubridtest/CUBRID** 디렉터리에 설치되었다면, **/home/cubridtest/CUBRID**를 입력한다.

2. 설정 파일을 수정한다.

- CentOS 6.0 이상 버전이나 Fedora 15 이상 버전을 사용한다면 **pdo\_cubrid.ini** 파일을 생성하고 내용에 **extension=pdo\_cubrid.so** 를 입력하여 **/etc/php.d** 디렉터리에 저장한다.
- 다른 운영체제를 사용한다면 **php.ini** 파일 끝에 다음 두 줄의 내용을 추가한다. **php.ini** 파일의 기본 위치는 **/etc/php5/apache2** 또는 **/etc** 이다.

```
[CUBRID]
extension=pdo_cubrid.so
```

3. 변경된 내용을 반영하려면 웹 서버를 재시작한다.

## Windows

### 기본 환경

- 운영체제: Windows 32 비트 또는 64비트
- 웹 서버: Apache 또는 IIS
- PHP: 5.6.x, 7.1.x 또는 7.4.x(<https://windows.php.net/download/>)
- PHP 7.1.x 또는 7.4.x 의 경우 32bit 또는 64bit용 Microsoft Visual C ++ 2015 재배포 가능 패키지를 설치해야 한다.

### 빌드된 드라이버 다운로드 및 설치

운영체제와 PHP 버전에 맞는 Windows용 CUBRID PHP/PDO 드라이버를 <https://www.cubrid.org/downloads#pdo> 에서 다운로드한다.

PDO 드라이버를 다운로드하면 **php\_cubrid.dll** 파일을 볼 수 있으며, PDO 드라이버를 다운로드하면 **php\_pdo\_cubrid.dll** 파일을 볼 수 있다. 드라이버를 설치하는 방법은 다음과 같다.

1. 드라이버 파일을 기본 PHP 확장 디렉터리( **C:\Program Files\PHP\ext** )에 복사한다.
2. 시스템 환경 변수를 설정한다. 시스템 환경 변수 **PHPRC**의 값으로 **C:\Program Files\PHP**가 설정되고, **Path**에 **%PHPRC%**와 **%PHPRC%\ext**가 추가되었는지 확인한다.
3. **php.ini** ( **C:\Program Files\PHP\php.ini** ) 파일을 열어 끝에 다음 두 줄을 추가한다.

```
[PHP_CUBRID]
extension=php_cubrid.dll
```

PDO 드라이버의 경우에는 다음 내용을 추가한다.

```
[PHP_PDO_CUBRID]
extension = php_pdo_cubrid.dll
```

4. 웹 서버를 재시작한다.

### PHP 드라이버 빌드

CUBRID PDO 드라이버를 빌드하는 방법은 **CUBRID PHP 드라이버를 빌드하는 방법**과 동일하므로 참고하여 진행한다.

### PDO 프로그래밍

## 데이터 원본 이름(DSN)

PDO\_CUBRID 데이터 원본 이름(DSN)은 다음과 같은 요소로 구성된다.

| 요소      | 설명                                 |
|---------|------------------------------------|
| DSN 접두어 | DSN 접두어(prefix)는 <b>cubrid</b> 이다. |
| host    | 데이터베이스 서버가 위치한 호스트의 이름을 입력한다.      |
| port    | 데이터베이스 서버의 수신 대기 포트 번호를 입력한다.      |
| dbname  | 데이터베이스의 이름을 입력한다.                  |

### 예제

```
"cubrid:host=127.0.0.1;port=33000;dbname=demodb"
```

## 미리 정의된 상수

CUBRID PDO 드라이버에 의해 정의되는 상수(predefined constants)는 CUBRID PDO 드라이버가 PHP 와 함께 컴파일되거나 런타임에 동적으로 로드되는 경우에만 사용할 수 있다. 이처럼 특정 드라이버에 의해 정의된 상수를 다른 드라이버와 함께 사용하면 예상과 다르게 동작할 수도 있다.

코드가 여러 개의 드라이버와 함께 실행될 수 있다면, **PDO\_ATTR\_DRIVER\_NAME** 속성 값을 얻어 드라이버를 확인하기 위해 `PDO::getAttribute()` 함수를 사용할 수 있다.

다음 상수는 `PDO::cubrid_schema()` 함수를 이용하여 스키마 정보를 얻을 때 사용할 수 있다.

| 상수                    | 타입      | 설명                       |
|-----------------------|---------|--------------------------|
| PDO::CUBRID_SCH_TABLE | integer | CUBRID 테이블의 이름과 타입을 얻는다. |

| 상수                              | 타입      | 설명                                                                                                   |
|---------------------------------|---------|------------------------------------------------------------------------------------------------------|
| PDO::CUBRID_SCH_VIEW            | integer | CUBRID 뷰의 이름과 타입을 얻는다.                                                                               |
| PDO::CUBRID_SCH_QUERY_SPEC      | integer | 뷰의 쿼리 정의를 얻는다.                                                                                       |
| PDO::CUBRID_SCH_ATTRIBUTE       | integer | 테이블 칼럼의 속성을 얻는다.                                                                                     |
| PDO::CUBRID_SCH_TABLE_ATTRIBUTE | integer | 테이블의 속성을 얻는다.                                                                                        |
| PDO::CUBRID_SCH_TABLE_METHOD    | integer | 인스턴스 메서드를 얻는다. 인스턴스 메서드는 클래스 인스턴스가 호출하는 메서드이다. 대부분의 작업은 인스턴스에서 실행되기 때문에 인스턴스 메서드는 클래스 메서드보다 자주 사용된다. |
| PDO::CUBRID_SCH_METHOD_FILE     | integer | 테이블의 메서드가 정의된 파일의 정보를 얻는다.                                                                           |
| PDO::CUBRID_SCH_SUPER_TABLE     | integer | 현재 테이블에 속성을 상속한 테이블의 이름과 타입을 얻는다.                                                                    |
| PDO::CUBRID_SCH_SUB_TABLE       | integer | 현재 테이블로부터 속성을 상속받은 테이블의 이름과 타입을 얻는다.                                                                 |
| PDO::CUBRID_SCH_CONSTRAINT      | integer | 테이블의 제약 조건을 얻는다.                                                                                     |

| 상수                                 | 타입      | 설명                       |
|------------------------------------|---------|--------------------------|
| PDO::CUBRID_SCH_TRIGGER            | integer | 테이블의 트리거를 얻는다.           |
| PDO::CUBRID_SCH_TABLE_PRIVILEGE    | integer | 테이블의 권한 정보를 얻는다.         |
| PDO::CUBRID_SCH_COL_PRIVILEGE      | integer | 칼럼의 권한 정보를 얻는다.          |
| PDO::CUBRID_SCH_DIRECT_SUPER_TABLE | integer | 현재 테이블의 바로 상위 테이블을 얻는다.  |
| PDO::CUBRID_SCH_DIRECT_PRIMARY_KEY | integer | 테이블의 기본키를 얻는다.           |
| PDO::CUBRID_SCH_IMPORTED_KEYS      | integer | 테이블의 외래키가 참조하는 기본키를 얻는다. |
| PDO::CUBRID_SCH_EXPORTED_KEYS      | integer | 테이블의 기본키를 참조하는 외래키를 얻는다. |
| PDO::CUBRID_SCH_CROSS_REFERENCE    | integer | 두 테이블 간의 상호 참조 관계를 얻는다.  |

## PDO 예제 프로그램

### CUBRID PDO 드라이버 확인

사용 가능한 PDO 드라이버를 확인하려면 다음과 같이 `PDO::getAvailableDrivers()` 함수를 사용한다.

```
<?php
echo 'PDO Drivers available:
';
foreach(PDO::getAvailableDrivers() as $driver)
```

```
{
if($driver == "cubrid"){
echo " - Driver: ".$driver.""
';
}else{
echo " - Driver: ".$driver.'
';
}
}
?>
```

위 스크립트는 다음과 같이 설치된 PDO 드라이버를 출력한다.

```
PDO Drivers available:
- Driver: mysql
- Driver: pgsql
- Driver: sqlite
- Driver: sqlite2
- Driver: cubrid
```

## CUBRID 연결

데이터 원본 이름(DSN)을 사용하여 데이터베이스에 연결한다. 데이터 원본 이름에 대한 자세한 설명은 [데이터 원본 이름\(DSN\)](#)을 참고한다.

다음은 *demodb*라는 CUBRID 데이터베이스에 PDO 연결을 수행하는 간단한 PHP 스크립트이다. PDO에서는 try-catch로 오류를 처리하며, 연결을 해제할 때에는 연결 객체에 **NULL**을 할당한다는 것을 알 수 있다.

```
<?php
$database = "demodb";
$host = "localhost";
$port = "30000";//use default value
$username = "dba";
$password = "";

try{
//cubrid:host=localhost;port=33000;dbname=demodb
$conn_str = "cubrid:dbname=".$database.";host=".$host.";port=".$port;
echo "PDO connect string: ".$conn_str."
";
$db = new PDO($conn_str, $username, $password);
echo "PDO connection created ok!."
";
$db = null;//disconnect
}catch(PDOException $e){
echo "Error: ".$e->getMessage()."
";
```

```
}
?>
```

연결에 성공하면 다음과 같은 스크립트가 출력된다.

```
PDO connect string: cubrid:dbname=demodb;host=localhost;port=30000
PDO connection created ok!
```

## SELECT 실행

PDO에서 SQL 질의를 수행하려면 질의나 응용 프로그램의 성격에 따라 다음 중 하나의 방법을 사용할 수 있다.

- `query()` 함수 사용
- prepared statements(`prepare()`/ `execute()`) 함수 사용
- `exec()` 함수 사용

다음 예제에서는 가장 간단한 `query()` 함수를 사용한다. 리턴 값은 PDOStatement 객체인 resultset에서 `$rs["column_name"]`와 같이 칼럼 이름을 이용하여 얻을 수 있다.

`query()` 함수를 사용할 때에는 질의 코드가 제대로 이스케이프되었는지 확인해야 한다. 이스케이프에 대한 내용은 `PDO::quote()`를 참고한다.

```
<?php
include("_db_config.php");
include("_db_connect.php");

$sql = "SELECT * FROM code";
echo "Executing SQL: ".$sql."";
';
echo '
';

try{
foreach($db->query($sql)as $row){
echo $row['s_name'].' - '. $row['f_name'].'
';
}
}catch(PDOException $e){
echo $e->getMessage();
}

$db = null;//disconnect
?>
```



위 스크립트의 결과는 다음과 같이 출력된다.

```
Executing SQL: SELECT * FROM code
```

```
X - Mixed
W - Woman
M - Man
B - Bronze
S - Silver
G - Gold
```

## UPDATE 실행

다음은 prepared statement와 파라미터를 사용하여 UPDATE를 실행하는 예제이다. prepared statement 대신 `exec()` 함수를 사용할 수도 있다.

```
<?php
include("_db_config.php");
include("_db_connect.php");

$s_name = 'X';
$f_name = 'test';
$sql = "UPDATE code SET f_name=:f_name WHERE s_name=:s_name";

echo "Executing SQL: ".$sql."
';
echo '
';

echo ":f_name: ".$f_name."
';
echo '
';
echo ":s_name: ".$s_name."
';
echo '
';

$qe = $db->prepare($sql);
$qe->execute(array(':s_name'=>$s_name, ':f_name'=>$f_name));

$sql = "SELECT * FROM code";
echo "Executing SQL: ".$sql."
';
echo '
';

try{
foreach($db->query($sql)as $row){
```

```

echo $row['s_name'].' - '. $row['f_name'].'
';
}
}catch(PDOException $e){
echo $e->getMessage();
}

$db = null;//disconnect
?>

```

위 스크립트의 실행 결과는 다음과 같이 출력된다.

```

Executing SQL: UPDATE code SET f_name=:f_name WHERE s_name=:s_name

:f_name: test

:s_name: X

Executing SQL: SELECT * FROM code

X - test
W - Woman
M - Man
B - Bronze
S - Silver
G - Gold

```

## prepare와 bind

prepared statement는 PDO가 제공하는 유용한 기능 중 하나로, 사용하면 다음과 같은 이점이 있다.

- SQL prepared statement는 다양한 파라미터와 함께 여러 번 실행되어도 한 번만 파싱하면 된다. 따라서 여러 번 실행되는 SQL문에 prepared statement를 사용하면 CUBRID 응용 프로그램의 성능을 높일 수 있다.
- 수동으로 파라미터를 이스케이프할 필요가 없으므로 SQL 인젝션 공격을 방지할 수 있다(그러나 SQL 질의의 다른 부분이 이스케이프되지 않은 입력으로 구성된다면 SQL 인젝션을 완전히 막을 수는 없다).

다음은 prepared statement를 이용하여 데이터를 조회하는 예이다.

```

<?php
include("_db_config.php");
include("_db_connect.php");

$sql ="SELECT * FROM code WHERE s_name NOT LIKE :s_name";
echo"Executing SQL: ".$sql.""

```

```
' ;

$s_name = 'xyz';
echo "s_name: ".$s_name."";
';

echo '
';

try{
$stmt = $db->prepare($sql);

$stmt->bindParam(':s_name', $s_name, PDO::PARAM_STR);
$stmt->execute();

$result = $stmt->fetchAll();
foreach($result as $row)
{
echo $row['s_name'].' - '. $row['f_name'].'
';
}
}catch(PDOException $e){
echo $e->getMessage();
}
echo '
';

$sql = "SELECT * FROM code WHERE s_name NOT LIKE :s_name";
echo "Executing SQL: ".$sql."";
';

$s_name = 'X';
echo "s_name: ".$s_name."";
';

echo '
';

try{
$stmt = $db->prepare($sql);

$stmt->bindParam(':s_name', $s_name, PDO::PARAM_STR);
$stmt->execute();

$result = $stmt->fetchAll();
foreach($result as $row)
{
echo $row['s_name'].' - '. $row['f_name'].'
';
}
$stmt->closeCursor();
}catch(PDOException $e){
```

```
echo $e->getMessage();
}
echo '
';

$db = null;//disconnect
?>
```

위 스크립트의 결과는 다음과 같이 출력된다.

```
Executing SQL: SELECT * FROM code WHERE s_name NOT LIKE :s_name
:s_name: xyz

X - Mixed
W - Woman
M - Man
B - Bronze
S - Silver
G - Gold

Executing SQL: SELECT * FROM code WHERE s_name NOT LIKE :s_name
:s_name: X

W - Woman
M - Man
B - Bronze
S - Silver
G - Gold
```

## PDO::getAttribute() 사용

`PDO::getAttribute()` 함수는 다음과 같은 데이터베이스 연결 속성을 조회할 때 유용하다.

- 드라이버 이름
- 데이터베이스 버전
- 자동 커밋 모드 여부
- 오류 모드

속성을 쓸 수 있다면 `PDO::setAttribute()` 함수를 사용하여 속성을 설정할 수 있다.

다음은 `PDO::getAttribute()` 함수를 사용하여 클라이언트와 서버의 현재 버전을 조회하는 PHP PDO 스크립트이다.

```
<?php
include("_db_config.php");
```

```
include("_db_connect.php");

echo "Driver name: ".$db->getAttribute(PDO::ATTR_DRIVER_NAME)."";
echo "
";
echo "Client version: ".$db->getAttribute(PDO::ATTR_CLIENT_VERSION)."";
echo "
";
echo "Server version: ".$db->getAttribute(PDO::ATTR_SERVER_VERSION)."";
echo "
";

$db = null; //disconnect
?>
```

위 스크립트의 결과는 다음과 같이 출력된다.

```
Driver name: cubrid
Client version: 8.3.0
Server version: 8.3.0.0337
```

## CUBRID PDO 확장

CUBRID PDO 확장은 데이터베이스 스키마와 메타데이터 정보를 조회하는 데 사용할 수 있는 PDO::cubrid\_schema() 함수를 제공한다. 다음은 이 함수를 이용하여 *nation* 테이블의 기본키를 반환하는 스크립트이다.

```
<?php
include("_db_config.php");
include("_db_connect.php");
try{
echo "Get PRIMARY KEY for table: nation:

";
$pk_list = $db->cubrid_schema(PDO::CUBRID_SCH_PRIMARY_KEY,"nation");
print_r($pk_list);
}catch(PDOException $e){
echo $e->getMessage();
}

$db = null; //disconnect
?>
```

위 스크립트의 결과는 다음과 같이 출력된다.

```
Get PRIMARY KEY for table: nation:
Array ([0] => Array ([CLASS_NAME] => nation [ATTR_NAME] => code
[KEY_SEQ] => 1 [KEY_NAME] => pk_nation_code))
```

## PDO API

PDO API와 관련하여 <https://www.php.net/manual/en/book.pdo.php>를 참고한다.

CUBRID PDO 드라이버가 제공하는 PDO API는 [http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/PDO/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/PDO/)를 참고한다.

## ODBC 드라이버

CUBRID ODBC 드라이버는 ODBC 3.52 버전을 지원하며, ODBC 코어 부분과 Level 1과 Level 2 일부 API를 지원한다. CUBRID ODBC 드라이버는 ODBC Spec 3.x를 기반으로 구현되었으므로 ODBC Spec 2.x을 이용하여 작성한 프로그램에 대해서 하위 호환성을 완벽하게 보장하지는 않는다.

CUBRID ODBC 드라이버는 CCI API를 기반으로 작성되었지만, 예외적으로 `CCI_DEFAULT_AUTOCOMMIT` 파라미터의 영향을 받지 않는다.

### 참고

ODBC가 `CCI_DEFAULT_AUTOCOMMIT`의 영향을 받지 않는 것은 9.3 버전부터이다. 그 이전 버전에서는 `CCI_DEFAULT_AUTOCOMMIT`를 OFF로 설정해야 한다.

## CUBRID와 ODBC의 데이터 타입 매핑

다음은 CUBRID가 지원하는 데이터 타입과 ODBC 데이터 타입을 매핑한 표이다.

| CUBRID 데이터 타입 | ODBC 데이터 타입     |
|---------------|-----------------|
| CHAR          | SQL_CHAR        |
| VARCHAR       | SQL_VARCHAR     |
| STRING        | SQL_LONGVARCHAR |

| CUBRID 데이터 타입            | ODBC 데이터 타입                    |
|--------------------------|--------------------------------|
| BIT                      | SQL_BINARY                     |
| BIT VARYING              | SQL_VARBINARY                  |
| NUMERIC                  | SQL_NUMERIC                    |
| INT                      | SQL_INTEGER                    |
| SHORT                    | SQL_SMALLINT                   |
| MONETARY                 | SQL_DOUBLE                     |
| FLOAT                    | SQL_FLOAT                      |
| DOUBLE                   | SQL_DOUBLE                     |
| BIGINT                   | SQL_BIGINT                     |
| DATE                     | SQL_TYPE_DATE                  |
| TIME                     | SQL_TYPE_TIME                  |
| TIMESTAMP                | SQL_TYPE_TIMESTAMP             |
| DATETIME                 | SQL_TYPE_TIMESTAMP             |
| OID                      | SQL_CHAR(32)                   |
| SET, MULTISSET, SEQUENCE | SQL_VARCHAR(MAX_STRING_LENGTH) |

**CUBRID 데이터 타입****ODBC 데이터 타입**

ENUM

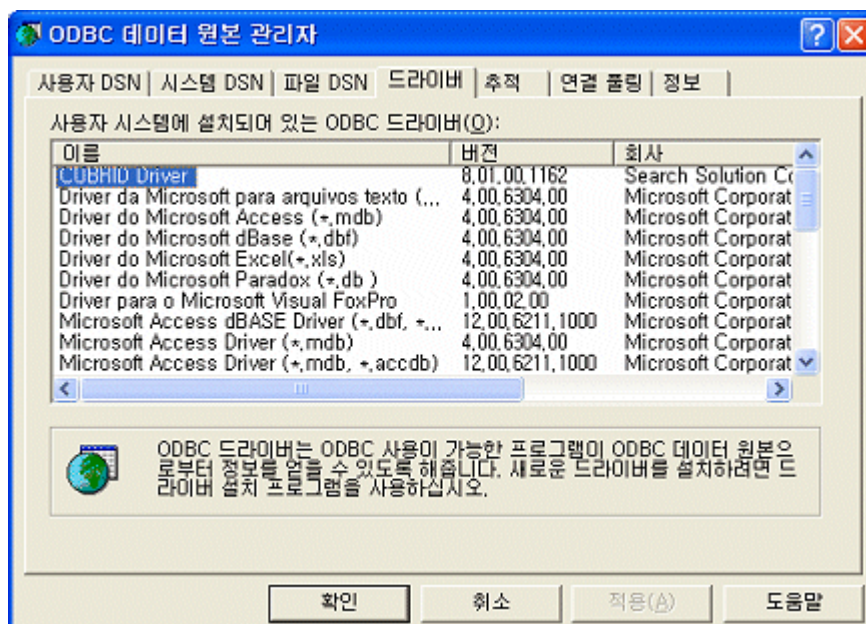
SQL\_VARCHAR

**ODBC 설치 및 설정****기본 환경**

- CUBRID 2008 R4.4 (8.4.4) 이상 32비트 또는 64비트

**ODBC 드라이버 설정**

CUBRID ODBC Driver는 CUBRID 설치 시 자동으로 설치되며, [제어판] > [관리 도구] > [데이터 원본 (ODBC)]을 실행하면 [드라이버] 탭에서 확인할 수 있다.

**64비트 Windows에서 32비트 ODBC 드라이버 선택**

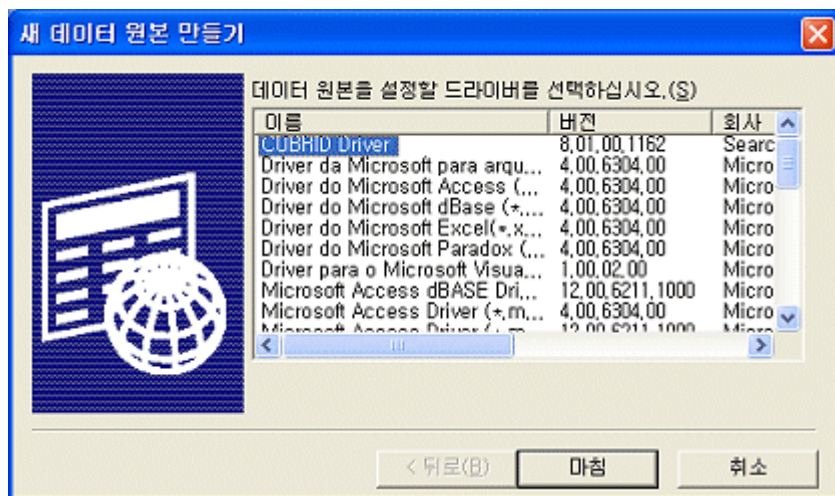
32비트 응용 프로그램을 수행하는 경우, 32비트 ODBC 드라이버를 사용해야 한다. 64비트 Windows OS에서 32비트 ODBC 드라이버를 선택해야 하는 경우, C:\WINDOWSSysWOW64\odbcad32.exe를 실행하면 된다.

