CUBRID 정기교육

2022-05