마이크로소프트 윈도우즈 64비트 플랫폼은 32비트 응용 프로그램이 64비트 환경에서 수행 가능한 환경을 제공하는데, 이를 WOW64(Windows-32-on-Windows-64)라고 한다. 이 환경은 32비트 전용 레지스트리 복사본을 유지하고 있다.

**DSN 설정**



CUBRID ODBC Driver가 확인되었다면 응용 프로그램에서 접속하려는 데이터베이스로 DSN을 설정한다. DSN 설정을 위해 ODBC 데이터 원본 관리자에서 [추가] 버튼을 클릭하면 다음과 같은 대화 상자가 나타난다. **CUBRID Driver** 를 선택하고 [마침] 버튼을 클릭한다.



[Config CUBRID Data Sources] 대화 상자가 나타나면 다음과 같은 내용을 입력한다.

Config CUBRID Data Sources

Data Source: CUBRID Driver

Description: CUBRID ODBC Driver

Database: demodb

User: dba

Password:

Server: localhost

Port: 33000

FETCH\_SIZE: 100

Charset: utf-8

Autocommit: on

Omit Schmea: no

OK Cancel

- **DSN** : 데이터 원본의 이름을 설정한다.

- **DB Name** : 접속할 데이터베이스 이름을 입력한다.
- **DB User** : 데이터베이스 사용자 이름을 입력한다.
- **Password** : 데이터베이스 사용자 비밀번호를 입력한다.
- **Server Address** : 데이터베이스의 호스트 주소를 입력한다. **localhost** 또는 다른 서버의 IP 주소를 입력한다.
- **Server Port** : CUBRID 브로커의 포트 번호를 입력한다. 기본값은 33000이다. CUBRID 브로커 포트 번호를 확인하려면 **cubrid\_broker.conf** 파일에서 **BROKER\_PORT** 값을 확인하거나, 터미널에서 **cubrid service status** 명령을 입력한다. 터미널에서 입력했을 때 화면은 다음과 같다.

```

Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

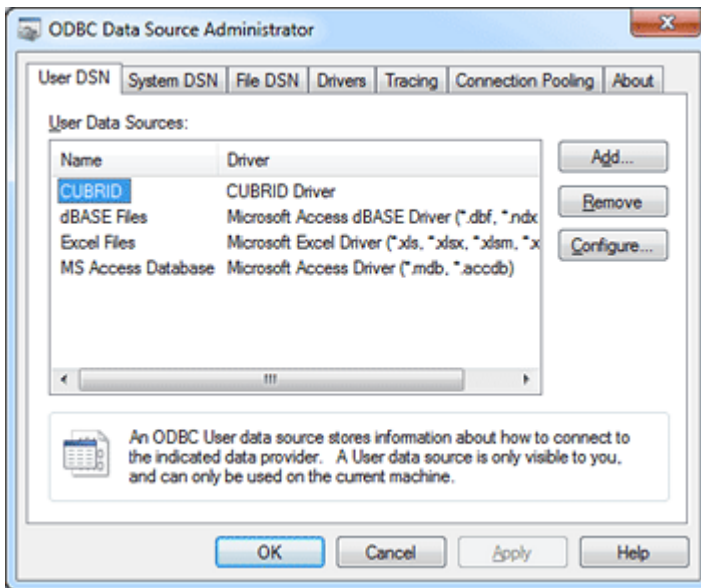
C:\Users\cub>cubrid service status
@ cubrid master status
++ cubrid master is running.
@ cubrid server status
@ cubrid broker status
 NAME PID PORT AS JQ REQ TPS QPS LONG-T LONG-Q ERR-Q
=====
* broker1 2724 33000 5 0 0 0 0 0/0.0 0/0.0 0
* query_editor 3112 30000 5 0 0 0 0 0/0.0 0/0.0 0
@ cubrid manager server status
++ cubrid manager server is running.
@ cubrid replication status

C:\Users\cub>_

```

- **Charset** : 데이터베이스의 문자셋(charSet) 입력한다.
- **Autocommit** : Autocommit mode를 입력한다. 기본값은 OFF이다.
- **Omit Schema** : 스키마 개념을 지원하는 11.2 버전 이상의 데이터베이스 서버에서 스키마 정보 관련 함수 사용시 로그인한 사용자가 생성한 스키마만을 제공하는 기능이다. 이 기능은 **ERWin™** 이 CUBRID 11.2 버전 이상을 사용할 경우 필요하다 (omit\_schema=yes). **YES** 인 경우 로그인한 사용자가 소유한 테이블/뷰/Primary Key/Foreign Key만 제공되며, 이때 소유자의 이름은 제거된다. 기본값은 **NO** 이다
- **FETCH\_SIZE** : ODBC 드라이버가 내부적으로 사용하는 CCI 라이브러리의 `cci_fetch()` 함수를 호출할 때마다 서버로부터 fetch하는 레코드의 개수를 설정한다.

위와 같이 입력한 후 [확인]을 클릭하면 다음과 같이 [User Data Sources]에 데이터 원본이 추가된 것을 확인할 수 있다.



### DSN을 사용하지 않고 데이터베이스에 직접 연결

연결 문자열을 사용하여 CUBRID 데이터베이스에 직접 연결할 수도 있다. 연결 문자열의 예는 다음과 같다.

```
conn = "driver={CUBRID
Driver};server=localhost;port=33000;uid=dba;pwd=;db_name=demodb;"
```

#### 참고

CUBRID 데이터베이스에 연결하기 전에 데이터베이스를 먼저 시작하지 않으면 ODBC 오류가 발생한다. 데이터베이스(demodb)를 시작하려면 터미널에 **cubrid server start demodb** 명령을 입력한다.

## ODBC 프로그래밍

### 연결 문자열(connection string) 구성

CUBRID ODBC 프로그래밍을 할 때 연결 문자열(connection string)은 다음과 같이 작성한다.

| 항목     | 예                     | 설명      |
|--------|-----------------------|---------|
| Driver | CUBRID Driver Unicode | 드라이버 이름 |

| 항목          | 예           | 설명                               |
|-------------|-------------|----------------------------------|
| UID         | PUBLIC      | 사용자 아이디                          |
| PWD         | xxx         | 비밀번호                             |
| FETCH_SIZE  | 100         | Fetch 크기                         |
| PORT        | 33000       | 브로커 포트 번호                        |
| SERVER      | 127.0.0.1   | CUBRID 브로커 서버 IP 주소<br>또는 호스트 이름 |
| DB_NAME     | demodb      | 데이터베이스 이름                        |
| DESCRIPTION | cubrid_test | 설명                               |
| CHARSET     | utf-8       | 문자셋                              |
| AUTOCOMMIT  | ON          | Autocommit mode                  |
| OMIT_SCHEMA | NO          | 단일 스키마 특징 제공 (11.2버전 이상만 적용)     |

위의 예를 이용한 연결 문자열은 다음과 같다.

```
"DRIVER={CUBRID Driver
Unicode};UID=PUBLIC;PWD=xxx;FETCH_SIZE=100;PORT=33000;SERVER=127.0.0.1;DB_NAME=demodb;DESCRIPTION=cubrid_test;CHARSET=utf-8;OMIT_SCHEMA=YES"
```

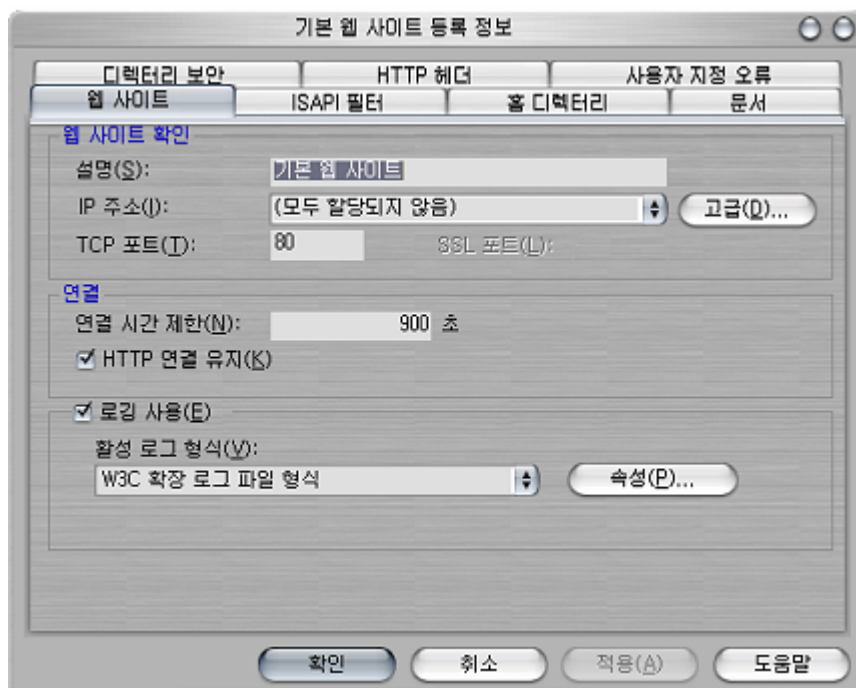
UTF-8 유니코드를 사용하는 경우, 파일 이름 중간에 “unicode”가 쓰여있는 유니코드 전용 드라이버를 설치하고, 연결 문자열에서 드라이버의 이름을 “Driver={CUBRID Driver Unicode}”와 같이 입력한다. 유니코드는 9.3.0.0002 버전 이상에서만 지원된다.

## 참고

- 연결 문자열에서 세미콜론(;)은 구분자로 사용되므로, 연결 문자열에 암호(PWD)를 지정할 때 암호의 일부에 세미콜론을 사용할 수 없다.
- 스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.
- 자동 커밋 모드에서 SELECT 문 수행 이후 모든 결과 셋이 fetch되지 않으면 커밋이 되지 않는다. 따라서, 자동 커밋 모드라 하더라도 프로그램 내에서 결과 셋에 대한 fetch 도중 어떠한 오류가 발생한다면 반드시 커밋 또는 롤백을 수행하여 트랜잭션을 종료 처리하도록 한다.

## ASP 예제 프로그램

ASP 예제를 실행할 가상 디렉터리의 '기본 웹 사이트' 항목에서 마우스 오른쪽 버튼을 클릭한 뒤 [속성]을 클릭한다.



위의 그림에서 웹사이트 확인 아래 IP 주소를 (모두 할당되지 않음) 으로 선택하면 localhost로 인식한다. 특정한 IP 주소를 통해 예제를 확인하려면 해당 IP에 해당 디렉터리를 가상 디렉터리로 인식시키고 등록 정보에 IP 주소를 등록한다.

아래의 예제 코드를 cubrid.asp로 만들고 가상 디렉터리에 저장한다.

```
<HTML>
 <HEAD>
 <meta http-equiv="Content-Type" content="text/html; charset=EUC-
KR">
 <title>CUBRID Query Test Page</title>
```

```

</HEAD>

<BODY topmargin="0" leftmargin="0">

<table border="0" width="748" cellspacing="0" cellpadding="0">
 <tr>
 <td width="200"></td>
 <td width="287">
 <p align="center"><font size="3" face="Times New
Roman">CUBRIDQuery Test</
td>
 <td width="200"></td>
 </tr>
</table>

<form action="cubrid.asp" method="post" >
<table border="1" width="700" cellspacing="0" cellpadding="0"
height="45">
 <tr>
 <td width="113" valign="bottom" height="16" bgcolor="#DBD7BD"
bordercolorlight="#FFFFCC">SERVER IP</td>
 <td width="78" valign="bottom" height="16" bgcolor="#DBD7BD"
bordercolorlight="#FFFFCC">Broker PORT</td>
 <td width="148" valign="bottom" height="16" bgcolor="#DBD7BD"
bordercolorlight="#FFFFCC">DB NAME</td>
 <td width="113" valign="bottom" height="16" bgcolor="#DBD7BD"
bordercolorlight="#FFFFCC">DB USER</td>
 <td width="113" valign="bottom" height="16" bgcolor="#DBD7BD"
bordercolorlight="#FFFFCC">DB PASS</td>
 <td width="80" height="37" rowspan="4"
bordercolorlight="#FFFFCC" bgcolor="#F5F5ED">
 <p><input type="submit" value="실행하기" name="B1"
tabindex="7"></p></td>
 </tr>
 <tr>
 <td width="113" height="1" bordercolorlight="#FFFFCC"
bgcolor="#F5F5ED"><input type="text" name="server_ip"
size="20" tabindex="1" maxlength="15" value="<%=Request("server_ip")
%>"></td>
 <td width="78" height="1" bordercolorlight="#FFFFCC"
bgcolor="#F5F5ED"><input type="text" name="cas_port"
size="15" tabindex="2" maxlength="6" value="<%=Request("cas_port")
%>"></td>
 <td width="148" height="1" bordercolorlight="#FFFFCC"
bgcolor="#F5F5ED"><input type="text" name="db_name"
size="20" tabindex="3" maxlength="20" value="<%=Request("db_name")
%>"></td>
 <td width="113" height="1" bordercolorlight="#FFFFCC"
bgcolor="#F5F5ED"><input type="text" name="db_user"
size="15" tabindex="4" value="<%=Request("db_user")%>"></td>
 <td width="113" height="1" bordercolorlight="#FFFFCC"
bgcolor="#F5F5ED"><input type="password"

```

```

name="db_pass" size="15" tabindex="5" value="<%=Request("db_pass")
%>"></td>
</tr>
<tr>
<td width="573" colspan="5" valign="bottom" height="18"
bordercolorlight="#FFFFCC" bgcolor="#DBD7BD">QUERY</
font></td>
</tr>
<tr>
<td width="573" colspan="5" height="25"
bordercolorlight="#FFFFCC" bgcolor="#F5F5ED"><textarea rows="3"
name="query" cols="92" tabindex="6"><%=Request("query")%></
textarea></td>
</tr>
</table>
</form>
<hr>

</BODY>
</HTML>

```

```

<%
' DSN과 SQL문을 가져온다.
strIP = Request("server_ip")
strPort = Request("cas_port")
strUser = Request("db_user")
strPass = Request("db_pass")
strName = Request("db_name")
strQuery = Request("query")

if strIP = "" then
Response.Write "SERVER_IP를 입력하세요"
Response.End ' IP가 없으면 페이지 종료
end if
if strPort = "" then
Response.Write "Port 번호를 입력하세요"
Response.End ' Port가 없으면 페이지 종료
end if
if strUser = "" then
Response.Write "DB_USER를 입력하세요"
Response.End ' DB_User가 없으면 페이지 종료
end if
if strName = "" then
Response.Write "DB_NAME을 입력하세요"
Response.End ' DB_NAME이 없으면 페이지 종료
end if
if strQuery = "" then
Response.Write "확인하고자 하는 Query를 입력하세요"
Response.End ' Query가 없으면 페이지 종료
end if
' 연결 객체 생성
strDsn = "driver={CUBRID Driver};server=" & strIP & ";port=" &

```

```

strPort & ";uid=" & strUser & ";pwd=" & strPass & ";db_name=" &
strName & ";";
' DB연결
Set DBConn = Server.CreateObject("ADODB.Connection")
 DBConn.Open strDsn
' SQL 실행
Set rs = DBConn.Execute(strQuery)
' SQL문에 따라 메시지 보이기
if InStr(Ucase(strQuery),"INSERT")>0 then
 Response.Write "레코드가 추가되었습니다."
 Response.End
end if

if InStr(Ucase(strQuery),"DELETE")>0 then
 Response.Write "레코드가 삭제되었습니다."
 Response.End
end if

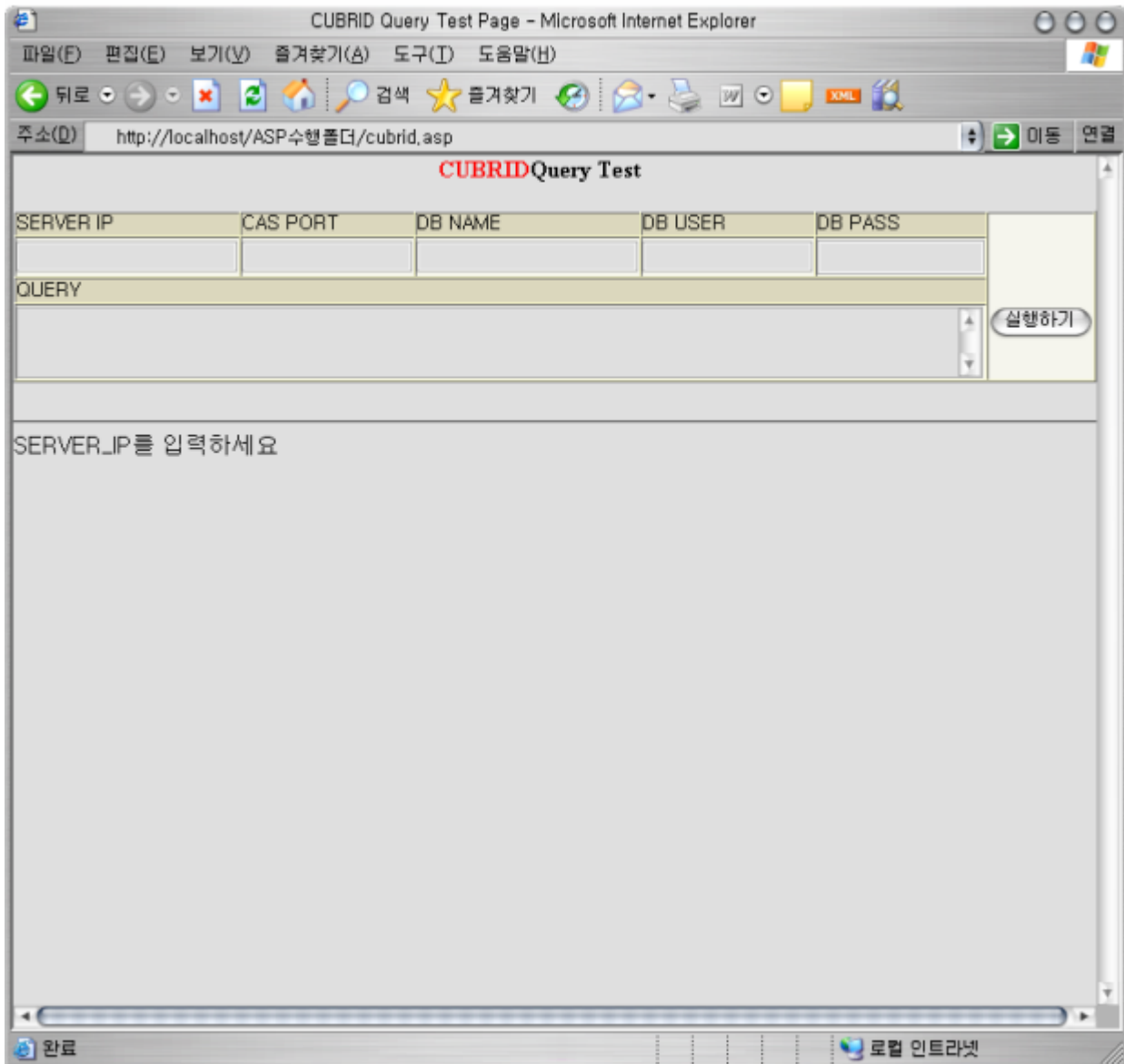
if InStr(Ucase(strQuery),"UPDATE")>0 then
 Response.Write "레코드가 수정되었습니다."
 Response.End
end if
%>
<table>
<%
' 필드 이름 보여주기
Response.Write "<tr bgColor=#f3f3f3>"
For index =0 to (rs.fields.count-1)
 Response.Write "<td>" & rs.fields(index).name & "</td>"
Next
Response.Write "</tr>"
' 필드 값 보여주기
Do While Not rs.EOF
 Response.Write "<tr bgColor=#f3f3f3>"
 For index =0 to (rs.fields.count-1)
 Response.Write "<td>" & rs(index) & "</td>"
 Next
 Response.Write "</tr>"

 rs.MoveNext
Loop
%>
<%
 set rs = nothing
%>
</table>

```

http://localhost/ASP수행폴더/cubrid.asp에 접속하면 수행 결과를 확인할 수 있다. 위의 ASP 예제 코드를 실행하면 다음과 같은 결과를 출력한다. 해당 항목에 알맞은 값을 넣고 Query 항목에 질의문을 입력하고 [실행하기]를 클릭하면 하단에 질의 문의 결과가 출력된다.





## ODBC API

ODBC API에 대한 자세한 내용은 ODBC API Reference 문서( <https://learn.microsoft.com/en-us/sql/odbc/reference/syntax/odbc-api-reference?view=sql-server-ver15> )를 참고한다. CUBRID ODBC에서 지원하는 함수 목록, ODBC Spec 버전 및 호환성은 다음과 같다.

| API            | Version Introduced | Standards Compliance | Support |
|----------------|--------------------|----------------------|---------|
| SQLAllocHandle | 3.0                | ISO 92               | YES     |
| SQLBindCol     | 1.0                | ISO 92               | YES     |

| API                 | Version Introduced | Standards Compliance | Support |
|---------------------|--------------------|----------------------|---------|
| SQLBindParameter    | 2.0                | ODBC                 | YES     |
| SQLBrowseConnect    | 1.0                | ODBC                 | NO      |
| SQLBulkOperations   | 3.0                | ODBC                 | YES     |
| SQLCancel           | 1.0                | ISO 92               | YES     |
| SQLCloseCursor      | 3.0                | ISO 92               | YES     |
| SQLColAttribute     | 3.0                | ISO 92               | YES     |
| SQLColumnPrivileges | 1.0                | ODBC                 | NO      |
| SQLColumns          | 1.0                | X/Open               | YES     |
| SQLConnect          | 1.0                | ISO 92               | YES     |
| SQLCopyDesc         | 3.0                | ISO 92               | YES     |
| SQLDescribeCol      | 1.0                | ISO 92               | YES     |
| SQLDescribeParam    | 1.0                | ODBC                 | NO      |
| SQLDisconnect       | 1.0                | ISO 92               | YES     |
| SQLDriverConnect    | 1.0                | ODBC                 | YES     |

| API               | Version Introduced | Standards Compliance | Support           |
|-------------------|--------------------|----------------------|-------------------|
| SQLEndTran        | 3.0                | ISO 92               | YES               |
| SQLExecDirect     | 1.0                | ISO 92               | YES               |
| SQLExecute        | 1.0                | ISO 92               | YES               |
| SQLFetch          | 1.0                | ISO 92               | YES               |
| SQLFetchScroll    | 3.0                | ISO 92               | YES               |
| SQLForeignKeys    | 1.0                | ODBC                 | YES(2008 R3.1 이상) |
| SQLFreeHandle     | 3.0                | ISO 92               | YES               |
| SQLFreeStmt       | 1.0                | ISO 92               | YES               |
| SQLGetConnectAttr | 3.0                | ISO 92               | YES               |
| SQLGetCursorName  | 1.0                | ISO 92               | YES               |
| SQLGetData        | 1.0                | ISO 92               | YES               |
| SQLGetDescField   | 3.0                | ISO 92               | YES               |
| SQLGetDescRec     | 3.0                | ISO 92               | YES               |
| SQLGetDiagField   | 3.0                | ISO 92               | YES               |

| API                 | Version Introduced | Standards Compliance | Support           |
|---------------------|--------------------|----------------------|-------------------|
| SQLGetDiagRec       | 3.0                | ISO 92               | YES               |
| SQLGetEnvAttr       | 3.0                | ISO 92               | YES               |
| SQLGetFunctions     | 1.0                | ISO 92               | YES               |
| SQLGetInfo          | 1.0                | ISO 92               | YES               |
| SQLGetStmtAttr      | 3.0                | ISO 92               | YES               |
| SQLGetTypeInfo      | 1.0                | ISO 92               | YES               |
| SQLMoreResults      | 1.0                | ODBC                 | YES               |
| SQLNativeSql        | 1.0                | ODBC                 | YES               |
| SQLNumParams        | 1.0                | ISO 92               | YES               |
| SQLNumResultCols    | 1.0                | ISO 92               | YES               |
| SQLParamData        | 1.0                | ISO 92               | YES               |
| SQLPrepare          | 1.0                | ISO 92               | YES               |
| SQLPrimaryKeys      | 1.0                | ODBC                 | YES(2008 R3.10이상) |
| SQLProcedureColumns | 1.0                | ODBC                 | YES(2008 R3.10이상) |

| API                | Version Introduced | Standards Compliance | Support           |
|--------------------|--------------------|----------------------|-------------------|
| SQLProcedures      | 1.0                | ODBC                 | YES(2008 R3.10이상) |
| SQLPutData         | 1.0                | ISO 92               | YES               |
| SQLRowCount        | 1.0                | ISO 92               | YES               |
| SQLSetConnectAttr  | 3.0                | ISO 92               | YES               |
| SQLSetCursorName   | 1.0                | ISO 92               | YES               |
| SQLSetDescField    | 3.0                | ISO 92               | YES               |
| SQLSetDescRec      | 3.0                | ISO 92               | YES               |
| SQLSetEnvAttr      | 3.0                | ISO 92               | NO                |
| SQLSetPos          | 1.0                | ODBC                 | YES               |
| SQLSetStmtAttr     | 3.0                | ISO 92               | YES               |
| SQLSpecialColumns  | 1.0                | X/Open               | YES               |
| SQLStatistics      | 1.0                | ISO 92               | YES               |
| SQLTablePrivileges | 1.0                | ODBC                 | YES(2008 R3.10이상) |
| SQLTables          | 1.0                | X/Open               | YES               |

ODBC 3.x에서 하위 호환성을 지원하지 않는 일부 함수에 대해서는 아래의 매핑 테이블을 참고하여 적합한 함수로 변환한다.

| ODBC 2.x 함수               | ODBC 3.x 함수       |
|---------------------------|-------------------|
| SQLAllocConnect           | SQLAllocHandle    |
| SQLAllocEnv               | SQLAllocHandle    |
| SQLAllocStmt              | SQLAllocHandle    |
| SQLBindParam              | SQLBindParameter  |
| SQLColAttributes          | SQLColAttribute   |
| SQLError                  | SQLGetDiagRec     |
| SQLFreeConnect            | SQLFreeHandle     |
| SQLFreeEnv                | SQLFreeHandle     |
| SQLFreeStmt with SQL_DROP | SQLFreeHandle     |
| SQLGetConnectOption       | SQLGetConnectAttr |
| SQLGetStmtOption          | SQLGetStmtAttr    |
| SQLParamOptions           | SQLSetStmtAttr    |
| SQLSetConnectOption       | SQLSetConnectAttr |
| SQLSetParam               | SQLBindParameter  |

| ODBC 2.x 함수        | ODBC 3.x 함수    |
|--------------------|----------------|
| SQLSetScrollOption | SQLSetStmtAttr |
| SQLSetStmtOption   | SQLSetStmtAttr |
| SQLTransact        | SQLEndTran     |

## ADO.NET 드라이버

ADO.NET은 .NET 개발자에게 데이터 액세스 서비스를 제공하는 클래스 집합이다. ADO.NET은 분산된 데이터를 분산된 데이터 공유 응용 프로그램을 개발할 때 사용할 수 있는 다양한 구성 요소를 제공한다. 또한 관계형, XML 및 응용 프로그램 데이터에 대한 액세스를 제공하는 .NET Framework의 핵심 부분이다. ADO.NET은 응용 프로그램, 도구, 언어 또는 웹 브라우저에서 사용되는 중간 계층 비즈니스 개체 및 프런트 엔드 데이터베이스 클라이언트 개발을 비롯하여 다양한 개발 요구 사항을 지원한다.

## ADO.NET 설치 및 설정

### 기본 환경

- Windows(Windows Vista 또는 Windows 7 권장)
- .NET 프레임워크 2.0 이상(4.0 이상 권장):
- Microsoft Visual Studio Express edition( <https://visualstudio.microsoft.com/> )

### 설치 및 설정

CUBRID를 사용하는 .NET 응용 프로그램을 개발하려면 CUBRID ADO.NET Data Provider(Cubrid.Data.dll)가 필요하다. CUBRID ADO.NET Data Provider를 설치하려면 다음 중 하나를 수행한다.

- CUBRID ADO.NET Data Provider Installer를 다음 주소에서 다운로드하여 실행한다.

<https://www.cubrid.org/downloads#adonet>

- 소스코드에서 직접 빌드한다. 소스코드는 GitHub에서 다운 받을 수 있습니다.

<https://github.com/CUBRID/cubrid-adonet>

CUBRID .NET Data Provider는 full-managed .NET 코드로 작성되어 CUBRID 라이브러리 파일에 의존하지 않는다. 따라서 CUBRID를 설치하거나 CUBRID 파일을 다운로드하지 않아도 CUBRID .NET Data Provider를 사용할 수 있다.

CUBRID ADO.NET Data Provider를 가장 간단하게 설치하는 방법은 CUBRID ADO.NET Data Provider Installer를 실행하는 것이다. 기본 설정(x86)으로 설치하면 **Program Files\CUBRID\CUBRID ADO.NET Data Provider 8.4.1** 디렉터리에 설치된다.

드라이버를 GAC( [https://en.wikipedia.org/wiki/Global\\_Assembly\\_Cache](https://en.wikipedia.org/wiki/Global_Assembly_Cache) )에 설치할 수도 있다. 드라이버를 GAC에 설치하는 가장 좋은 방법은 tlbimp( [https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-2.0/tt0cf3sx\(v=vs.80\)](https://learn.microsoft.com/en-us/previous-versions/dotnet/netframework-2.0/tt0cf3sx(v=vs.80)) )를 사용하는 것이다. 필요한 네임스페이스는 다음과 같이 import한다.

```
33 using System.Data.Common;
34 using System.Diagnostics;
35 using System.IO;
36 using System.Reflection;
37 using System.Text.RegularExpressions;
38 using CUBRID.Data.CUBRIDClient;
39
40 namespace CUBRID.ADO.NET.DataProvider.TestCases
41 {
42 /// <summary>
43 /// Implementation of test cases for CUBRID ADO.NET D
44 ...
```

## ADO.NET 프로그래밍

CUBRID ADO.NET API 문서는 [http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/ADO.NET/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/ADO.NET/)를 참고한다.

### 단순 질의/조회

CUBRID 데이터베이스의 테이블에서 값을 조회하는 간단한 코드를 살펴보자. 연결은 이미 생성되었다고 가정한다.

```
String sql = "select * from nation order by `code` asc";
using (CUBRIDCommand cmd = new CUBRIDCommand(sql, conn))
{
 using (DbDataReader reader = cmd.ExecuteReader())
 {
 reader.Read();
 //(read the values using: reader.Get...() methods)
 }
}
```



위와 같이 `DbDataReader` 객체를 생성한 후에는 `Get...()` 메서드를 사용하여 칼럼 데이터를 조회할 수 있다. CUBRID ADO.NET 드라이버는 다음과 같이 CUBRID의 모든 데이터 타입을 읽는 데 필요한 모든 메서드를 제공한다.

```
reader.GetString(3)
reader.GetDecimal(1)
```

`Get...()` 메서드의 파라미터로 0부터 시작하는 숫자를 입력하여 칼럼에서 조회할 칼럼 데이터의 인덱스 위치를 지정한다.

특정 CUBRID 데이터 타입의 데이터를 조회하려면 `DbDataReader` 인터페이스 대신 다음과 같이 `CUBRIDDataReader`를 사용해야 한다.

```
using (CUBRIDCommand cmd = new CUBRIDCommand("select * from t",
conn))
{
 CUBRIDDataReader reader = (CUBRIDDataReader)cmd.ExecuteReader();

 reader.Read();
 Debug.Assert(reader.GetDateTime(0) == new DateTime(2008, 10, 31,
10, 20, 30, 040));
 Debug.Assert(reader.GetDate(0) == "2008-10-31");
 Debug.Assert(reader.GetDate(0, "yy/MM/dd") == "08-10-31");
 Debug.Assert(reader.GetTime(0) == "10:20:30");
 Debug.Assert(reader.GetTime(0, "HH") == "10");
 Debug.Assert(reader.GetTimestamp(0) == "2008-10-31
10:20:30.040");
 Debug.Assert(reader.GetTimestamp(0, "yyyy HH") == "2008 10");
}
```

## batch 명령어

CUBRID ADO.NET Data Provider를 사용하면 하나의 batch에서 데이터 서비스에 하나 이상의 질의를 실행할 수 있다. batch에 대한 자세한 내용은 [https://learn.microsoft.com/en-us/previous-versions/dd744839\(v=vs.90\)](https://learn.microsoft.com/en-us/previous-versions/dd744839(v=vs.90))를 참고한다.

예를 들면 다음과 같은 코드를 작성할 수 있다.

```
string[] sql_arr = new string[3];
sql_arr0 = "insert into t values(1)";
sql_arr1 = "insert into t values(2)";
sql_arr2 = "insert into t values(3)";
conn.BatchExecute(sql_arr);
```

위 코드는 다음과 같이 작성할 수도 있다.

```
string[] sqls = newstring3;
sqls0 = "create table t(id int)";
sqls1 = "insert into t values(1)";
sqls2 = "insert into t values(2)";

conn.BatchExecuteNoQuery(sqls);
```

## 연결 문자열

.NET 응용 프로그램에서 CUBRID 연결을 생성하려면 다음과 같은 형식의 연결 문자열을 생성해야 한다.

```
ConnectionString = "server=<server address>;database=<database name>;port=<port number to use for connection to broker>;user=<user name>;password=<user password>;"
```

**port**를 제외한 모든 파라미터는 반드시 값을 입력해야 한다. **port** 값을 입력하지 않았을 때의 기본값은 **30000** 이다.

연결 옵션에 따른 연결 문자열의 예는 다음과 같다.

- 로컬 서버의 *demodb* 데이터베이스에 연결하는 연결 문자열은 다음과 같다.

```
ConnectionString =
"server=127.0.0.1;database=demodb;port=30000;user=public;password="
```

- 원격 서버의 *demodb* 데이터베이스에 **dba** 사용자로 연결하는 문자열은 다음과 같다.

```
ConnectionString =
"server=10.50.88.1;database=demodb;user=dba;password="
```

- 원격 서버의 *demodb* 데이터베이스에 **dba** 사용자, 비밀번호는 *secret* 으로 연결하는 문자열은 다음과 같다.

```
ConnectionString =
"server=10.50.99.1;database=demodb;port=30000;user=dba;password=secret"
```

연결 문자열은 CUBRIDConnectionStringBuilder 클래스를 사용하여 다음과 같이 생성할 수도 있다.

```
CUBRIDConnectionStringBuilder sb = new
CUBRIDConnectionStringBuilder("localhost","33000","demodb","public","");
using (CUBRIDConnection conn = new
```

```
CUBRIDConnection(sb.GetConnectionString()))
{
 conn.Open();
}
```

위 코드와 같은 동작을 수행하는 코드를 다음과 같이 작성할 수도 있다.

```
sb = new CUBRIDConnectionStringBuilder();
sb.User = "public" ;
sb.Database = "demodb";
sb.Port = "33000";
sb.Server = "localhost";
using (CUBRIDConnection conn = new
CUBRIDConnection(sb.GetConnectionString()))
{
 conn.Open();
}
```

#### 참고

스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.

## CUBRID 컬렉션

컬렉션은 CUBRID에서 사용하는 데이터 타입이다. 컬렉션 타입에 대한 자세한 내용은 [컬렉션 데이터 타입](#) 을 참고한다. 컬렉션 타입은 다른 데이터베이스에서 흔히 사용하지 않으므로, 이 타입을 사용하려면 다음과 같은 CUBRID 컬렉션 메서드를 사용해야 한다.

```
public void AddElementToSet(CUBRIDoid oid, String attributeName,
Object value)
public void DropElementInSet(CUBRIDoid oid, String attributeName,
Object value)
public void UpdateElementInSequence(CUBRIDoid oid, String
attributeName, int index, Object value)
public void InsertElementInSequence(CUBRIDoid oid, String
attributeName, int index, Object value)
public void DropElementInSequence(CUBRIDoid oid, String
attributeName, int index)
public int GetCollectionSize(CUBRIDoid oid, String attributeName)
```

다음은 컬렉션 타입에서 값을 읽는 코드의 예이다.

```
using (CUBRIDCommand cmd = new CUBRIDCommand("SELECT * FROM t",
conn))
```

```

{
 using (DbDataReader reader = cmd.ExecuteReader())
 {
 while (reader.Read())
 {
 object[] o = (object[])reader[0];
 for (int i = 0; i < SeqSize; i++)
 {
 //...
 }
 }
 }
}

```

다음은 컬렉션 타입을 갱신하는 코드의 예이다.

```

conn.InsertElementInSequence(oid, attributeName, 5, value);
SeqSize = conn.GetCollectionSize(oid, attributeName);
using (CUBRIDCommand cmd = new CUBRIDCommand("SELECT * FROM t",
conn))
{
 using (DbDataReader reader = cmd.ExecuteReader())
 {
 while (reader.Read())
 {
 int[] expected = { 7, 1, 2, 3, 7, 4, 5, 6 };
 object[] o = (object[])reader[0];
 }
 }
}
conn.DropElementInSequence(oid, attributeName, 5);
SeqSize = conn.GetCollectionSize(oid, attributeName);

```

## BLOB/CLOB 사용

CUBRID 2008 R4.0(8.4.0) 이상 버전에서는 GLO 데이터 타입을 더 이상 사용하지 않고 BLOB, CLOB와 같은 LOB 데이터 타입을 사용한다. 이 데이터 타입은 다른 데이터베이스에서 흔히 사용하지 않으므로, 이 타입을 사용하려면 CUBRID ADO.NET Data Provider가 제공하는 메서드를 사용해야 한다.

다음은 BLOB 데이터를 읽는 코드의 예이다.

```

CUBRIDCommand cmd = new CUBRIDCommand(sql, conn);
DbDataReader reader = cmd.ExecuteReader();

while (reader.Read())
{
 CUBRIDBlob bImage = (CUBRIDBlob)reader[0];
 byte[] bytes = new byte[(int)bImage.BlobLength];
}

```

```

 bytes = bImage.GetBytes(1, (int)bImage.BlobLength);
 //...
 }

```

다음은 CLOB 데이터를 갱신하는 코드의 예이다.

```

string sql = "UPDATE t SET c = ?";
CUBRIDCommand cmd = new CUBRIDCommand(sql, conn);

CUBRIDClob clob = new CUBRIDClob(conn);
str = conn.ConnectionString; //Use the ConnectionString for testing

clob.SetString(1, str);

CUBRIDParameter param = new CUBRIDParameter();

param.ParameterName = "?";
param.CUBRIDDataType = CUBRIDDataType.CCI_U_TYPE_CLOB;
param.Value = clob;

cmd.Parameters.Add(param);
cmd.ExecuteNonQuery();

```

## CUBRID 메타데이터 지원

CUBRID ADO.NET Data Provider는 데이터베이스 메타데이터를 지원하는 메서드를 제공한다. 메타데이터를 지원하는 메서드는 CUBRIDSchemaProvider 클래스에 구현되어 있다.

```

public DataTable GetDatabases(string[] filters)
public DataTable GetTables(string[] filters)
public DataTable GetViews(string[] filters)
public DataTable GetColumns(string[] filters)
public DataTable GetIndexes(string[] filters)
public DataTable GetIndexColumns(string[] filters)
public DataTable GetExportedKeys(string[] filters)
public DataTable GetCrossReferenceKeys(string[] filters)
public DataTable GetForeignKeys(string[] filters)
public DataTable GetUsers(string[] filters)
public DataTable GetProcedures(string[] filters)
public static DataTable GetDataTypes()
public static DataTable GetReservedWords()
public static String[] GetNumericFunctions()
public static String[] GetStringFunctions()
public DataTable GetSchema(string collection, string[] filters)

```

다음은 데이터베이스에서 테이블의 목록을 얻는 코드의 예이다.

```

CUBRIDSchemaProvider schema = new CUBRIDSchemaProvider(conn);
DataTable dt = schema.GetTables(new string[] { "%" });

Debug.Assert(dt.Columns.Count == 3);
Debug.Assert(dt.Rows.Count == 10);

Debug.Assert(dt.Rows[0].ToString() == "demodb");
Debug.Assert(dt.Rows[1].ToString() == "demodb");
Debug.Assert(dt.Rows[2].ToString() == "stadium");

Get the list of Foreign Keys in a table:

CUBRIDSchemaProvider schema = new CUBRIDSchemaProvider(conn);
DataTable dt = schema.GetForeignKeys(new string[] { "game" });

Debug.Assert(dt.Columns.Count == 9);
Debug.Assert(dt.Rows.Count == 2);

Debug.Assert(dt.Rows[0].ToString() == "athlete");
Debug.Assert(dt.Rows[1].ToString() == "code");
Debug.Assert(dt.Rows[2].ToString() == "game");
Debug.Assert(dt.Rows[3].ToString() == "athlete_code");
Debug.Assert(dt.Rows[4].ToString() == "1");
Debug.Assert(dt.Rows[5].ToString() == "1");
Debug.Assert(dt.Rows[6].ToString() == "1");
Debug.Assert(dt.Rows[7].ToString() == "fk_game_athlete_code");
Debug.Assert(dt.Rows[8].ToString() == "pk_athlete_code");

```

다음은 테이블의 인덱스 목록을 얻는 코드의 예이다.

```

CUBRIDSchemaProvider schema = new CUBRIDSchemaProvider(conn);
DataTable dt = schema.GetIndexes(new string[] { "game" });

Debug.Assert(dt.Columns.Count == 9);
Debug.Assert(dt.Rows.Count == 5);

Debug.Assert(dt.Rows32.ToString() ==
"pk_game_host_year_event_code_athlete_code"); //Index name
Debug.Assert(dt.Rows34.ToString() == "True"); //Is it a PK?

```

## DataTable 지원

**DataTable** 은 ADO.NET에서 가장 중심이 되는 객체로, CUBRID ADO.NET Data Provider는 다음과 같은 기능을 지원한다.

- **DataTable** 데이터 채우기
- 기본 제공 명령어: **INSERT, UPDATE, DELETE**

- 칼럼 메타데이터/속성
- `DataSet`, `DataView` 상호 연결

칼럼 속성을 얻는 코드의 예는 다음과 같다.

```
String sql = "select * from nation";
CUBRIDDataAdapter da = new CUBRIDDataAdapter();
da.SelectCommand = new CUBRIDCommand(sql, conn);
DataTable dt = new DataTable("nation");
da.FillSchema(dt, SchemaType.Source); //To retrieve all the column
properties you have to use the FillSchema() method

Debug.Assert(dt.Columns[0].ColumnName == "code");
Debug.Assert(dt.Columns[0].AllowDBNull == false);
Debug.Assert(dt.Columns[0].DefaultValue.ToString() == "");
Debug.Assert(dt.Columns[0].Unique == true);
Debug.Assert(dt.Columns[0].DataType == typeof(System.String));
Debug.Assert(dt.Columns[0].Ordinal == 0);
Debug.Assert(dt.Columns[0].Table == dt);
```

**INSERT** 문 지원 기능을 이용하여 테이블에 값을 삽입하는 코드의 예는 다음과 같다.

```
String sql = " select * from nation order by `code` asc";
using (CUBRIDDataAdapter da = new CUBRIDDataAdapter(sql, conn))
{
 using (CUBRIDDataAdapter daCmd = new CUBRIDDataAdapter(sql,
conn))
 {
 CUBRIDCommandBuilder cmdBuilder = new
CUBRIDCommandBuilder(daCmd);
 da.InsertCommand = cmdBuilder.GetInsertCommand();
 }

 DataTable dt = new DataTable("nation");
 da.Fill(dt);

 DataRow newRow = dt.NewRow();

 newRow["code"] = "ZZZ";
 newRow["name"] = "ABCDEF";
 newRow["capital"] = "MyXYZ";
 newRow["continent"] = "QWERTY";

 dt.Rows.Add(newRow);
 da.Update(dt);
}
```

## 트랜잭션

CUBRID ADO.NET Data Provider는 직접 SQL 트랜잭션(direct-SQL transaction)과 비슷한 방법으로 트랜잭션 지원을 구현한다. 다음은 트랜잭션을 사용하는 코드의 예이다.

```
conn.BeginTransaction();

string sql = "create table t(idx integer)";
using (CUBRIDCommand command = new CUBRIDCommand(sql, conn))
{
 command.ExecuteNonQuery();
}

conn.Rollback();

conn.BeginTransaction();

sql = "create table t(idx integer)";
using (CUBRIDCommand command = new CUBRIDCommand(sql, conn))
{
 command.ExecuteNonQuery();
}

conn.Commit();
```

## 파라미터 사용

CUBRID에서는 위치 기반 파라미터만 지원하며 명명된 파라미터는 지원하지 않으므로, CUBRID ADO.NET Data Provider는 위치 기반 파라미터 지원을 구현한다. 파라미터 이름은 자유롭게 사용할 수 있으며 파라미터 이름 앞에는 물음표 기호를 붙여야 한다. 파라미터를 선언하고 초기화할 때 반드시 파라미터의 순서를 지켜야 한다.

다음은 파라미터를 사용하여 SQL문을 실행하는 코드의 예이다. 다음 코드에서 중요한 것은 **Add ()** 메서드가 호출되는 순서이다.

```
using (CUBRIDCommand cmd = new CUBRIDCommand("insert into t
values(?, ?)", conn))
{
 CUBRIDParameter p1 = new CUBRIDParameter("?p1",
CUBRIDDataType.CCI_U_TYPE_INT);
 p1.Value = 1;
 cmd.Parameters.Add(p1);

 CUBRIDParameter p2 = new CUBRIDParameter("?p2",
CUBRIDDataType.CCI_U_TYPE_STRING);
 p2.Value = "abc";
 cmd.Parameters.Add(p2);
```



```
cmd.ExecuteNonQuery();
}
```

## 오류 코드 및 메시지

다음은 CUBRID ADO.NET Data Provider를 사용하면서 오류가 발생할 때 나타나는 오류이다.

| 오류 코드 번호 | 오류 코드              | 오류 메시지                                                                |
|----------|--------------------|-----------------------------------------------------------------------|
| 0        | ER_NO_ERROR        | "No Error"                                                            |
| 1        | ER_NOT_OBJECT      | "Index's Column is Not Object"                                        |
| 2        | ER_DBMS            | "Server error"                                                        |
| 3        | ER_COMMUNICATION   | "Cannot communicate with the broker"                                  |
| 4        | ER_NO_MORE_DATA    | "Invalid dataReader position"                                         |
| 5        | ER_TYPE_CONVERSION | "DataType conversion error"                                           |
| 6        | ER_BIND_INDEX      | "Missing or invalid position of the bind variable provided"           |
| 7        | ER_NOT_BIND        | "Attempt to execute the query when not all the parameters are binded" |
| 8        | ER_WAS_NULL        | "Internal Error: NULL value"                                          |
| 9        | ER_COLUMN_INDEX    |                                                                       |

| 오류 코드 번호 | 오류 코드                | 오류 메시지                                                          |
|----------|----------------------|-----------------------------------------------------------------|
|          |                      | “Column index is out of range”                                  |
| 10       | ER_TRUNCATE          | “Data is truncated because receive buffer is too small”         |
| 11       | ER_SCHEMA_TYPE       | “Internal error: Illegal schema paramCUBRIDDataType”            |
| 12       | ER_FILE              | “File access failed”                                            |
| 13       | ER_CONNECTION        | “Cannot connect to a broker”                                    |
| 14       | ER_ISO_TYPE          | “Unknown transaction isolation level”                           |
| 15       | ER_ILLEGAL_REQUEST   | “Internal error: The requested information is not available”    |
| 16       | ER_INVALID_ARGUMENT  | “The argument is invalid”                                       |
| 17       | ER_IS_CLOSED         | “Connection or Statement might be closed”                       |
| 18       | ER_ILLEGAL_FLAG      | “Internal error: Invalid argument”                              |
| 19       | ER_ILLEGAL_DATA_SIZE | “Cannot communicate with the broker or received invalid packet” |

| 오류 코드 번호 | 오류 코드                  | 오류 메시지                                  |
|----------|------------------------|-----------------------------------------|
| 20       | ER_NO_MORE_RESULT      | "No More Result"                        |
| 21       | ER_OID_IS_NOT_INCLUDED | "This ResultSet do not include the OID" |
| 22       | ER_CMD_IS_NOT_INSERT   | "Command is not insert"                 |
| 23       | ER_UNKNOWN             | "Error"                                 |

## ADO.NET API

[http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/ADO.NET/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/ADO.NET/)을 참고한다.

## Perl 드라이버

**DBD::cubrid** 는 Perl5 DBI(Database Interface)에서 CUBRID 데이터베이스에 접근하기 위해 사용하는 CUBRID Perl 드라이버로, Perl5 DBI의 모든 API를 지원한다.

CUBRID Perl 드라이버는 CCI API를 기반으로 작성되었으므로, CCI API 및 CCI에 적용되는 **CCI\_DEFAULT\_AUTOCOMMIT** 과 같은 설정 파라미터에 영향을 받는다.

### 참고

- 스레드 기반 프로그램에서 데이터베이스 연결은 각 스레드마다 독립적으로 사용해야 한다.
- 자동 커밋 모드에서 SELECT 문 수행 이후 모든 결과 셋이 fetch되지 않으면 커밋이 되지 않는다. 따라서, 자동 커밋 모드라 하더라도 프로그램 내에서 결과 셋에 대한 fetch 도중 어떠한 오류가 발생한다면 반드시 커밋 또는 롤백을 수행하여 트랜잭션을 종료 처리하도록 한다.

## Perl 설치 및 설정

### 기본 환경

- Perl: 시스템에 적합한 버전의 Perl을 사용하는 것을 권장한다. 모든 Linux와 FreeBSD에는 Perl이 포함되어 있으며, Windows에서는 ActivePerl을 권장한다. Active Perl에 대한 자세한 내용은 <https://www.activestate.com/platform/supported-languages/legacy-languages>을 참고한다.
- CUBRID: Perl 드라이버를 빌드하기 위해 CCI 드라이버가 필요하며, 이를 위해 CUBRID Engine을 설치해야 한다. <https://www.cubrid.org/downloads>에서 필요한 버전의 CUBRID Engine을 다운로드하여 설치하면 된다. CUBRID Engine의 설치후 CCI 드라이버는 CUBRID 기본 경로 \$CUBRID/cci에서 확인할 수 있다.
- DBI: <https://metacpan.org/dist/DBI/>
- C 컴파일러: 대부분의 경우에는 **DBD::cubrid** 바이너리(<https://www.cubrid.org/downloads#perl>)를 사용할 수 있으나, 만약 소스코드에서 드라이버를 빌드하려면 C 컴파일러가 필요하다. C 컴파일러를 사용하려면 Perl과 CUBRID를 컴파일한 컴파일러와 같은 컴파일러를 사용해야 한다. 그렇지 않으면 C 런타임 라이브러리 차이 때문에 문제가 발생할 수 있다.
- Linux에서는 CCI Driver를 빌드하기 위해 GNU Developer Toolset 8 또는 그 이상이 필요하다.

### CPAN을 이용한 설치

다음과 같이 **CPAN** (Comprehensive Perl Archive Network)을 사용하면 자동으로 소스코드에서 드라이버를 설치할 수 있다.

```
cpan
install DBD::cubrid
```

만약 **CPAN** 모듈을 처음으로 사용한다면 기본 설정에 따르는 것을 권장한다.

최신 버전의 Perl을 사용하지 않는다면 위 명령어 대신 다음 명령어를 사용해야 할 수도 있다.

```
perl -MCPAN -e shell
install DBD::cubrid
```

### 수동 설치

**CPAN** 을 이용해서 설치할 수 없다면 **DBD::cubrid** 소스코드를 다운로드해야 한다. 최신 버전은 아래 주소에서 다운로드할 수 있다.

<https://www.cubrid.org/downloads#perl>

파일 이름은 일반적으로 **DBD-cubrid-X.X.X.tar.gz** 와 같은 형식이다. 압축을 해제한 후 **DBD-cubrid-X.X.X** 디렉터리로 이동하여 다음 명령어를 실행한다.

```
Perl Makefile.PL
make
make test
```

Windows에서는 **make** 대신 **nmake** 또는 **dmake** 를 사용해야 할 수도 있다. 테스트 결과가 성공적이면 다음 명령어를 실행하여 드라이버를 빌드한다.

```
make install
```

## Perl API

CUBRID Perl 드라이버는 현재에는 기본 기능만을 제공하고 있다. LOB 타입이나 칼럼 정보 확인 등의 기능은 현재 지원하지 않는다.

CUBRID Perl API는 [http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/Perl/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/Perl/) 을 참고한다.

## Python 드라이버

**CUBRIDdb** 는 Python Database API 2.0을 준수하며 CUBRID 데이터베이스를 지원하는 Python 확장 패키지이다. CUBRID Python API는 Python Database API가 제공하는 기본 기능 외에도, CUBRID 데이터베이스 엔진에서 제공하는 기능을 **\_cubrid** 모듈에서 제공한다.

CUBRID Python 드라이버는 CCI API를 기반으로 작성되었으므로, CCI API 및 CCI에 적용되는 **CCI\_DEFAULT\_AUTOCOMMIT** 과 같은 설정 파라미터에 영향을 받는다.

## Python 설치 및 설정

### Linux/Unix

Linux, Unix 및 유사 운영체제에서는 다음과 같은 세 가지 방법으로 CUBRID Python 드라이버를 설치할 수 있다.

#### 기본 환경

- 운영체제: Linux: 32 비트/64비트 또는 유사 Unix 운영체제
- Python: 2.4 이상(<https://www.python.org/downloads/>)

#### 소스코드로 설치(Linux)

소스코드를 컴파일하여 CUBRID Python 드라이버를 설치하려면 Python Development Package가 필요하다.

1. CCI 드라이버 빌드하기 위해 GNU Developer Toolset 8 또는 그 이상이 필요하다.
2. 소스 코드를 <https://www.cubrid.org/downloads#python>에서 다운로드한다.
3. 다음 명령어를 실행하여 원하는 위치에 다운로드한 파일의 압축을 해제한다.

```
tar xvfz cubrid-python-11.2-latest.tar.gz
```

4. 압축을 해제한 디렉터리로 이동한다.

```
cd RB-11.0.0
```

5. 드라이버를 빌드한다. 이 단계와 다음 단계는 루트 사용자 계정으로 실행해야 한다.

```
python setup.py build
```

6. 빌드한 드라이버를 설치한다.

```
python setup.py install
```

### Easy Install을 이용한 설치(Linux)

Easy Install은 자동으로 Python 패키지를 다운로드/빌드/설치/관리할 수 있는 Python 모듈로, `setuptools`에 포함되어 있다. Easy Install을 사용하면 패키지 인덱스뿐만 아니라 다른 웹 사이트에도 HTTP로 연결하여 패키지를 설치할 수 있다. Perl의 CPAN이나 PHP의 PEAR와 유사하다. Easy Install에 대한 더 자세한 설명은 [https://setuptools.pypa.io/en/latest/deprecated/easy\\_install.html](https://setuptools.pypa.io/en/latest/deprecated/easy_install.html)을 참고한다.

Easy Install을 이용하여 CUBRID Python 드라이버를 설치하려면 다음 명령어를 입력한다.

```
easy_install CUBRID-Python
```

### Windows

Windows에 CUBRID Python 드라이버를 설치하려면 다음과 같이 CUBRID Python 드라이버를 다운로드하여 설치한다.

- 다음 주소에서 운영체제와 Python의 버전에 맞는 드라이버를 다운로드한다.

<https://www.cubrid.org/downloads#python>

- 다운로드한 파일의 압축을 해제하여 Python이 설치된 경로의 **Lib** 폴더( **C:\Program Files\Python\Lib** ) 안에 복사한다.

## Python 프로그래밍

CUBRIDdb 패키지는 Python Database API 2.0에 따라 다음과 같은 상수를 갖는다.

| 이름           | 값     |
|--------------|-------|
| threadsafety | 2     |
| apilevel     | 2.0   |
| paramstyle   | qmark |

## Python 예제 프로그램

여기에서는 Python으로 CUBRID 데이터베이스에 대한 작업을 수행하는 예제 프로그램을 작성한다. 예제로 다음과 같은 테이블을 생성한다.

```
csql -u dba -c "CREATE TABLE posts(id integer, title varchar(255),
body string, last_updated timestamp);" demodb
csql -u dba -c "grant ALL PRIVILEGES on posts to public;" demodb
```

### Python에서 demodb에 연결

1. 새 Python 콘솔을 열어 다음과 같이 Python에 CUBRID Python 드라이버를 import한다.

```
import CUBRIDdb
```

2. localhost에 위치한 demodb 데이터베이스에 연결을 생성한다.

```
conn = CUBRIDdb.connect('CUBRID:localhost:33000:demodb::', 'dba',
'')
```

demodb 데이터베이스는 비밀번호가 필요하지 않으므로 비밀번호를 입력하지 않았다. 그러나 실제 데이터베이스에 연결할 때에는 비밀번호가 필요하다면 비밀번호를 입력해야 한다. `connect()` 함수의 구문은 다음과 같다.

```
connect (url[,user[password]])
```

연결하려는 데이터베이스가 시작되지 않았다면 다음과 같은 오류가 발생한다.

```
Traceback (most recent call last):
 File "tutorial.py", line 3, in <module>
 conn = CUBRIDdb.connect('CUBRID:localhost:30000:demodb:dba::')
 File "/usr/local/lib/python3.5/site-packages/CUBRIDdb/
__init__.py", line 61, in Connect
 return Connection(*args, **kwargs)
 File "/usr/local/lib/python3.5/site-packages/CUBRIDdb/
connections.py", line 22, in __init__
 self.connection = _cubrid.connect(*args, **kwargs2)
_cubrid.OperationalError: (-677, "ERROR: DBMS, -677, Failed to
connect to database server, 'demodb', on the following host(s):
localhost:localhost[CAS INFO-127.0.0.1:30000,0,0].")
```

자격이 잘못되었다면 다음과 같은 오류가 발생한다.

```
Traceback (most recent call last):
 File "tutorial.py", line 3, in <module>
 con = CUBRIDdb.connect('CUBRID:localhost:
33000:demodb::','a','b')
 File "/usr/local/lib/python3.5/site-packages/CUBRIDdb/
__init__.py", line 61, in Connect
 return Connection(*args, **kwargs)
 File "/usr/local/lib/python3.5/site-packages/CUBRIDdb/
connections.py", line 22, in __init__
 self.connection = _cubrid.connect(*args, **kwargs2)
_cubrid.DatabaseError: (-165, 'ERROR: DBMS, -165, User "a" is
invalid.[CAS INFO-127.0.0.1:33000,0,0].')
```

## INSERT 문 실행

테이블이 비어있으므로 데이터를 입력한다. 먼저 커서를 얻은 후에 **INSERT** 문을 실행해야 한다.

```
cur = conn.cursor()
cur.execute("INSERT INTO posts (id, title, body, last_updated)
VALUES (1, 'Title 1', 'Test body #1', CURRENT_TIMESTAMP)")
conn.commit()
```

CUBRID Python 드라이버에서는 기본적으로 자동 커밋 모드가 비활성화되어 있다. 따라서 SQL문을 실행한 후에는 수동으로 `commit()` 함수를 사용하여 커밋을 수행해야 한다. 이 함수는 `cur.execute("COMMIT")` 와 같은 동작을 수행한다. 반대로 현재 트랜잭션을 중단하고 롤백하려면 `rollback()` 함수를 사용한다.



데이터를 입력하는 다른 방법으로 prepared statement를 사용할 수도 있다. 다음과 같이 파라미터를 포함하는 튜플을 정의한 후 `execute()` 함수에 전달하여 안전하게 데이터베이스에 데이터를 입력할 수 있다.

```
args = (2, 'Title 2', 'Test body #2')
cur.execute("INSERT INTO posts (id, title, body, last_updated)
VALUES (?, ?, ?, CURRENT_TIMESTAMP)", args)
```

여기까지 작성한 코드는 다음과 같다.

```
import CUBRIDdb
conn = CUBRIDdb.connect('CUBRID:localhost:33000:demodb::', 'dba',
'')
cur = conn.cursor()

Plain insert statement
cur.execute("INSERT INTO posts (id, title, body, last_updated)
VALUES (1, 'Title 1', 'Test body #1', CURRENT_TIMESTAMP)")

Parameterized insert statement
args = (2, 'Title 2', 'Test body #2')
cur.execute("INSERT INTO posts (id, title, body, last_updated)
VALUES (?, ?, ?, CURRENT_TIMESTAMP)", args)

conn.commit()
```

### 전체 레코드를 한 번에 조회

`fetchall()` 함수를 사용하면 전체 레코드를 한 번에 조회할 수 있다.

```
cur.execute("SELECT * FROM posts ORDER BY last_updated")
rows = cur.fetchall()
for row in rows:
 print (row)
```

위 코드는 다음과 같은 내용을 출력한다.

```
[1, 'Title 1', 'Test body #1', '2011-4-7 14:34:46']
[2, 'Title 2', 'Test body #2', '2010-4-7 14:34:46']
```

### 하나의 레코드를 조회

데이터의 양이 많다면 전체 결과를 메모리로 가져오는 대신 다음과 같이 `fetchone()` 함수를 사용하여 레코드를 한 번에 하나씩 조회할 수 있다.

```
cur.execute("SELECT * FROM posts")
row = cur.fetchone()
while row:
 print (row)
 row = cur.fetchone()
```

### 레코드 개수를 지정하여 조회

다음과 같이 `fetchmany()` 함수를 사용하면 조회할 레코드의 개수를 지정할 수 있다.

```
cur.execute("SELECT * FROM posts")
rows = cur.fetchmany(3)
for row in rows:
 print (row)
```

### 반환된 데이터의 메타데이터에 접근

조회한 레코드의 칼럼 속성에 대한 정보가 필요하면 커서의 `description` 메서드를 사용한다.

```
for description in cur.description:
 print (description)
```

위 코드는 다음과 같은 내용을 출력한다.

```
('id', 8, 0, 0, 0, 0, 0)
('title', 2, 0, 0, 255, 0, 0)
('body', 2, 0, 0, 1073741823, 0, 0)
('last_updated', 15, 0, 0, 0, 0, 0)
```

각 튜플은 다음과 같은 정보를 포함한다.

```
(column_name, data_type, display_size, internal_size, precision,
scale, nullable)
```

데이터 타입을 나타내는 숫자에 대한 자세한 내용은 [https://pythonhosted.org/CUBRID-Python/toc-CUBRIDdb.FIELD\\_TYPE-module.html](https://pythonhosted.org/CUBRID-Python/toc-CUBRIDdb.FIELD_TYPE-module.html) 을 참고한다.

### 자원 해제

데이터베이스 연결이나 커서를 사용하는 모든 작업을 마친 후에는 객체의 `close()` 함수를 호출하여 자원을 해제해야 한다.

```
cur.close()
conn.close()
```

## Python API

Python Database API는 connect() 모듈 클래스와 Connection 객체, Cursor 객체, 그리고 그 밖의 보조적인 함수들로 이루어진다. 이에 대한 자세한 내용은 <https://peps.python.org/pep-0249/> 를 참고한다.

CUBRID Python API에 대한 자세한 내용은 [http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/Python/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/Python/) 을 참고한다.

## Ruby 드라이버

CUBRID Ruby 드라이버는 Ruby로 작성한 응용 프로그램에서 CUBRID 데이터베이스를 사용할 수 있게 하는 드라이버로, RubyGem 패키지 형태로 제공된다.

CUBRID Ruby 드라이버는 CCI API를 기반으로 작성되었으므로, CCI API 및 CCI에 적용되는 **CCI\_DEFAULT\_AUTOCOMMIT** 과 같은 설정 파라미터에 영향을 받는다.

## Ruby 설치 및 설정

### 기본 환경

- Ruby 1.8.7 이상
- CUBRID gem
- ActiveRecord gem

### Linux

**gem** 을 사용하여 CUBRID Connector를 설치할 수 있다. 다음과 같이 **sudo** 명령어에 **-E** 옵션을 사용하여 **sudo** 명령어가 CUBRID 데이터베이스 설치 경로 환경 변수를 변경하지 않도록 해야 한다.

```
sudo -E gem install cubrid
```

### Windows

다음 명령어를 실행하여 최신 버전의 CUBRID Ruby 드라이버를 설치한다.

```
gem install cubrid
```

## Ruby 예제 프로그램

여기에서는 Ruby로 CUBRID 데이터베이스에 대한 작업을 수행하는 예제 프로그램을 작성한다. 예제로 다음과 같이 테이블을 생성한다.

```
CREATE TABLE countries(
 id INTEGER AUTO_INCREMENT,
 code CHARACTER VARYING(3) NOT NULL UNIQUE,
 name CHARACTER VARYING(40) NOT NULL UNIQUE,
 record_date DATETIME DEFAULT SYSDATETIME NOT NULL,
 CONSTRAINT pk_countries_id PRIMARY KEY(id)
);

CREATE TABLE cities(
 id INTEGER AUTO_INCREMENT NOT NULL UNIQUE,
 name CHARACTER VARYING(40) NOT NULL,
 country_id INTEGER NOT NULL,
 record_date DATETIME DEFAULT SYSDATETIME NOT NULL,
 FOREIGN KEY (country_id) REFERENCES countries(id) ON DELETE
 RESTRICT ON UPDATE RESTRICT,
 CONSTRAINT pk_cities_id PRIMARY KEY(id)
);
```

### 라이브러리 로드

예제 프로그램으로 *tutorial.rb* 라는 파일을 생성하고 다음과 같은 기본 설정을 작성한다.

```
require 'rubygems'
require 'active_record'
require 'pp'
```

### 데이터베이스 연결

다음과 같이 파라미터를 정의하여 데이터베이스 연결을 생성한다.

```
ActiveRecord::Base.establish_connection(
 :adapter => "cubrid",
 :host => "localhost",
 :database => "demodb" ,
 :user => "dba"
)
```

### 데이터베이스에 객체 삽입

테이블을 조작하기 전에 테이블을 ActiveRecord의 클래스와 매핑해야 한다.

```

class Country < ActiveRecord::Base
end

class City < ActiveRecord::Base
end

Country.create(:code => 'ROU', :name => 'Romania')
Country.create(:code => 'HUN', :name => 'Hungary')
Country.create(:code => 'DEU', :name => 'Germany')
Country.create(:code => 'FRA', :name => 'France')
Country.create(:code => 'ITA', :name => 'Italy', :record_date =>
Time.now)
Country.create(:code => 'SPN', :name => 'Spain')

```

### 데이터베이스에서 레코드 조회

다음과 같이 데이터베이스에서 레코드를 조회한다.

```

romania = Country.find(1)
pp(romania)

romania = Country.where(:code => 'ROU')
pp(romania)

Country.find_each do |country|
 pp(country)
end

```

### 데이터베이스 레코드 갱신

여기에서는 다음과 같이 *Spain* 의 *code* 를 'SPN' 에서 'ESP' 로 변경한다.

```

Country.transaction do
 spain = Country.where(:code => 'SPN')[0]
 spain.code = 'ESP'
 spain.save
end

```

### 데이터베이스 레코드 삭제

데이터베이스의 레코드를 삭제하는 코드는 다음과 같다.

```

Country.transaction do
 spain = Country.where(:code => 'ESP')[0]
 spain.destroy
end

```

## 연관(association)을 이용한 작업

국가에 도시를 추가하는 방법 중 하나는 *Country* 를 조회하여 *Country* 의 *code* 를 새로운 *City* 객체에 할당하는 것이다.

```
romania = Country.where(:code => 'ROU')[0]
City.create(:country_id => romania.id, :name => 'Bucharest');
```

더 좋은 방법은 다음과 같이 ActiveRecord에 관계를 알리고 이를 *Country* 클래스에 선언하는 것이다.

```
class Country < ActiveRecord::Base
 has_many :cities, :dependent => :destroy
end

class City < ActiveRecord::Base
end
```

위 코드에 따라 한 국가는 여러 개의 도시를 가질 수 있다. 이제 다음과 같이 간단하게 국가에 새 도시를 추가할 수 있다. 이 방법을 사용하면 도시에 접근할 때 참조되는 국가의 모든 도시들을 얻을 수 있으므로 유용하게 사용할 수 있다.

```
italy = Country.where(:code => 'ITA')[0]
italy.cities.create(:name => 'Milano');
italy.cities.create(:name => 'Napoli');

pp (romania.cities)
pp (italy.cities)
```

또한 다음과 같은 코드로 국가를 삭제하면 그 국가의 모든 도시가 삭제된다.

```
romania.destroy
```

**ActiveRecord** 는 일대일이나 다대다(many-to-many)와 같은 관계도 지원한다.

## 메타데이터 관리

ActiveRecord를 사용하면 코드를 수정하지 않아도 다른 데이터베이스를 사용할 수 있다.

## 데이터베이스 구조 정의

**ActiveRecord::Schema.define** 을 사용하여 새 테이블을 정의할 수 있다. 예를 들면 다음과 같이 일대다(one-to-many)로 대응되는 책에 대한 테이블(*books*)과 저자에 대한 테이블(*authors*)을 생성할 수 있다.

```
ActiveRecord::Schema.define do
 create_table :books do |table|
 table.column :title, :string, :null => false
 table.column :price, :float, :null => false
 table.column :author_id, :integer, :null => false
 end

 create_table :authors do |table|
 table.column :name, :string, :null => false
 table.column :address, :string
 table.column :phone, :string
 end

 add_index :books, :author_id
end
```

CUBRID에서 지원하는 칼럼 타입은 **:string, :text, :integer, :float, :decimal, :datetime, :timestamp, :time, :boolean, :bit, :smallint, :bigint, :char** 이다. 현재 **:binary** 는 지원하지 않는다.

### 테이블 칼럼 관리

**ActiveRecord::Migration** 의 기능을 사용하여 테이블의 칼럼을 추가하거나 업데이트, 삭제할 수 있다.

```
ActiveRecord::Schema.define do
 create_table :todos do |table|
 table.column :title, :string
 table.column :description, :string
 end

 change_column :todos, :description, :string, :null => false
 add_column :todos, :created, :datetime, :default => Time.now
 rename_column :todos, :created, :record_date
 remove_column :todos, :record_date

end
```

### 데이터베이스 스키마 덤프

**ActiveRecord::SchemaDumper.dump** 를 사용하여 현재 사용 중인 스키마의 정보를 덤프할 수 있다. 덤프된 스키마 정보는 플랫폼과 상관없이 사용할 수 있는 형식으로 저장되며 Ruby ActiveRecord에서도 사용할 수 있다.

단, **:bigint, :bit** 등과 같이 특정 데이터베이스에서 사용되는 커스텀 칼럼 타입을 사용한다면 제대로 동작하지 않을 수 있다.

### 서버 용량 정보 획득

현재 연결에서 다음과 같이 데이터베이스 정보를 획득할 수 있다.

```
puts "Maximum column length : " +
ActiveRecord::Base.connection.column_name_length.to_s
puts "SQL statement maximum length : " +
ActiveRecord::Base.connection.sql_query_length.to_s
puts "Quoting : ''test'' : " +
ActiveRecord::Base.connection.quote("''test''")
```

## 데이터베이스 생성

CUBRID에서는 데이터베이스 생성을 **cubrid create** 유틸리티 명령어로만 처리하기 때문에, 프로그램 내에서는 데이터베이스를 생성할 수 없다.

```
ActiveRecord::Schema.define do
 create_database('not_supported')
end
```

## Ruby API

[http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/Ruby/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/Ruby/)를 참고한다.

## Node.js 드라이버

CUBRID Node.js 드라이버는 100% 순수 자바스크립트로 개발되었으며, 특정 플랫폼 상에서의 컴파일 필요로 하지 않는다.

Node.js는 **크롬의 자바스크립트 런타임** 상에서 빌드된 플랫폼이다.

Node.js는 다음의 특징을 가지고 있다.

- 이벤트에 의해 제어되는(event-driven) 서버 측 자바스크립트이다.
- 동시에 여러 종류의 많은 I/O를 다루는데 적합하다.
- 블로킹 없는(non-blocking) I/O 모델이라 경량이며 효율적이다.

보다 자세한 사항은 <https://nodejs.org>를 참고한다.

큐브리드 Node.js 드라이버를 다운로드하거나 최신의 정보를 찾고자 할 때는 아래의 사이트를 참고한다.

- 소스코드 메인 저장소: <https://github.com/CUBRID/node-cubrid>



## Node.js 설치

### 기본 환경

- CUBRID 8.4.1 Patch 2 이상
- [Node.js](#)

### 설치

CUBRID Node.js 드라이버는 먼저 <https://nodejs.org/en/download/> 에서 node.js를 설치한 후, npm(Node Packaged Modules) install 명령을 사용하여 설치할 수 있다.

```
npm install node-cubrid
```

언인스톨하려면 다음 명령을 수행한다.

```
npm uninstall node-cubrid
```

## CUBRID Node.js API

[http://ftp.cubrid.org/CUBRID\\_Docs/Drivers/Node.js/](http://ftp.cubrid.org/CUBRID_Docs/Drivers/Node.js/) 를 참고한다.

## 릴리스 노트

### 11.4 릴리스 노트

#### 목차

|                    |          |
|--------------------|----------|
| <b>11.4 릴리스 노트</b> | <b>7</b> |
| ● 릴리스 노트 정보        | 2521     |
| ● 릴리스 개요           | 10       |
| ● 드라이버 호환성         | 12       |

|                                                      |      |
|------------------------------------------------------|------|
| ● 11.4 변경사항                                          | 2522 |
| ● 기능 개선                                              | 2522 |
| ● PL/CSQL                                            | 1438 |
| ● SQL                                                | 864  |
| ● HASH JOIN 지원                                       | 2523 |
| ● SERIAL 소유자 변경 SQL 구문 지원                            | 2523 |
| ● 생성된 그룹에 구성원을 추가/삭제할 수 있는 SQL 구문 지원                 | 21   |
| ● 쿼리 캐시 확장 지원                                        | 64   |
| ● 조인 순서를 명시적으로 제어 가능한 LEADING 힌트 지원                  | 2525 |
| ● 유틸리티                                               | 159  |
| ● <code>cubrid plandump</code> 의 특정 plan만 삭제하는 옵션 지원 | 2525 |
| ● JNI 사용한 JavaSP를 위한 별도의 loadjava 옵션 지원              | 2525 |
| ● 메모리 모니터링 유틸리티 지원                                   | 161  |
| ● <code>diagdb -d 9</code> 에 하나의 테이블만 덤프할 수 있는 옵션 지원 | 2526 |
| ● Broker,CAS,CMS                                     | 2526 |
| ● 브로커별 통신 버퍼 크기 조절을 위한 파라미터 지원                       | 163  |
| ● ACL 정책 정의를 위한 파라미터 지원                              | 2526 |
| ● 기타                                                 | 167  |
| ● Non-HA DB서버가 비정상 종료 시 자동 재 구동 기능 지원                | 2527 |
| ● 백업본을 신규 DB로 복구하는 스크립트 지원                           | 58   |

|                                                                                                 |      |
|-------------------------------------------------------------------------------------------------|------|
| ● 스펙변경                                                                                          | 2527 |
| ● SQL                                                                                           | 174  |
| ● CHAR 타입의 최대 문자 개수를 2048로 제한( 변경 전 최대 : 268,435,456 )                                          | 2528 |
| ● LOB 검색자(locator)의 경로를 상대 경로로 변경                                                               | 2528 |
| ● UPDATE JOIN 구문에서 분석 함수 사용 제한                                                                  | 2528 |
| ● 시스템 테이블 및 뷰에 대해서 <code>for update</code> 구문 사용 제한                                             | 758  |
| ● 뷰 생성시 데이터 타입의 정합성을 체크하지 않도록 변경                                                                | 175  |
| ● 뷰 질의 스펙의 select절에 NULL 값 사용 허용                                                                | 2529 |
| ● 테이블 생성 또는 컬럼 속성 변경시 <code>AUTO_INCREMENT</code> 와 <code>DEFAULT</code> 값이 공존할 수 없도록 변경        | 2529 |
| ● <b>ALTER INDEX ... REBUILD</b> 에 신규 컬럼을 추가시 오류 발생하도록 변경                                       | 2529 |
| ● ROWNUM 연산 값이 NUMERIC 타입 범위를 초과할 경우 오류 발생하도록 변경                                                | 2529 |
| ● 유틸리티                                                                                          | 176  |
| ● <code>csql</code> 의 인터랙티브 모드에서 SQL문 또는 PL/CSQL 문장(create문, body문 등) 작성 중 세션 명령어가 인식되도록 변경     | 2529 |
| ● <code>cubrid plandump -d</code> 수행시 삭제전 plan 정보 출력하지 않게 변경                                    | 2529 |
| ● HA                                                                                            | 1773 |
| ● <code>ha_mode</code> 가 on 이고 <code>ha_node_list</code> 에 자신의 호스트명 이 있는 경우 오류 발생하도록 변경         | 2529 |
| ● 기타                                                                                            | 178  |
| ● 시스템 테이블 중 <code>_db_serial</code> 의 컬럼명 변경 ( <code>att_name</code> → <code>attr_name</code> ) | 2530 |

|                                                                                                     |      |
|-----------------------------------------------------------------------------------------------------|------|
| ● 개선 사항 (성능 개선 포함)                                                                                  | 2530 |
| ● SQL                                                                                               | 180  |
| ● 상관 하위 질의 결과에 캐시 도입으로 성능 개선                                                                        | 181  |
| ● LIMIT 절에 바인드 변수와 계산식이 포함된 경우, Sort-Limit 최적화가 적용 되도록 개선                                           | 2530 |
| ● PL(JavaSP 포함)                                                                                     | 2531 |
| ● JNI 프로그램이 세그먼트 폴트 오류 발생시 자동으로 jvasp 서버를 재 구 동하게 개선                                                | 2531 |
| ● 여러 종류의 문자 인코딩(euckr, utf8)을 지원하도록 문자열 기능 개선                                                       | 2531 |
| ● 인덱스                                                                                               | 184  |
| ● 필터 인덱스를 생성할 때 where 절의 길이 제한 제거                                                                   | 739  |
| ● 다중 컬럼 인덱스에서 레코드를 읽을 때 필요한 midxkey.buf 크기 계산 방 식을 개선하여 인덱스 스캔 성능 향상                                | 2531 |
| ● 불필요한 연산 및 비교 회수를 최적화를 통해 인덱스 스캔 성능 향상                                                             | 185  |
| ● 옵티마이저                                                                                             | 186  |
| ● 옵티마이저의 전반적인 개선을 통한 성능 향상                                                                          | 188  |
| ● 저장 프로시저 실행 계획 개선                                                                                  | 191  |
| ● 불필요한 행에 X-LOCK 설정되지 않도록 개선하여 동시성 향상                                                               | 2533 |
| ● like 질의 조건 값으로 JavaSP 함수를 사용하는 경우에 인덱스 스캔을 사용 하도록 개선                                              | 2533 |
| ● 함수 인덱스와 함께 $\leq$ 와 $\geq$ 범위 조건 질의 사용시 범위 인덱스 스캔을 사용하도록 개선                                       | 192  |
| ● 유틸리티                                                                                              | 193  |
| ● 테이블과 뷰가 없는 DB에서 unloaddb를 실행시 그 외의 schema(user, serial, sp, server, synonym, grant)가 추출될 수 있도록 개선 | 2534 |

|                                                                                         |      |
|-----------------------------------------------------------------------------------------|------|
| ● unloaddb 성능 개선                                                                        | 1660 |
| ● <code>csql</code> 의 환경 변수명 명확성 개선                                                     | 72   |
| ● <code>cubrid spacedb</code> 결과값 개선                                                    | 2534 |
| ● Broker,CAS,CMS                                                                        | 194  |
| ● cubrid broker info 결과값 개선                                                             | 2535 |
| ● 잘못된 ACL 설정에 대해 오류 발생하여 사용자 인지 개선                                                      | 195  |
| ● CMS(CUBRID Manager Server)에서 TLS v1.2 클라이언트가 접속할 수 있도록 개선                             | 2536 |
| ● CMS 통해 생성된 broker/cas 프로세스 종료시 좀비 프로세스 남지 않게 개선                                       | 2536 |
| ● CMS getlogfileinfo() API를 호출하면 동일한 SQL LOGFILE의 정보를 1회만 반환되도록 개선                      | 2536 |
| ● CMS ha_status() API가 Master,Slave,Replica 구성된 HA환경에서 Replica Node의 상태가 출력되도록 개선       | 2536 |
| ● HA                                                                                    | 196  |
| ● <b>ha_node_list</b> 및 <b>ha_replica_list</b> 의 잘 못된 설정에 대해 명확한 오류 메시지로 출력으로 사용자 인지 개선 | 2536 |
| ● Failover와 Failback 발생에 대해 명확한 오류 메시지 출력으로 사용자 인지 개선                                   | 2049 |
| ● Salve 노드에서 <b>restoreslave</b> 실행시 잘 못된 데이터 불일치 오류 메시지가 발생하지 않도록 개선                   | 2536 |
| ● 기타                                                                                    | 197  |
| ● 사용자 편의를 위해 user hosts (cubrid_host.conf) 개선                                           | 2536 |
| ● DDL Audit 로그 개선                                                                       | 1917 |
| ● REDO 복구 과정에 병렬 스레드 적용하여 성능 개선                                                         | 2537 |

- 교착 상태(**deadlock**) 발생 시 데이터베이스명\_latest.event 로그파일에 기록되는 잠금관련 정보를 간소화 2537
- time\_format() 및 date\_format()의 성능 개선 2537
- TRACE 활성화 시 쿼리 성능 개선 2537
- 문자열 유형에 대한 불필요한 길이 검사 제거를 통해 성능 개선 201
- lock\_escalation 설정 값 만큼만 락 리소스를 유지하도록 변경해서 메모리 사용량 개선 2538
- optimization\_level 설정값에 잘못된 값을 입력 시 오류 출력을 통해 사용자 인지 개선 2538
- DB볼륨 및 로그의 생성 시간을 확인 개선 202
- SHOW INDEX CAPACITY 및 diagdb 에 fence key 정보 추가를 통해 사용자 인지 개선 2538
- 오류수정 2538
  - SQL 206
    - 다른 계정의 동일명 원격 서버 객체가 존재하는 경우, 사용자명(user schema)없이 원격 서버 객체 참조시 자신의 원격 서버 객체를 참조하지 못하는 오류 수정 1289
    - ORDER BY 절이 있는 스칼라 하위 질의에서 hidden 컬럼 처리 오류 수정 2540
    - ORDER BY를 포함하는 인라인 뷰와 스칼라 하위 질의의 뷰 머징 처리시 ROWNUM이 ORDER\_BY\_NUM() 으로 잘못 재 작성되는 오류 수정 2540
    - oracle\_style\_empty\_string=yes 설정시 'NULL || 문자열' 연산 결과가 NULL로 출력되는 오류 수정 2540
    - oracle\_style\_empty\_string=yes 설정시 REPLACE 함수 결과가 NULL로 출력되는 오류 수정 2540
    - 함수 인덱스를 기반으로 Covered Index 스캔 수행시 잘 못된 결과를 반환하는 오류 수정 2540
    - 항상 false (또는 NULL) 조건을 가진 CTE 질의 수행시 발생한 세그먼트 폴트 오류 수정 2540

- UNION ALL 구문에서 여러 테이블과 serial의 next\_value를 함께 사용한 질의 수행시 발생한 세그먼트 폴트 오류 수정 2540
- SELECT list 절에 alias 없이 255자 초과한 상수값을 사용한 질의 수행시 발생한 세그먼트 폴트 오류 수정 2540
- OUTER JOIN을 포함한 질의에서 뷰머징 처리 시 발생한 오류 수정 2540
- **SHOW CREATE VIEW** 질의 수행시 현재 사용자의 뷰와 public 사용자의 동일명 뷰가 함께 출력되는 오류 수정 2541
- repeat() 함수에 string\_max\_size\_bytes 설정값을 초과한 요청한 경우 오류 메시지가 출력되지 않고 NULL이 반환되는 오류 수정 2541
- 쿼리 최적화 중 INNER JOIN 제거 과정에서 테이블 위치가 잘못 처리되는 오류 수정 910
- PREPARE 구문에서 특정 내장 함수에 호스트 변수 사용시 발생한 오류 수정 807
- **DROP USER** 수행시 관련 시스템 카탈로그 테이블의 사용자 정보가 삭제되지 않은 오류 수정 1209
- SP 호출 및 WHERE 절에 호스트 변수를 사용한 Oracle 스타일 LEFT OUTER JOIN이 INNER JOIN으로 재 작성되지 않는 오류 수정 2541
- **ORDERBY\_NUM()**이 포함된 하위 질의가 뷰 머징에 잘 못된 결과를 출력하는 오류 수정 682
- 질의 최적화 중 조인 관계가 없는 테이블이 조인 테이블로 인식되는 오류 수정 207
- WHERE 절의 동등 조건(=)과 ORDER BY 절에서 사용된 컬럼이 동일할 때 ROWNUM 값이 0으로 출력되는 오류 수정 2541
- INNER JOIN과 Oracle 스타일 OUTER JOIN을 함께 사용하고, WHERE 절에 OUTER JOIN 조인 조건의 위치에 따라 발생한 오류 수정 2541
- **INSERT INTO tbl ... SELECT ... FROM 뷰테이블 ... ON DUPLICATE KEY UPDATE** 질의 수행시 잘못된 결과를 출력하는 오류 수정 2541
- 뷰에 분석 함수가 포함될 때, 분석 함수 내부의 컬럼 순서 번호가 잘못되어 잘못된 결과를 출력하는 오류 수정 369
- 범위 조건을 파이프 연산자(||)와 함께 사용시 범위 항목으로 인지하지 못하는 오류 수정 369

- PL(javasp) 369
  - JavaSP의 DATETIME 매개변수에 DATETIMELTZ 값을 시용시 발생한 오류 수정 2542
  - JavaSP의 선언된 Java 메서드명과 여는 괄호 사이의 공백으로 인해 JavaSP 수행되지 않는 오류 수정 2542
  - JavaSP의 미지원 타입을 보유한 테이블의 다른 컬럼을 인자로 사용시 발생한 오류 수정 2542
  - JavaSP에서 **Cannot allocate query entry any more** 발생한 오류 수정 2542
  - DBLINK 구문의 호스트 변수에 NULL 처리시 발생하는 오류 수정 2542
  - EUC-KR 오라클 DB에 DBLINK Insert 구문에 호스트 변수를 통해 한글 입력 시 한글 깨지는 오류 수정 2542
- 유틸리티 369
  - `loaddb` 실행 중 오류 발생 시 종료 코드가 3으로 설정되지 않는 오류 수정 2543
  - `loaddb -no-logging` 실행 후 잘 못된 **count(\*)** 값을 출력하는 오류 수정 2543
  - `loaddb` 실행 중 스키마명이 없는 serial 처리시 발생한 오류 수정 2543
  - `loaddb` 실행 중 오류 발생 시 멈추지 않고 계속 실행되는 오류 수정 1667
  - `loaddb` 실행 중 대상이 시스템 테이블인 synonym 처리시 발생하는 오류 수정 2543
  - 11.4 이전 버전의 `unloaddb` 파일을 `loaddb` 실행 시, 트리거 시스템 테이블 컬럼 (condition, action\_definition)에 스키마명이 잘못 저장되거나 누락되는 오류 수정 2543
  - `unloaddb` 실행 시 Reverse Unique Index의 주석이 누락되는 오류 수정 2543
  - `unloaddb` 실행 시 auto\_increment 설정과 함께 PK로 선언된 컬럼의 auto\_increment 값이 1로 초기화되는 오류 수정 2543
  - `unloaddb -split-schema-files` 실행 시 Unique Index가 없는 경우 'db명\_schema\_uk' 파일이 생성되는 오류 수정 2543



- `unloaddb -i (-input-class-only)` 옵션을 함께 실행시 특정 ALTER SERIAL 문이 누락되는 오류 수정 2543
- `unloaddb` 실행시 serial의 current\_val과 max\_val 동일한 경우 발생한 오류 수정 2543
- `unloaddb` 실행시 1MB 초과한 JSON 타입 데이터 처리에 대해 무한 루프 발생하는 오류 수정 2543
- 일반 사용자로 `unloaddb` 실행시 뷰의 query\_spec 출력시 일반 사용자와 동일한 스키마명이 제거되지 않는 오류 수정 2544
- DBA 사용자로 `unloaddb` 실행시 serial과 trigger의 스키마명이 누락되는 오류 수정 2544
- DBA 사용자가 `-as-dba` 옵션 없이 `unloaddb` 실행시 다른 사용자가 부여한 PROCEDURE 권한(`GRANT ... ON PROCEDURE`)이 출력되는 오류 수정 2544
- 일반 사용자가 `unloaddb` 실행시, 트리거의 condition과 action\_definition에 일반 사용자의 동일한 스키마명이 제거되지 않는 오류 수정 2544
- `backupdb` 실행시 불필요한 보관 로그 볼륨이 생성되는 오류 수정 1645
- FlashBack를 강제 종료 (Ctrl+C)시 작업 종료되지 않은 오류 수정 2544
- 특정 상황에서 backupdb 수행시 hang 발생하는 오류 수정 2544
- Broker,CAS, CMS 369
  - addBatch()와 executeBatch() 처리 시 발생할 수 있는 메모리 누수(memory leak) 수정 2544
  - gettranscationinfo() 함수 호출시 잘 못된 결과를 출력하는 오류 수정 2544
- HA 369
  - Master, Slave 및 Replica 노드로 구성된 HA 환경에서 ha\_replica\_delay 설정값이 60초 이상으로 설정된 경우, Replica 노드에서 slave 대상 복제 로그가 삭제되지 않는 오류 수정 2545
  - applylogdb 수행중 생성된 sql log 파일이 특정 조건에서 자동 삭제되지 않는 오류 수정 2545

|                                                                                                                                                                                          |             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| ● 주의사항                                                                                                                                                                                   | 2545        |
| ● 신규 주의 사항                                                                                                                                                                               | 2545        |
| ● CUBRID 11.4의 데이터베이스 볼륨은 CUBRID 11.3 및 그 이전 버전의 볼륨과 호환되지 않는다.                                                                                                                           | 2545        |
| ● 기존 주의 사항                                                                                                                                                                               | 484         |
| ● 11.3                                                                                                                                                                                   | 484         |
| ● 질의 최적화기의 통계 정보는 DDL 문이 수행될 때 자동으로 갱신되지 않고, 사용자가 직접 UPDATE STATISTICS 문을 실행하여 통계정보를 갱신해야 함 (조인 방식 참조)                                                                                   | 81925462546 |
| ● ALTER TABLE 구문에서 AUTO_INCREMENT 속성 갖는 컬럼 또는 default 값을 가지고 있는 컬럼의 타입 변경시, AUTO_INCREMENT 속성을 사용할 수 없는 타입으로 변경하거나 기존 default 값을 변환할 수 없는 타입으로 변경할 때, 에러가 발생함 (CHANGE/MODIFY 절 참조)       | 2546        |
| ● 서버측 JDBC에서 cubrid.jdbc.driver.CUBRIDDriver.getDefaultConnection()을 통해 Connection 객체를 가져올 수 없음. 대신, DriverManager.getConnection("jdbc:default:connection:");을 사용해야 함 (Connection 생성 참조) | 2546        |
| ● USE INDEX(USING INDEX) 구문에서 유효하지 않은 인덱스명이나 테이블명을 지정하는 경우 에러 처리하지 않고 무시함 (해당 사항을 log 파일에 남기려면 error_log_warning 설정값을 yes로 설정하면 됨)                                                       | 2546        |
| ● CREATE INDEX 생성시 COMMENT 절의 위치가 WITH 절 또는 INVISIBLE 절 뒤인 구문 마지막 위치로 변경됨 (인덱스 정의문 참조)                                                                                                   | 2612546     |
| ● 11.2                                                                                                                                                                                   | 485         |
| ● CUBRID 11.2의 데이터베이스 볼륨은 CUBRID 11.1 및 그 이전 버전의 볼륨과 호환되지 않는다.                                                                                                                           | 2546        |
| ● User schema 개념 도입으로 사용자 별로 동일한 객체명을 사용할 수 있으며, 지원에 따라 다음 동작이 변경됨                                                                                                                       | 1082        |
| ● JavaSP의 "jdbc:default:connection:" 또는 getDefaultConnection() 사용 시 다음 함수 및 동작 변경됨 (JavaSP 주의 사항 참조)                                                                                     | 2547        |
| ● Truncate table 수행 시 FK의 set null 또는 cascade가 있는 경우 동작이 변경됨 (TRUNCATE 참조)                                                                                                               | 8042547     |

- Alter change/modify 시 기입하지 않은 컬럼 속성은 유지되게 변경되었으며, alter 구문으로 auto\_increment, on update property를 제거할 수 없게 변경됨 (CHANGE/MODIFY 절 참조) 2547
- 조건절에 컬럼명만 존재하는 경우 오류 처리되도록 변경됨 1083
- Multiple SQL는 반드시 세미콜론으로 구분하도록 변경됨 2548
- CUBRID 패키지내의 CCI Driver 디렉토리가 \$CUBRID/lib, \$CUBRID/include에서 \$CUBRID/cci/lib, \$CUBRID/cci/include로 각각 변경됨 (CCI 개요 참조) 2548
- 백업 시 압축(-z, --compress) 옵션이 기본으로 설정되도록 변경됨 (backupdb 참조) 2548
- 새로운 기능 추가로 인해 시스템 카탈로그 정보가 변경되거나 추가됨 (시스템 카탈로그 참조) 12912548
- 11.0 1083
  - CUBRID 11.0의 데이터베이스 볼륨은 CUBRID 10.2 및 그 이전 버전의 볼륨과 호환되지 않는다. 2548
  - 옵션 없이 테이블 생성 시 reuse\_oid 테이블로 생성된다. 229
  - CHAR 데이터 타입의 최대 길이는 256M 문자열로 변경되었다. 2548
  - 문자열 데이터 타입에 길이보다 큰 문자열이 들어가는 경우 오류가 발생하게 수정되었다. 1084
  - 문자열 끝의 공백 문자를 인식하게 수정하여, 문자열 끝의 공백 문자에 따라 다른 문자열로 인식된다. 1084
  - 통계 수집 방식의 변경으로 주기적인 통계 수집 수행이 필요하다. 1085
- 10.2 1086
  - CUBRID 10.2의 데이터베이스 볼륨은 CUBRID 10.1 및 그 이전 버전의 볼륨과 호환되지 않는다. 2549
- 10.1 2549
  - CUBRID 10.1의 데이터베이스 볼륨은 CUBRID 10.0 및 그 이전 버전의 볼륨과 호환되지 않는다. 2549

- 10.0 및 이하 버전 1361

---

- CUBRID 10.0의 데이터베이스 볼륨은 CUBRID 9.3 및 그 이전 버전의 볼륨과 호환되지 않는다. 2549

---

- DB를 생성할 때 로케일(언어 및 문자 집합)을 지정 1362

---

- CUBRID\_CHARSET 환경 변수 제거 2549

---

- [JDBC] 연결 URL의 zeroDateTimeBehavior 값이 “round”일 때 TIMESTAMP의 zero date가 ‘0001-01-01 00:00:00’에서 ‘1970-01-01 00:00:00’(GST)으로 변경 (CUBRIDSUS-11612) 2549

---

- AIX에서 CUBRID SH 패키지 설치 시 권장 사항 (CUBRIDSUS-12251) 2549

---

- CUBRID\_LANG 제거, CUBRID\_MSG\_LANG 추가 2550

---

- CCI 응용 프로그램에서 여러 개의 질의를 한 번에 수행한 결과의 배열에 대한 오류 처리 방식 수정 (CUBRIDSUS-9364) 2550

---

- java.sql.XAConnection 인터페이스에서 HOLD\_CURSORS\_OVER\_COMMIT을 지원하지 않음 (CUBRIDSUS-10800) 2552

---

- 9.0부터 STRCMP가 대소문자를 구분하여 동작 408

---

- 2008 R4.1 버전부터 CCI\_DEFAULT\_AUTOCOMMIT의 기본값이 ON으로 변경됨 (CUBRIDSUS-5879) 2552

---

- 2008 R4.0 버전부터 페이지 단위를 사용하는 옵션 및 파라미터가 볼륨 크기 단위를 사용하도록 변경됨 (CUBRIDSUS-5136) 2553

---

- 2008 R4.0 베타 이전 버전 사용자는 db 볼륨 크기를 설정할 때 주의할 것 (CUBRIDSUS-4222) 2553

---

- 2008 R4.0 이전 버전의 일부 시스템 파라미터의 기본값 변경 (CUBRIDSUS-4095) 2553

---

- 시스템 파라미터가 잘못 설정된 경우 데이터베이스 서비스, 유틸리티 및 응용 프로그램을 수행할 수 없도록 변경됨 (CUBRIDSUS-5375) 2554

---

- CUBRID 32비트 버전에서 2G를 초과하는 값을 사용하여 data\_buffer\_size를 설정할 경우 데이터베이스 구동에 실패함 (CUBRIDSUS-5349) 2554

- Windows Vista 및 그 이후 버전에서 CUBRID 유틸리티를 사용한 서비스 제어 시 권장 사항 (CUBRIDSUS-4186) 2554

---

- CUBRID 소스 빌드 후 수행 시, 매니저 서버 프로세스 관련 오류 발생 (CUBRIDSUS-3553) 2555

---

- 2008 r3.0 또는 그 이전 버전에서 사용하던 GLO 클래스 지원 중단 (CUBRIDSUS-3826) 2555

---

- 마스터와 서버 프로세스 사이의 프로토콜이 변경된 경우 또는 두 버전이 동시에 실행되는 경우 포트 설정 필요 (CUBRIDSUS-3564) 2556

---

- JDBC에서 URL 문자열로서 연결 정보를 입력할 때 물음표 명시 (CUBRIDSUS-3217) 2557

---

- 데이터베이스명에 @을 포함할 수 없음 (CUBRIDSUS-2828) 2557

## 릴리스 노트 정보

본 문서는 CUBRID 11.4(빌드번호 11.4.0.1778)에 관한 정보를 포함한다.

CUBRID 11.4는 CUBRID 11.3 버전에서 발견된 오류 수정 및 기능 개선과 이전 버전들에 반영된 모든 오류 수정 및 기능 개선을 포함한다.

CUBRID 11.3에 대한 정보는 [https://www.cubrid.org/manual/ko/11.3/release\\_note/index.html](https://www.cubrid.org/manual/ko/11.3/release_note/index.html) 에서 확인할 수 있다.

CUBRID 11.2에 대한 정보는 [https://www.cubrid.org/manual/ko/11.2/release\\_note/index.html](https://www.cubrid.org/manual/ko/11.2/release_note/index.html) 에서 확인할 수 있다.

CUBRID 11.0에 대한 정보는 [https://www.cubrid.org/manual/ko/11.0/release\\_note/index.html](https://www.cubrid.org/manual/ko/11.0/release_note/index.html) 에서 확인할 수 있다.

CUBRID 10.2에 대한 정보는 [https://www.cubrid.org/manual/ko/10.2/release\\_note/index.html](https://www.cubrid.org/manual/ko/10.2/release_note/index.html) 에서 확인할 수 있다.

CUBRID 10.1에 대한 정보는 [https://www.cubrid.org/manual/ko/10.1/release\\_note/index.html](https://www.cubrid.org/manual/ko/10.1/release_note/index.html) 에서 확인할 수 있다.

CUBRID 10.0에 대한 정보는 [https://www.cubrid.org/manual/en/10.0/release\\_note/index.html](https://www.cubrid.org/manual/en/10.0/release_note/index.html) 에서 확인할 수 있다.

CUBRID 9.3에 대한 정보는 [https://www.cubrid.org/manual/ko/9.3.0/release\\_note/index.html](https://www.cubrid.org/manual/ko/9.3.0/release_note/index.html) 에서 확인할 수 있다.

## 릴리스 개요

CUBRID 11.4는 새로운 기능, 주요 변경 사항, 개선 사항 및 이전 버전에서 발생한 다수의 오류 수정이 포함된 최신 안정화 버전이다.

CUBRID 11.4는

1. 오라클 호환성을 위한 **PL/CSQL** 지원
2. 대용량 처리 지원을 위한 **HASH JOIN** 추가
3. 옵티마이저 및 인덱스 처리 방식 개선을 통한 성능 향상
4. 데이터 복구시 병렬 처리로 인한 성능 향상
5. 결과 캐시 확장을 통한 성능 개선
6. 데이터 덤프 성능 향상
7. 메모리 모니터링 기능 추가
8. 접근 제어 기능 향상
9. 백업/복구 운영 사용자 편의 향상

을 통해서 새로운 기능과 더불어 다양한 부분에서의 성능 개선이 이루어 졌다.

## 드라이버 호환성

- CUBRID 11.4는 하위 드라이버와도 호환 가능하지만, 11.4에서 제공되는 기능을 사용하기 위해서는 최신 드라이버 사용을 권장한다.

## 11.4 변경사항

### 기능 개선

#### PL/CSQL

- 절차적 SQL 언어인 PL/SQL(Oracle 제공)과의 호환성을 갖는 CUBRID PL/CSQL 지원.
- 자세한 문법 및 사용법은 매뉴얼(**CUBRID 절차적 언어**) 참조 바람.

## SQL

### HASH JOIN 지원

- 조인 처리 방식 중 하나인 조인 조건 컬럼을 기준으로 해시 테이블을 생성해서 조인 처리하는 HASH JOIN을 지원한다.
- 옵티마이저가 HASH JOIN을 사용하기 위해서는 조인 힌트를 사용해야만 한다.
  - `/*+ USE_HASH */` 힌트가 추가되면 HASH JOIN 고려함.
  - `/*+ NO_USE_HASH */` 힌트가 추가되면 HASH JOIN 사용하지 않음.

### SERIAL 소유자 변경 SQL 구문 지원

- `call change_serial_owner()` 메소드만으로 SERIAL 소유자를 변경 가능했던 것을 SQL문으로 소유자 변경할 수 있도록 ALTER SERIAL 구문을 확장 지원함.

```
ALTER SERIAL serial_name OWNER TO user_name
```

- DBA와 DBA 그룹 멤버만 SERIAL 소유자 변경이 가능 함.
- 권한이 없거나 변경 대상인 SERIAL이 없을 경우에는 에러가 발생함.

### 생성된 그룹에 구성원을 추가/삭제할 수 있는 SQL 구문 지원

- `call add_member()`와 `call drop_member()` 메소드만을 통해 생성된 그룹에 사용자를 추가/삭제 할 수 있던 것을 SQL문으로 사용자를 추가/삭제할 수 있도록 ALTER USER 구문을 확장 지원함.
  - HA 환경에서 메소드 호출 통해 생성된 그룹에 사용자를 추가/삭제하는 경우, 마스터와 슬레이브에서 별도 작업이 필요했던 작업을 ALTER USER 구문을 통해 쉽게 처리할 수 있도록 개선됨.

```
ALTER USER user_name
[PASSWORD password] |
[ADD MEMBERS user_name {, user_name}...] |
[DROP MEMBERS user_name {, user_name}...]
[COMMENT 'comment_string']
```

#### - DBA/DBA 그룹 구성원

- DBA 그룹에 구성원 추가/삭제
- 일반 사용자 (**PUBLIC** 포함) 그룹에 구성원 추가/삭제

#### - 일반 사용자

- 자신의 그룹에만 구성원 추가/삭제

## 쿼리 캐시 확장 지원

- 전체 질의 단위만으로도 사용할 수 있던 쿼리 캐시 기능을 하위 질의 단위까지 사용할 수 있도록 확장 지원함.
- CTE의 하위 질의에 쿼리 캐시 적용

```
with cte0 as (select /* query_cache */ count(*) from public.game
where host_year > 2000),
cte1 as (select /* query_cache */ count(*) from public.game where
host_year = 2004)
select * from cte1;
```

### Trace Statistics:

```
SELECT (time: 0, fetch: 4, fetch_time: 0, ioread: 0)
 SCAN (temp time: 0, fetch: 4, ioread: 0, readrows: 1, rows: 1)
 SUBQUERY (uncorrelated)
 CTE (non_recursive_part)
 SELECT (time: 0, fetch: 0, fetch_time: 0, ioread: 0)
 RESULT CACHE (reference count: 1)
 CTE (non_recursive_part)
 SELECT (time: 0, fetch: 0, fetch_time: 0, ioread: 0)
 RESULT CACHE (reference count: 1)
```

- 비 상관 하위 질의에 쿼리 캐시 적용

```
select count(host_year)
from (select /* query_cache */ code from public.nation where code
like 'K%') n, public.game g
where n.code = g.nation_code;
```

### Trace Statistics:

```
SELECT (time: 218, fetch: 872, fetch_time: 2, ioread: 0)
 SCAN (temp time: 0, fetch: 4, ioread: 0, readrows: 7, rows: 7)
 SCAN (table: public.game), (heap time: 215, fetch: 868, ioread:
0, readrows: 60571, rows: 386)
 SUBQUERY (uncorrelated)
 SELECT (time: 0, fetch: 0, fetch_time: 0, ioread: 0)
 RESULT CACHE (reference count: 3)
```

- trace 정보에 쿼리 캐시(RESULT CACHE) 사용 여부 표시 (reference count : 쿼리 캐시 참조 횟수)



## 조인 순서를 명시적으로 제어 가능한 LEADING 힌트 지원

- 옵티마이저에게 특정 테이블들의 조인 수행 순서를 명시적으로 지정하기 위해 사용하는 것으로 ORDERED 힌트보다 테이블 조인 순서를 유연하게 제어 가능하도록 함.

```
SELECT /** LEADING(e j) */ *
FROM employees e, departments d, job_history j
WHERE e.department_id = d.department_id
AND e.hire_date = j.start_date;
```

### - LEADING 힌트 무시 조건

- 조인 그래프의 종속성으로 인해 지정된 테이블들을 먼저 조인할 수 없는 경우 무시
- ORDERED 힌트가 있으면 모든 LEADING 힌트 무시
- 여러 LEADING 힌트 사용시, 첫번째 LEADING 힌트만 적용되고 나머지 힌트는 무시

## 유틸리티

### cubrid plandump의 특정 plan만 삭제하는 옵션 지원

- 옵션 : -s

### JNI 사용한 JavaSP를 위한 별도의 loadjava 옵션 지원

- 옵션 : -j 또는 -jni
- 해당 옵션을 사용하지 않고 로드된 JNI 사용한 javasp는 수행시 오류 발생

## 메모리 모니터링 유틸리티 지원

- 메모리 모니터링 유틸리티인 cubrid memmon는 서버 프로세스(cub\_server)의 메모리 관리 모듈로부터 다음과 같은 힙 메모리 정보를 얻어 출력함.
  - 서버 프로세스가 할당해서 사용중인 힙 메모리 총 사용량
  - 메모리를 할당 요청한 소스코드와 라인 정보 기준으로 세부적인 메모리 할당량 정보
- 서버 프로세스의 메모리 관리 모듈을 사용하기 위해서는 시스템 파라미터 **enable\_memory\_monitoring** 가 yes로 설정한 후, 서버를 재 구동해야 함.

- 자세한 사용 방법 및 내용은 매뉴얼(memmon) 참조.

## diagdb -d 9 에 하나의 테이블만 덤프할 수 있는 옵션 지원

- 옵션 : -n class-name
- 파티션 테이블인 경우에는 모든 파티션을 덤프하고 서브 파티션 테이블인 경우는 해당 파티션 테이블만 덤프함.

## Broker,CAS,CMS

## 브로커별 통신 버퍼 크기 조절을 위한 파라미터 지원

- 브로커별 클라이언트에 결과 전송시 사용하는 버퍼 크기를 조절할 수 있는 **NET\_BUF\_SIZE** 파라미터 지원.
- **NET\_BUF\_SIZE**에 설정 가능한 값
  - 16K(기본값)
  - 32K
  - 48K
  - 64K

## ACL 정책 정의를 위한 파라미터 지원

- 브로커의 ACL 사용(**ACCESS\_CONTROL=ON**)하는 경우, ACCESS\_CONTROL\_FILE에 미 작성된 브로커에 대한 기본 동작 정책을 설정할 수 있는 ACCESS\_CONTROL\_DEFAULT\_POLICY 파라미터 지원.
- **cubrid\_broker.conf**의 **[broker]** 섹션에서 ACCESS\_CONTROL\_DEFAULT\_POLICY에 **DENY** 또는 **ALLOW** 값을 설정 (기본 값은 **DENY**)
- ACCESS\_CONTROL\_FILE에 미 작성된 브로커에 대해 ALLOW인 경우에는 모든 클라이언트에 대해 접속 허용하고, DENY인 경우에는 반대로 접속을 불허한다.

```
$ cubrid broker acl status
ACCESS_CONTROL=ON
ACCESS_CONTROL_DEFAULT_POLICY=DENY
ACCESS_CONTROL_FILE=cubrid_acl.conf
```

```
[%query_editor]
```

```
testdb:dba:acl_ip_list.conf

CLIENT IP LAST ACCESS TIME
=====
172.29.80.1
192.168.0.31
172.31.0.175

[%broker1]

++ cubrid broker acl: success
```

- **[%broker1]** 은 cubrid\_acl.conf 파일에 클라이언트 관련 접속 정보가 미 작성되어 있으므로 기본 정책인 **DENY**에 의해 모든 클라이언트에 대해 접속을 허용하지 않음

## 기타

### Non-HA DB서버가 비정상 종료 시 자동 재 구동 기능 지원

Non-HA 환경에서 DB 서버가 OOM Killer 등으로 인해 비정상 종료될 경우, 자동으로 서버를 다시 시작할 수 있는 기능을 지원

- 특이 사항
  - 정상 종료시에는 자동으로 재시작 되지 않음.
  - `auto_restart_server` 파라미터로 ON/OFF 가능.
  - 짧은 시간 내 반복적으로 비 정상 종료되거나, 일정 횟수 이상 구동 실패시 자동 재 구동하지 않음

### 백업본을 신규 DB로 복구하는 스크립트 지원

- 운영 서버에서 백업본으로 데이터 복구가 필요할 때, 추가 계정 생성 및 추가 엔진 설치 등의 번거로운 작업이 없어 간단히 새로운 DB로 복구할 수 있는 `restore_to_newdb.sh` 스크립트 지원. 자세한 사용 방법 및 내용은 매뉴얼 ([restore\\_to\\_newdb.sh 스크립트](#)) 참조.

## 스펙변경

### SQL

#### CHAR 타입의 최대 문자 개수를 2048로 제한( 변경 전 최대 : 268,435,456 )

- 메모리 할당 오류 등의 잠재 오류를 방지하기 위해 CHAR 타입의 최대 문자 개수를 268,435,456에서 2048로 제한
- 11.4 이전 버전 DB 이관시 2048 초과되는 CHAR 타입은 VARCHAR 타입으로 전환 필요

#### LOB 검색자(locator)의 경로를 상대 경로로 변경

- LOB 데이터 파일의 이동 및 관리 편의성을 개선하기 위해, LOB 관련 파일 경로를 절대 경로에서 상대 경로로 변경
- databases.txt의 DB의 LOB 디렉토리의 위치의 변경시, 반드시 변경전 LOB 디렉토리에 존재하는 LOB 파일들을 변경 후 LOB 디렉토리로 이동 또는 복사 필요

#### UPDATE JOIN 구문에서 분석 함수 사용 제한

- 오동작 방지를 위해 2개 이상 테이블의 UPDATE JOIN 구문에서 분석 함수를 사용할 수 없게 제한
- 11.4 이전 버전에서 작성된 해당 질의는 11.4 버전에서 오류 발생하므로 수정 필요

#### 시스템 테이블 및 뷰에 대해서 `for update` 구문 사용 제한

- 시스템 테이블에서 X-LOCK 발생으로 인한 대기 현상을 방지하기 위해, 시스템 테이블 및 뷰에 대해 FOR UPDATE 문을 사용할 경우 오류가 발생하도록 변경

#### 뷰 생성시 데이터 타입의 정합성을 체크하지 않도록 변경

- 오라클 호환성을 높이기 위해 뷰 생성 시점에 타입 검사를 하지 않고 실행 시점에 타입 검사하는 것으로 변경
- 타입 변환이 불가능한 경우, 뷰 실행 시 오류 발생함.

## 뷰 질의 스펙의 select절에 NULL 값 사용 허용

- 오라클 호환성을 높이기 위해 뷰 질의 스펙의 select 절에 NULL 값 사용 허용

```
CREATE VIEW a_view(col1) AS select NULL as col1 from a_tbl;
```

## 테이블 생성 또는 컬럼 속성 변경시 AUTO\_INCREMENT 와 DEFAULT 값이 공존할 수 없도록 변경

- CREATE TABLE 및 ALTER COLUMN 구문 수행 시 AUTO\_INCREMENT 및 DEFAULT 속성을 함께 사용시 오류 발생하도록 변경

## ALTER INDEX ... REBUILD에 신규 컬럼을 추가시 오류 발생하도록 변경

- 신규 컬럼을 추가한 후 REBUILD시 신규 컬럼을 추가할 수 없다는 오류가 발생하도록 변경
- 11.4 이전 버전에서 작성된 해당 질의는 11.4 버전에서 오류 발생하므로 수정 필요

## ROWNUM 연산 값이 NUMERIC 타입 범위를 초과할 경우 오류 발생하도록 변경

## 유틸리티

### csql 의 인터랙티브 모드에서 SQL문 또는 PL/CSQL 문장(create문, body문 등) 작성 중 세션 명령어가 인식되도록 변경

cubrid plandump -d 수행시 삭제전 plan 정보 출력하지 않게 변경

## HA

### ha\_mode 가 on 이고 ha\_node\_list 에 자신의 호스트명 이 있는 경우 오류 발생하도록 변경

- 11.4 이전 버전에서는 replica 모드로 변경

## 기타

### 시스템 테이블 중 \_db\_serial의 컬럼명 변경 (att\_name → attr\_name)

- 11.4 이전 버전에서 해당 컬럼명을 명시적으로 사용한 검색 질의는 11.4 버전에서 오류 발생하므로 수정 필요

## 개선 사항 (성능 개선 포함)

## SQL

### 상관 하위 질의 결과에 캐시 도입으로 성능 개선

- 상관 하위 질의는 주 질의의 각 행에 대해 개별적으로 수행된다. 이 과정에서 동일한 조건값이 많을 경우, 동일한 하위 질의가 반복적으로 수행되어 성능 저하를 유발할 수 있다. 이를 방지하기 위해 캐시를 적용하여, 동일 조건값의 하위 질의가 반복 수행되지 않도록 최적화하여 성능을 개선함.

### LIMIT 절에 바인드 변수와 계산식이 포함된 경우, Sort-Limit 최적화가 적용되도록 개선

```
drop table if exists tbl1;
create table tbl1 (col1 int, col2 int);
insert into tbl1 select rownum, random() % 100000 +1 from db_class
a, db_class b, db_class c, db_class d limit 100000;
create index idx on tbl1(col1);

prepare stmt from
'SELECT a.col1, a.col2
FROM tbl1 a
 LEFT JOIN tbl1 b ON a.col1 = b.col1
 LEFT JOIN tbl1 c ON a.col1 = c.col1
ORDER BY a.col2,a.col1
LIMIT ?*10,?';

execute stmt using 10,10;
```

## PL(JavaSP 포함)

JNI 프로그램이 세그먼트 폴트 오류 발생시 자동으로 javasp 서버를 재 구동하게 개선

여러 종류의 문자 인코딩(euckr, utf8)을 지원하도록 문자열 기능 개선

- 저장 프로시저의 문자열 인코딩 방식을 byte 배열로 처리해서 utf8 인코딩 이외에 euckr 인코딩도 지원하도록 개선

## 인덱스

필터 인덱스를 생성할 때 where 절의 길이 제한 제거

다중 컬럼 인덱스에서 레코드를 읽을 때 필요한 midxkey.buf 크기 계산 방식을 개선하여 인덱스 스캔 성능 향상

- 기존에는 다중 컬럼 인덱스에서 키 길이 계산을 반복적으로 수행했던 것을 각 컬럼의 **OFFSET** 을 추가하여 계산 없이 직접 참조하도록 개선하여 **바이너리 검색, 키 필터링, DML 실행 시** 성능 향상.

불필요한 연산 및 비교 회수를 최적화를 통해 인덱스 스캔 성능 향상

- Range-Scan 과정에서 키를 읽고 처리하는 방식을 최적화하여 불필요한 연산을 줄임.
- Range-Scan 과정에서 *upper\_key* 와의 불필요한 비교를 줄여 성능 향상.
- 인덱스 컬럼 ID를 반복적으로 비교하는 작업을 줄여 연산 최적화.
- 리프 노드 헤더에 공통 접두사 (common prefix) 정보를 추가하여, 압축된 리프 노드 처리시 반복 비교 및 계산하는 부분을 최적화

## 옵티마이저

옵티마이저의 전반적인 개선을 통한 성능 향상

- 샘플링 통계 수집시 통계의 정확도를 높이기 위해 **샘플링 페이지 수**를 5000으로 늘림
- 고유 값 수(**NDV: Number of Distinct Values**) 수집 방법 개선으로 정확도를 높임

- 규칙 기반 최적화(**RBO**: Rule Based Optimization)를 최소화하여 정확도를 높임
  - 경험적 가중치(*Heuristic Factor*) 사용을 최소화
  - 비용 차이가 1.x 이내일때 사용하던 RBO를 제거하고 비용 기반 최적화 (**CBO**: Cost Based Optimization)로 동작하도록 개선
- 비용 계산식(cost formula) 개선
  - 인덱스 필터 스캔 선택도 반영
    - 인덱스 스캔 시 데이터 I/O 비용 계산 시 선택도를 고려하도록 변경
  - NOT LIKE 연산자 선택도 추가
  - 함수 기반 인덱스(*function-based index*) 선택도 추가
  - NDV(고유 값 수)가 1% 이하로 높을 경우, 통계 가중치(**weight**) 조정
  - SSCAN\_DEFAULT\_CARD 도입
    - NL JOIN(중첩 루프 조인) 시 카디널리티(결과 건수) 추정이 너무 낮으면 비효율적인 플랜 방지.
  - LIMIT 절을 고려 비용 및 카디널리티 설정을 도입하고, 소량 데이터 조회 성능을 높이기 위해 **LIMIT 최적화** 및 기본 카디널리티 값을 조정
- 불필요한 조건 제거
  - 조인(predicate) 조건 중 논리적으로 이미 평가된 중복 조건 제거 개선
  - 불필요한 INNER JOIN 제거
- 추가 개선
  - trace 정보에 **fetch\_time** 추가로 실행 시간 추적 강화.
  - `index_ss` 힌트 없이도 인덱스 스킵 스캔(**index skip scan**) 선택되도록 개선
  - PK(Primary Key) 보다 더 좋은 인덱스를 선택할 수 있도록 개선

## 저장 프로시저 실행 계획 개선

- 저장 프로시저의 실행 계획 처리를 내장 함수의 실행 계획과 동일하게 처리하도록 수정
- 개선 사항
  - 인덱스 스캔 가능



- 불 필요한 조인 제거
- 상관 하위 질의에서 저장 프로시저 사용시 결과 캐시 사용 가능

## 불필요한 행에 X-LOCK 설정되지 않도록 개선하여 동시성 향상

- 특정 질의에서 LOCK이 너무 이른 시점에 생성되어, 불필요한 LOCK이 발생하고 있었다. 이를 개선하기 위해, 모든 조건이 평가된 후에 불필요한 LOCK을 해제하도록 개선함.

like 질의 조건 값으로 JavaSP 함수를 사용하는 경우에 인덱스 스캔을 사용하도록 개선

```
CREATE OR REPLACE FUNCTION stringTest(x String) RETURN String AS
LANGUAGE JAVA NAME 'SpTest.typeeteststring(java.lang.String) return
java.lang.String';

CREATE TABLE tbl (ord INT, col_int INT, col_char char(1));
CREATE INDEX i_tbl ON tbl (ord);
CREATE INDEX i_tbl_char ON tbl (col_char);

INSERT INTO tbl VALUES (1,10,'a');
INSERT INTO tbl VALUES (2,10,'b');
INSERT INTO tbl VALUES (3,10,'c');
INSERT INTO tbl VALUES (4,10,'d');
INSERT INTO tbl VALUES (5,10,'e');

SELECT count(*) AS "like" FROM tbl WHERE col_char LIKE (SELECT
stringTest('a') FROM dual);
```

함수 인덱스와 함께 `<=` 와 `>=` 범위 조건 질의 사용시 범위 인덱스 스캔을 사용하도록 개선

## 유틸리티

테이블과 뷰가 없는 DB에서 unloaddb를 실행시 그 외의 schema(user, serial, sp, server, synonym, grant)가 추출될 수 있도록 개선

## unloaddb 성능 개선

- 데이터 추출 시간을 크게 단축하기 위해 멀티 스레드 방식을 도입을 통해 성능 개선
  - unloaddb 옵션
    - **-thread-count**: 동시 실행 스레드 수(0~127)
    - **enhanced-estimates**: 정확한 레코드 수 예측(verbose 모드 전용)

## csql 의 환경 변수명 명확성 개선

- CSQL 환경 변수명의 명확도를 높이기 위해 CUBRID\_CSQL 접두사를 추가하여 CUBRID 관련 환경 변수임을 명확히 표시하게 개선
  - 변경 사항
    - EDITOR → CUBRID\_CSQL\_EDITOR
    - SHELL → CUBRID\_CSQL\_SHELL
    - FORMATTER → CUBRID\_CSQL\_FORMATTER
- 기존 버전과의 호환성을 위해 기존 환경 변수명도 사용할 수 있음.

## cubrid spacedb 결과값 개선

- 사용자가 사용하는 Overflow 페이지가 SYSTEM 페이지로 잘못 분류된 것을 목적에 따라 INDEX와 HEAP로 분류되도록 개선

Broker,CAS,CMS

### cubrid broker info 결과값 개선

- 설정값에 사용된 디렉토리명을 절대 경로로 표시
- **ADMIN\_LOG\_FILE** 표시 추가

잘못된 ACL 설정에 대해 오류 발생하여 사용자 인지 개선

CMS(CUBRID Manager Server)에서 TLS v1.2 클라이언트가 접속할 수 있도록 개선

CMS 통해 생성된 broker/cas 프로세스 종료시 좀비 프로세스 남지 않게 개선

CMS getlogfileinfo() API를 호출하면 동일한 SQL LOGFILE의 정보를 1회만 반환되도록 개선

CMS ha\_status() API가 Master,Slave,Replica 구성된 HA환경에서 Replica Node의 상태가 출력되도록 개선

HA

**ha\_node\_list** 및 **ha\_replica\_list**의 잘 못된 설정에 대해 명확한 오류 메세지 출력으로 사용자 인지 개선

Failover와 Failback 발생에 대해 명확한 오류 메시지 출력으로 사용자 인지 개선

Salve 노드에서 **restoreslave** 실행시 잘 못된 데이터 불일치 오류 메세지가 발생하지 않도록 개선

기타

사용자 편의를 위해 user hosts (cubrid\_host.conf) 개선

- **0.0.0.0 your\_hostname** 추가
- 대소문자 구분없이 호스트명을 사용할 수 있도록 개선
- 유효성 검증 (IP 주소,호스트명)에 오류가 있는 경우 오류가 발생하도록 변경

## DDL Audit 로그 개선

- CAS에 발생하는 DDL Audit 로그 파일 내용에 'DB Name'을 추가하여 다수의 데이터베이스를 운영 하는 환경에서 어느 데이터베이스에서 처리한 것인지 확인 가능하게 개선
- 커밋/롤백 없이 트랜잭션이 종료되어도 ABORT 로그를 ddl\_audit 로그에 기록하도록 수정
- SetAutocommit(false)에서 여러 DDL문을 한 번에 실행할 때 명령문 사이에 `commit` 또는 `rollback` 남기도록 개선

## REDO 복구 과정에 병렬 스레드 적용하여 성능 개선

- 데이터 정합성을 위해 동기화 적용이 필요하지 않는 REDO 복구 과정에 대해 병렬 스레드를 이용한 데이터 페이지별 복구를 적용하여 복구 성능 개선.
- 복구시 REDO의 비중이 크고 병렬 지수가 좋은 경우 성능 개선 효과가 큼.

## 교착 상태(deadlock) 발생 시 데이터베이스명\_latest.event 로그파일에 기록되는 잠금관련 정보를 간소화

- 교착 상태(Deadlock) 발생 시, 서버 로그 파일( `데이터베이스명_latest.event` )에 기록되는 로그를 개선 하여 교착 상태에 직접 기여한 잠금 정보만 출력 하도록 변경.

## time\_format() 및 date\_format()의 성능 개선

- 불필요하게 사용된 문자열 연산 함수 호출을 제거하여성능 개선.

## TRACE 활성화 시 쿼리 성능 개선

- `set trace on` 상태에서 쿼리를 실행할 때 발생하던 **TRACE** 통계 정보 수집의 성능 저하 문제를 해결 하기 위해 고비용 작업을 줄여, `set trace off` 상태와 동일한 성능으로 실행할 수 있도록 개선

문자열 유형에 대한 불필요한 길이 검사 제거를 통해 성능 개선

**lock\_escalation** 설정 값 만큼만 락 리소스를 유지하도록 변경해서 메모리 사용량 개선

**optimization\_level** 설정값에 잘못된 값을 입력 시 오류 출력을 통해 사용자 인지 개선

DB볼륨 및 로그의 생성 시간을 확인 개선

- `SHOW VOLUME HEADER`, `SHOW LOG HEADER`, `SHOW ARCHIVE LOG HEADER` 의 **Creation\_time**으로 볼륨 생성 시간 확인할 수 있게 개선
- `diagdb` 에서 각 볼륨의 생성 시간을 확인할 수 있게 개선
- `applyinfo` 에서 **활성 로그 볼륨** 과 **보관 로그 볼륨** 의 볼륨 생성 시간을 확인할 수 있게 개선

**SHOW INDEX CAPACITY** 및 `diagdb` 에 fence key 정보 추가를 통해 사용자 인지 개선

- 리프 노트 키 압축시 사용한 fence key에 대한 정보를 일반 key 정보와 분리해서 사용자 인지를 개선

오류수정

## SQL

다른 계정의 동일명 원격 서버 객체가 존재하는 경우, 사용자명(user schema)없이 원격 서버 객체 참조시 자신의 원격 서버 객체를 참조하지 못하는 오류 수정

ORDER BY 절이 있는 스칼라 하위 질의에서 hidden 컬럼 처리 오류 수정

ORDER BY를 포함하는 인라인 뷰와 스칼라 하위 질의의 뷰 머징 처리시 ROWNUM이 ORDER\_BY\_NUM() 으로 잘못 재 작성되는 오류 수정

oracle\_style\_empty\_string=yes 설정시 'NULL || 문자열' 연산 결과가 NULL로 출력되는 오류 수정

oracle\_style\_empty\_string=yes 설정시 REPLACE 함수 결과가 NULL로 출력되는 오류 수정

함수 인덱스를 기반으로 Covered Index 스캔 수행시 잘 못된 결과를 반환하는 오류 수정

항상 false (또는 NULL) 조건을 가진 CTE 질의 수행시 발생한 세그먼트 폴트 오류 수정

UNION ALL 구문에서 여러 테이블과 serial의 next\_value를 함께 사용한 질의 수행시 발생한 세그먼트 폴트 오류 수정

SELECT list 절에 alias 없이 255자 초과한 상수값을 사용한 질의 수행시 발생한 세그먼트 폴트 오류 수정

OUTER JOIN을 포함한 질의에서 뷰머징 처리 시 발생한 오류 수정



**SHOW CREATE VIEW** 질의 수행시 현재 사용자의 뷰와 public 사용자의 동일명 뷰가 함께 출력되는 오류 수정

repeat() 함수에 string\_max\_size\_bytes 설정값을 초과한 요청한 경우 오류 메시지가 출력되지 않고 NULL이 반환되는 오류 수정

쿼리 최적화 중 INNER JOIN 제거 과정에서 테이블 위치가 잘못 처리되는 오류 수정

PREPARE 구문에서 특정 내장 함수에 호스트 변수 사용시 발생한 오류 수정

**DROP USER** 수행시 관련 시스템 카탈로그 테이블의 사용자 정보가 삭제되지 않은 오류 수정

SP 호출 및 WHERE 절에 호스트 변수를 사용한 Oracle 스타일 LEFT OUTER JOIN이 INNER JOIN으로 재 작성되지 않는 오류 수정

**ORDERBY\_NUM()**이 포함된 하위 질의가 뷰 머징에 잘 못된 결과를 출력하는 오류 수정

질의 최적화 중 조인 관계가 없는 테이블이 조인 테이블로 인식되는 오류 수정

WHERE 절의 동등 조건(=)과 ORDER BY 절에서 사용된 컬럼이 동일할 때 ROWNUM 값이 0으로 출력되는 오류 수정

INNER JOIN과 Oracle 스타일 OUTER JOIN을 함께 사용하고, WHERE 절에 OUTER JOIN 조인 조건의 위치에 따라 발생한 오류 수정

**INSERT INTO tbl ... SELECT ... FROM 뷰테이블 ... ON DUPLICATE KEY UPDATE** 질의 수행시 잘못된 결과를 출력하는 오류 수정

뷰에 분석 함수가 포함될 때, 분석 함수 내부의 컬럼 순서 번호가 잘못되어 잘못된 결과를 출력하는 오류 수정

범위 조건을 파이프 연산자( || )와 함께 사용시 범위 항목으로 인지하지 못하는 오류 수정

PL(javasp)

JavaSP의 DATETIME 매개변수에 DATETIMELTZ 값을 시용시 발생한 오류 수정

JavaSP의 선언된 Java 메서드명과 여는 괄호 사이의 공백으로 인해 JavaSP 수행되지 않는 오류 수정

JavaSP의 미지원 타입을 보유한 테이블의 다른 컬럼을 인자로 사용시 발생한 오류 수정

JavaSP에서 **Cannot allocate query entry any more** 발생한 오류 수정

DBLINK 구문의 호스트 변수에 NULL 처리시 발생하는 오류 수정

EUC-KR 오라클 DB에 DBLINK Insert 구문에 호스트 변수를 통해 한글 입력시 한글 깨지는 오류 수정

## 유틸리티

`loaddb` 실행 중 오류 발생 시 종료 코드가 3으로 설정되지 않는 오류 수정

`loaddb -no-logging` 실행 후 잘 못된 **count(\*)** 값을 출력하는 오류 수정

`loaddb` 실행 중 스키마명이 없는 serial 처리시 발생한 오류 수정

`loaddb` 실행 중 오류 발생 시 멈추지 않고 계속 실행되는 오류 수정

`loaddb` 실행 중 대상이 시스템 테이블인 synonym 처리시 발생하는 오류 수정

11.4 이전 버전의 `unloaddb` 파일을 `loaddb` 실행 시, 트리거 시스템 테이블 컬럼 (condition, action\_definition)에 스키마명이 잘못 저장되거나 누락되는 오류 수정

`unloaddb` 실행 시 Reverse Unique Index의 주석이 누락되는 오류 수정

`unloaddb` 실행 시 auto\_increment 설정과 함께 PK로 선언된 컬럼의 auto\_increment 값이 1로 초기화되는 오류 수정

`unloaddb -split-schema-files` 실행 시 Unique Index가 없는 경우 'db명\_schema\_uk' 파일이 생성되는 오류 수정

`unloaddb -i (-input-class-only)` 옵션을 함께 실행시 특정 ALTER SERIAL 문이 누락되는 오류 수정

`unloaddb` 실행시 serial의 current\_val과 max\_val 동일한 경우 발생한 오류 수정

`unloaddb` 실행시 1MB 초과한 JSON 타입 데이터 처리에 대해 무한 루프 발생하는 오류 수정

일반 사용자로 `unloaddb` 실행시 뷰의 `query_spec` 출력시 일반 사용자와 동일한 스키마명이 제거되지 않는 오류 수정

**DBA** 사용자로 `unloaddb` 실행시 `serial`과 `trigger`의 스키마명이 누락되는 오류 수정

**DBA** 사용자가 `-as-dba` 옵션 없이 `unloaddb` 실행시 다른 사용자가 부여한 PROCEDURE 권한(`GRANT ... ON PROCEDURE`)이 출력되는 오류 수정

일반 사용자가 `unloaddb` 실행시, 트리거의 `condition`과 `action_definition`에 일반 사용지외 동일한 스키마명이 제거되지 않는 오류 수정

`backupdb` 실행시 불필요한 보관 로그 볼륨이 생성되는 오류 수정

FlashBack를 강제 종료 (Ctrl+C)시 작업 종료되지 않은 오류 수정

특정 상황에서 `backupdb` 수행시 hang 발생하는 오류 수정

Broker,CAS, CMS

`addBatch()`와 `executeBatch()` 처리 시 발생할 수 있는 메모리 누수(memory leak) 수정

`gettranscationinfo()` 함수 호출시 잘 못된 결과를 출력하는 오류 수정

## HA

Master, Slave 및 Replica 노드로 구성된 HA 환경에서 ha\_replica\_delay 설정값이 60 초 이상으로 설정된 경우, Replica 노드에서 slave 대상 복제 로그가 삭제되지 않는 오류 수정

applylogdb 수행중 생성된 sql log 파일이 특정 조건에서 자동 삭제되지 않는 오류 수정

## 주의사항

### 신규 주의 사항

CUBRID 11.4의 데이터베이스 볼륨은 CUBRID 11.3 및 그 이전 버전의 볼륨과 호환되지 않는다.

- 새로운 기능 추가로 인해 시스템 카탈로그 정보가 변경되거나 추가됨 (시스템 카탈로그). 그 외 신규 주의 사항은 [스펙변경](#) 사항을 참조.

### 기존 주의 사항

## 11.3

질의 최적화기의 통계 정보는 DDL 문이 수행될 때 자동으로 갱신되지 않고, 사용자가 직접 UPDATE STATISTICS 문을 실행하여 통계정보를 갱신해야 함 (조인 방식 참조)

ALTER TABLE 구문에서 AUTO\_INCREMENT 속성 갖는 컬럼 또는 default 값을 가지고 있는 컬럼의 타입 변경시, AUTO\_INCREMENT 속성을 사용할 수 없는 타입으로 변경하거나 기존 default 값을 변환할 수 없는 타입으로 변경할 때, 에러가 발생함 (CHANGE/MODIFY 절 참조)

서버측 JDBC에서 cubrid.jdbc.driver.CUBRIDDriver.getDefaultConnection()을 통해 Connection 객체를 가져올 수 없음. 대신, DriverManager.getConnection("jdbc:default:connection:");을 사용해야 함 (Connection 생성 참조)

USE INDEX(USING INDEX) 구문에서 유효하지 않은 인덱스명이나 테이블명을 지정하는 경우 에러 처리하지 않고 무시함 (해당 사항을 log 파일에 남기려면 error\_log\_warning 설정값을 yes로 설정하면 됨)

CREATE INDEX 생성시 COMMENT 절의 위치가 WITH 절 또는 INVISIBLE 절 뒤인 구문 마지막 위치로 변경됨 (인덱스 정의문 참조)

## 11.2

CUBRID 11.2의 데이터베이스 볼륨은 CUBRID 11.1 및 그 이전 버전의 볼륨과 호환되지 않는다.

User schema 개념 도입으로 사용자 별로 동일한 객체명을 사용할 수 있으며, 지원에 따라 다음 동작이 변경됨

- 객체명에 "."(dot)을 허용하지 않는다.

- 질의 또는 유틸리티 명령어 사용 시 “[사용자명].객체명”으로 사용해야 한다. (단, 로그인 된 사용자의 객체를 질의하는 경우에는 사용자명을 생략할 수 있음) ([사용자 스키마](#) 참조)
- info schema, show full tables 결과에 사용자명 포함되도록 변경되었다. ([SHOW](#) 참조)
- 11.2 이전 loaddb 파일은 11.2에서 수행할 수 있도록 user명.table명으로 수정하거나 --no-user-specified-name 옵션을 설정하여 loaddb를 수행할 수 있다. ([loaddb](#) 참조)

### JavaSP의 “jdbc:default:connection:” 또는 getDefaultConnection() 사용 시 다음 함수 및 동작 변경됨 (JavaSP 주의 사항 참조)

- java.sql.DatabaseMetaData의 모든 function 지원하지 않는다.
- java.sql.Connection의 createClob(), createBlob() 지원하지 않는다.
- java.sql.Statement의 addBatch(), clearBatch(), executeBatch(), setMaxRows(), cancel() 지원하지 않는다.
- 하나의 prepare (또는 execute)에 multiple sql을 지원하지 않는다.
- cursor는 non-holdable로 변경되었다.
- ResultSet은 non-updatable, non-scrollable, non-sensitive로 변경되었다.

### Truncate table 수행 시 FK의 set null 또는 cascade가 있는 경우 동작이 변경됨 (TRUNCATE 참조)

Alter change/modify 시 기입하지 않은 컬럼 속성은 유지되게 변경되었으며, alter 구문으로 auto\_increment, on update property를 제거할 수 없게 변경됨 (CHANGE/MODIFY 절 참조)

### 조건절에 컬럼명만 존재하는 경우 오류 처리되도록 변경됨

- UPDATE t1 SET c1 = 9 WHERE c1; 형태로 사용하면 오류 발생한다.

Multiple SQL는 반드시 세미콜론으로 구분하도록 변경됨

CUBRID 패키지내의 CCI Driver 디렉토리가 \$CUBRID/lib, \$CUBRID/include에서 \$CUBRID/cci/lib, \$CUBRID/cci/include로 각각 변경됨 (CCI 개요 참조)

- CCI 사용 시 환경변수 LD\_LIBRARY\_PATH에 \$CUBRID/cci/lib를 추가해야 한다.

백업 시 압축(-z, --compress) 옵션이 기본으로 설정되도록 변경됨 (backupdb 참조)

새로운 기능 추가로 인해 시스템 카탈로그 정보가 변경되거나 추가됨 (시스템 카탈로그 참조)

## 11.0

CUBRID 11.0의 데이터베이스 볼륨은 CUBRID 10.2 및 그 이전 버전의 볼륨과 호환되지 않는다.

옵션 없이 테이블 생성 시 reuse\_oid 테이블로 생성된다.

CHAR 데이터 타입의 최대 길이는 256M 문자열로 변경되었다.

문자열 데이터 타입에 길이보다 큰 문자열이 들어가는 경우 오류가 발생하게 수정되었다.

문자열 끝의 공백 문자를 인식하게 수정하여, 문자열 끝의 공백 문자에 따라 다른 문자열로 인식된다.

통계 수집 방식의 변경으로 주기적인 통계 수집 수행이 필요하다.



## 10.2

CUBRID 10.2의 데이터베이스 볼륨은 CUBRID 10.1 및 그 이전 버전의 볼륨과 호환되지 않는다.

## 10.1

CUBRID 10.1의 데이터베이스 볼륨은 CUBRID 10.0 및 그 이전 버전의 볼륨과 호환되지 않는다.

## 10.0 및 이하 버전

CUBRID 10.0의 데이터베이스 볼륨은 CUBRID 9.3 및 그 이전 버전의 볼륨과 호환되지 않는다.

DB를 생성할 때 로케일(언어 및 문자 집합)을 지정

- DB를 생성할 때 로케일을 지정하도록 변경되었다.

CUBRID\_CHARSET 환경 변수 제거

- 9.2 버전 이후 DB를 생성할 때 로케일(언어 및 문자 집합)을 지정하므로 CUBRID\_CHARSET는 더 이상 사용하지 않는다.

[JDBC] 연결 URL의 zeroDateTimeBehavior 값이 “round”일 때 TIMESTAMP의 zero date가 ‘0001-01-01 00:00:00’에서 ‘1970-01-01 00:00:00’(GST)으로 변경 (CUBRIDSUS-11612)

- 2008 R4.4부터 연결 URL의 “zeroDateTimeBehavior” 속성 값이 “round”일 때 TIMESTAMP의 zero date가 ‘0001-01-01 00:00:00’에서 ‘1970-01-01 00:00:00’(GST)으로 변경되었으므로, 응용 프로그램에서 zero date를 사용하는 경우 주의해야 한다.

AIX에서 CUBRID SH 패키지 설치 시 권장 사항 (CUBRIDSUS-12251)

- AIX OS에서 ksh를 사용하여 CUBRID SH 패키지를 설치하는 경우 다음 오류와 함께 실패한다.

0403-065 An incomplete **or** invalid multibyte character encountered.

- 따라서 ksh 대신 ksh93 또는 bash를 사용할 것을 권장한다.

```
$ ksh93 ./CUBRID-9.2.0.0146-AIX-ppc64.sh
$ bash ./CUBRID-9.2.0.0146-AIX-ppc64.sh
```

## CUBRID\_LANG 제거, CUBRID\_MSG\_LANG 추가

- 9.1 버전부터 CUBRID\_LANG 환경 변수를 더 이상 사용하지 않는다.
- 유틸리티 메시지 및 오류 메시지를 출력할 때는 CUBRID\_MSG\_LANG 환경 변수를 사용한다.

## CCI 응용 프로그램에서 여러 개의 질의를 한 번에 수행한 결과의 배열에 대한 오류 처리 방식 수정 (CUBRIDSUS-9364)

- CCI 응용에서 여러 개의 질의를 한 번에 수행할 때 2008 R3.0부터 2008 R4.1 버전까지는 cci\_execute\_array 함수나 cci\_execute\_batch 함수에 의한 질의 수행 결과들 중 하나만 오류가 발생해도 해당 질의의 오류 코드를 반환했으나, 2008 R4.3 버전 및 9.1 버전부터는 전체 질의 개수를 반환하고 CCI\_QUERY\_RESULT\_\* 매크로를 통해 개별 질의에 대한 오류를 확인할 수 있도록 수정했다.
- 수정 이전 버전에서는 오류가 발생한 경우에도 배열 내 각각의 질의들의 성공 실패 여부를 알 수 없으므로, 이를 판단해야 한다.

```
...
char *query = "INSERT INTO test_data (id, ndata, cdata, sdata,
ldata) VALUES (?, ?, 'A', 'ABCD', 1234)";
...
req = cci_prepare (con, query, 0, &cci_error);
...
error = cci_bind_param_array_size (req, 3);
...
error = cci_bind_param_array (req, 1, CCI_A_TYPE_INT, co_ex,
null_ind, CCI_U_TYPE_INT);
...
n_executed = cci_execute_array (req, &result, &cci_error);

if (n_executed < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
```

```

 for (i = 1; i <= 3; i++)
 {
 printf ("query %d\n", i);
 printf ("result count = %d\n", CCI_QUERY_RESULT_RESULT
(result, i));
 printf ("error message = %s\n", CCI_QUERY_RESULT_ERR_MSG
(result, i));
 printf ("statement type = %d\n", CCI_QUERY_RESULT_STMT_TYPE
(result, i));
 }
}
...

```

- 수정된 버전부터는 오류가 발생하면 전체 질의가 실패한 것이며, 오류가 발생하지 않은 경우에 대해 배열 내 각 질의의 성공 여부를 판단한다.

```

...
char *query = "INSERT INTO test_data (id, ndata, cdata, sdata,
ldata) VALUES (?, ?, 'A', 'ABCD', 1234)";
...
req = cci_prepare (con, query, 0, &cci_error);
...
error = cci_bind_param_array_size (req, 3);
...
error = cci_bind_param_array (req, 1, CCI_A_TYPE_INT, co_ex,
null_ind, CCI_U_TYPE_INT);
...
n_executed = cci_execute_array (req, &result, &cci_error);
if (n_executed < 0)
{
 printf ("execute error: %d, %s\n", cci_error.err_code,
cci_error.err_msg);
}
else
{
 for (i = 1; i <= 3; i++)
 {
 printf ("query %d\n", i);
 printf ("result count = %d\n", CCI_QUERY_RESULT_RESULT
(result, i));
 printf ("error message = %s\n", CCI_QUERY_RESULT_ERR_MSG
(result, i));
 printf ("statement type = %d\n", CCI_QUERY_RESULT_STMT_TYPE
(result, i));
 }
}
...

```

## java.sql.XAConnection 인터페이스에서 HOLD\_CURSORS\_OVER\_COMMIT을 지원하지 않음 (CUBRIDSUS-10800)

- 현재 CUBRID는 java.sql.XAConnection 인터페이스에서 ResultSet.HOLD\_CURSORS\_OVER\_COMMIT를 지원하지 않는다.

## 9.0부터 STRCMP가 대소문자를 구분하여 동작

- 9.0 이전 버전까지는 STRCMP가 대소문자를 구분하지 않았지만 9.0부터는 문자열에서 대소문자를 비교하여 구분한다.
- STRCMP가 대소문자를 구분하지 않도록 하려면 대소문자를 구분하지 않는 콜레이션(예: utf8\_en\_ci)을 사용해야 한다.

```
-- In previous version of 9.0 STRCMP works case-insensitively
SELECT STRCMP ('ABC','abc');
0

-- From 9.0 version, STRCMP distinguish the uppercase and the
lowercase when the collation is case-sensitive.
export CUBRID_CHARSET=en_US.iso88591

SELECT STRCMP ('ABC','abc');
-1

-- If the collation is case-insensitive, it distinguish the
uppercase and the lowercase.
export CUBRID_CHARSET=en_US.iso88591

SELECT STRCMP ('ABC' COLLATE utf8_en_ci , 'abc' COLLATE utf8_en_ci);
0
```

## 2008 R4.1 버전부터 CCI\_DEFAULT\_AUTOCOMMIT의 기본값이 ON으로 변경됨 (CUBRIDSUS-5879)

- CCI 인터페이스로 개발한 응용 프로그램의 자동 커밋 모드에 영향을 미치는 CCI\_DEFAULT\_AUTOCOMMIT 브로커 파라미터의 기본값이 CUBRID 2008 R4.1부터 ON으로 변경되었다. 이 변경의 결과로 CCI 및 CC 기반 인터페이스(PHP, ODBC, OLE DB 등) 사용자는 응용 프로그램의 자동 커밋 모드가 이에 대해 적합한지 확인해야 한다.

## 2008 R4.0 버전부터 페이지 단위를 사용하는 옵션 및 파라미터가 볼륨 크기 단위를 사용하도록 변경됨 (CUBRIDSUS-5136)

- 데이터베이스 볼륨 크기와 cubrid createdb 유틸리티의 로그 볼륨 크기를 지정하기 위해 페이지 단위를 사용하는 옵션(-p, -l, -s)은 제거되므로, 2008 R4.0 베타 이후 새로 추가된 옵션(-db-volume-size, -log-volume-size, -db-page-size, -log-page-size)을 사용한다.
- cubrid addvoldb 유틸리티의 데이터베이스 볼륨 크기를 지정하려면 페이지 단위를 사용하지 말고 2008 R4.0 베타 이후 새로 추가된 옵션(-db-volume-size)을 사용한다.
- 페이지 단위 시스템 파라미터가 제거되므로 바이트 형식의 새 시스템 파라미터 사용을 권장한다. 관련 시스템 파라미터에 대한 자세한 내용은 아래를 참고한다.

## 2008 R4.0 베타 이전 버전 사용자는 db 볼륨 크기를 설정할 때 주의할 것 (CUBRIDSUS-4222)

- 2008 R4.0 베타 버전부터 데이터베이스를 생성할 때 데이터 페이지 크기 및 로그 페이지 크기의 기본값이 4KB에서 16KB로 변경되었다. 페이지 수로 데이터베이스 볼륨을 지정하는 경우 볼륨의 바이트 크기는 예상과 다를 수 있다. 어떠한 옵션도 선택하지 않은 경우 이전 버전에서는 4KB 페이지 크기의 100MB 데이터베이스 볼륨이 생성되었다. 그러나 2008 R4.0부터는 16KB 페이지 크기의 512MB 데이터베이스 볼륨이 생성된다.

또한 사용 가능한 데이터베이스 볼륨의 최소 크기는 20MB로 제한된다. 따라서, 이 크기보다 작은 데이터베이스 볼륨을 생성할 수 없다.

## 2008 R4.0 이전 버전의 일부 시스템 파라미터의 기본값 변경 (CUBRIDSUS-4095)

- 2008 R4.0부터 일부 시스템 파라미터의 기본값이 변경되었다.
- max\_clients의 기본값(DB 서버에서 허용되는 동시 연결 수 지정)과 index\_unfill\_factor의 기본값(인덱스 페이지 생성 시 향후 갱신을 위한 예약 공간의 비율 지정)이 변경되었으며, 바이트 단위의 시스템 파라미터의 기본값이 페이지 단위의 이전 시스템 파라미터의 기본값을 초과하는 경우 더 많은 메모리를 사용하게 되었다.

| 기존 시스템 파라미터         | 추가된 시스템 파라미터 | 기존 기본값 | 변경된 기본값 (단위 : 바이트) |
|---------------------|--------------|--------|--------------------|
| max_clients         | 없음           | 50     | 100                |
| index_unfill_factor | 없음           | 0.2    | 0.05               |

| 기존 시스템 파라미터                 | 추가된 시스템 파라미터               | 기존 기본값             | 변경된 기본값 (단위 : 바이트) |
|-----------------------------|----------------------------|--------------------|--------------------|
| data_buffer_pages           | data_buffer_size           | 100M(page size=4K) | 512M               |
| log_buffer_pages            | log_buffer_size            | 200K(page size=4K) | 4M                 |
| sort_buffer_pages           | sort_buffer_size           | 64K(page size=4K)  | 2M                 |
| index_scan_oid_buffer_pages | index_scan_oid_buffer_size | 16K(page size=4K)  | 64K                |

- 또한, cubrid createdb를 사용하여 데이터베이스를 생성할 때 데이터 페이지 크기 및 로그 페이지 크기의 최소 값이 1K에서 4K로 변경되었다.

### 시스템 파라미터가 잘못 설정된 경우 데이터베이스 서비스, 유틸리티 및 응용 프로그램을 수행할 수 없도록 변경됨 (CUBRIDSUS-5375)

- cubrid.conf 또는 cubrid\_ha.conf에 정의되지 않은 시스템 파라미터를 설정하거나, 시스템 파라미터의 값이 임계값을 초과하거나, 페이지 단위 시스템 파라미터 및 바이트 단위 시스템 파라미터가 동시에 사용되는 경우 관련 서비스, 유틸리티 및 응용 프로그램이 수행되지 않도록 변경되었다.

### CUBRID 32비트 버전에서 2G를 초과하는 값을 사용하여 data\_buffer\_size를 설정할 경우 데이터베이스 구동에 실패함 (CUBRIDSUS-5349)

- CUBRID 32비트 버전에서 data\_buffer\_size의 값이 2G를 초과하는 경우 데이터베이스 구동에 실패한다. 이 설정 값은 OS 제한 때문에 32비트 버전에서 2G를 초과할 수 없다.

### Windows Vista 및 그 이후 버전에서 CUBRID 유틸리티를 사용한 서비스 제어 시 권장 사항 (CUBRIDSUS-4186)

- Windows Vista 및 그 이후 버전에서 cubrid 유틸리티를 사용하여 서비스를 제어하려면 명령 프롬프트 창을 관리자 권한으로 구동한 후 사용하는 것을 권장한다.
- 명령 프롬프트 창을 관리자 권한으로 구동하지 않고 cubrid 유틸리티를 사용하는 경우 UAC(User Account Control) 대화 상자를 통하여 관리자 권한으로 수행할 수 있으나 수행 결과의 메시지를 확인할 수 없다.

- Windows Vista 및 그 이후 버전에서 명령 프롬프트 창을 관리자 권한으로 구동하는 방법은 다음과 같다.:
  - [시작 > 모든 프로그램 > 보조프로그램 > 명령 프롬프트]를 마우스 오른쪽 버튼을 클릭한다.
  - [관리자로 수행(A)]을 선택하면 권한 상승을 확인하는 대화 상자가 활성화되고, “예”를 클릭하여 관리자 권한으로 구동한다.

## CUBRID 소스 빌드 후 수행 시, 매니저 서버 프로세스 관련 오류 발생 (CUBRIDSUS-3553)

- 사용자가 CUBRID 소스를 직접 빌드하고 설치하는 경우, CUBRID와 CUBRID Manager를 각각 빌드하여 설치해야 한다. CUBRID 소스만 체크 아웃하고 빌드 후 cubrid service start 또는 cubrid manager start를 실행하면 “cubrid manager server is not installed”라는 오류가 발생한다.

## 2008 r3.0 또는 그 이전 버전에서 사용하던 GLO 클래스 지원 중단 (CUBRIDSUS-3826)

- CUBRID 2008 R3.0 및 그 이전 버전은 glo(Generalized Large Object) 클래스를 사용하여 Large Object를 처리했지만 glo 클래스는 CUBRID 2008 R3.1 및 그 이후 버전에서 제거되었다. 대신, BLOB 및 CLOB(이후 LOB) 데이터 타입이 지원된다. LOB 데이터 타입에 대한 자세한 내용은 [BLOB/CLOB 데이터 타입 절](#)을 참고한다.
- glo 클래스 사용자는 다음 작업을 수행할 것을 권장한다.:
  - GLO 데이터를 파일로 저장한 후에 다른 응용 프로그램 및 DB 스키마에서 GLO를 사용하지 않도록 수정한다.
  - unloaddb 및 loaddb 유틸리티를 사용하여 DB 마이그레이션을 수행한다.
  - 수정된 응용 프로그램에 따라 파일을 LOB 데이터로 로드하는 작업을 수행한다.
  - 수정한 응용 프로그램이 정상적으로 동작하는지 확인한다.
- 예를 들어, cubrid loaddb 유틸리티가 GLO 클래스를 상속하거나 GLO 데이터 타입이 있는 테이블을 로드하는 경우 “Error occurred during schema loading.” 오류 메시지와 함께 데이터 로딩을 중지한다.
- GLO 클래스의 지원이 중단됨에 따라 각 인터페이스에 대해 삭제된 함수는 다음과 같다.:

| 인터페이스 | 삭제된 함수              |
|-------|---------------------|
| CCI   | cci_glo_append_data |

| 인터페이스 | 삭제한 함수                     |
|-------|----------------------------|
|       | cci_glo_compress_data      |
|       | cci_glo_data_size          |
|       | cci_glo_delete_data        |
|       | cci_glo_destroy_data       |
|       | cci_glo_insert_data        |
|       | cci_glo_load               |
|       | cci_glo_new                |
|       | cci_glo_read_data          |
|       | cci_glo_save               |
|       | cci_glo_truncate_data      |
|       | cci_glo_write_data         |
| JDBC  | CUBRIDConnection.getNewGLO |
|       | CUBRIDOID.loadGLO          |
|       | CUBRIDOID.saveGLO          |
| PHP   | cubrid_new_glo             |
|       | cubrid_save_to_glo         |
|       | cubrid_load_from_glo       |
|       | cubrid_send_glo            |

마스터와 서버 프로세스 사이의 프로토콜이 변경된 경우 또는 두 버전이 동시에 실행되는 경우 포트 설정 필요 (CUBRIDSUS-3564)

- 마스터 프로세스(cub\_master)와 서버 프로세스(cub\_server) 사이의 통신 프로토콜이 변경되었으므로 CUBRID 2008 R3.0 또는 그 이후 버전의 마스터 프로세스는 이전 버전의 서버 프로세스와 통신할 수 없고 이전 버전의 마스터 프로세스는 2008 R3.0 또는 그 이후 버전과 통신할 수 없다. 따라



서 이전 버전이 이미 설치된 환경에 새 버전을 추가하여 동시에 두 버전의 CUBRID를 실행하는 경우 버전별로 다른 포트가 사용되도록 cubrid.conf의 cubrid\_port\_id 시스템 파라미터를 수정해야 한다.

## JDBC에서 URL 문자열로서 연결 정보를 입력할 때 물음표 명시 (CUBRIDSUS-3217)

- JDBC에서 URL 문자열로서 연결 정보를 입력할 때 이전 버전에서 물음표(?)를 입력하지 않은 경우에도 속성 정보가 지정되었다. 그러나 이 CUBRID 2008 R3.0 버전에서는 구문에 따라 물음표를 명시해야 한다. 그렇지 않은 경우 오류가 표시된다. 또한, 연결 정보에 사용자명 또는 암호가 없는 경우에도 콜론(:)을 명시해야 한다.

```
URL=jdbc:CUBRID:
127.0.0.1:31000:db1:::altHosts=127.0.0.2:31000,127.0.0.3:31000 -- 에러처리
URL=jdbc:CUBRID:127.0.0.1:31000:db1:::?
altHosts=127.0.0.2:31000,127.0.0.3:31000 -- 정상처리
```

## 데이터베이스명에 @을 포함할 수 없음 (CUBRIDSUS-2828)

- 데이터베이스명에 @이 포함되면 호스트명이 지정된 것으로 해석될 수 있다. 이를 방지하기 위해 cubrid createdb, cubrid renamedb 및 cubrid copydb 유틸리티를 실행할 때는 데이터베이스에 @이 포함될 수 없도록 수정되었다.

## 공통 정보

### 개정 내역

| 작성 날짜    | 설명                       |
|----------|--------------------------|
| 2025년 3월 | CUBRID 11.4 릴리스 (11.4.0) |

## 버그 리포트 및 사용자 피드백 제공 방법

CUBRID 프로젝트에서는 사용자의 거침없는 버그 리포트와 솔직한 피드백을 기다리고 있으며, 아래 사이트에서 등록할 수 있다.

| 문서      | 설명                                                                                                                                                                               |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 버그 리포트  | CUBRID Issue Tracker: <a href="http://jira.cubrid.org/browse">http://jira.cubrid.org/browse</a>                                                                                  |
| 사용자 피드백 | CUBRID 오픈 소스 프로젝트: <a href="https://github.com/CUBRID/cubrid">https://github.com/CUBRID/cubrid</a><br>CUBRID 공식 사이트: <a href="https://www.cubrid.org">https://www.cubrid.org</a> |

## 라이선스

CUBRID의 서버 엔진에는 Apache 라이선스 2.0 이 적용되고 CUBRID 매니저 및 인터페이스(API)에는 BSD 라이선스가 적용된다. 보다 상세한 정보는 CUBRID 공식 사이트의 라이선스 가이드( <https://www.cubrid.org/license>) 를 참고한다.

## 추가 정보

업그레이드 및 마이그레이션과 관련된 정보는 [업그레이드](#)를 참고한다.

CUBRID 최신 소스는 <https://github.com/CUBRID/cubrid> 와 <https://github.com/CUBRID> 를 참고한다.

## 드라이버 관련 주의 사항

현재 CUBRID가 지원하는 드라이버들은 JDBC, Node.js, CCI(CUBRID C API), PHP, PDO, Python, Perl, Ruby, ADO.NET, ODBC, OLE DB이다. JDBC와 Node.js, ADO.NET을 제외한 모든 드라이버들은 CCI 기반으로 개발되었으므로 CCI와 관련하여 변경된 사항은 CCI 기반 드라이버들에 영향을 미칠 수 있다.

# 인덱스

· 색인

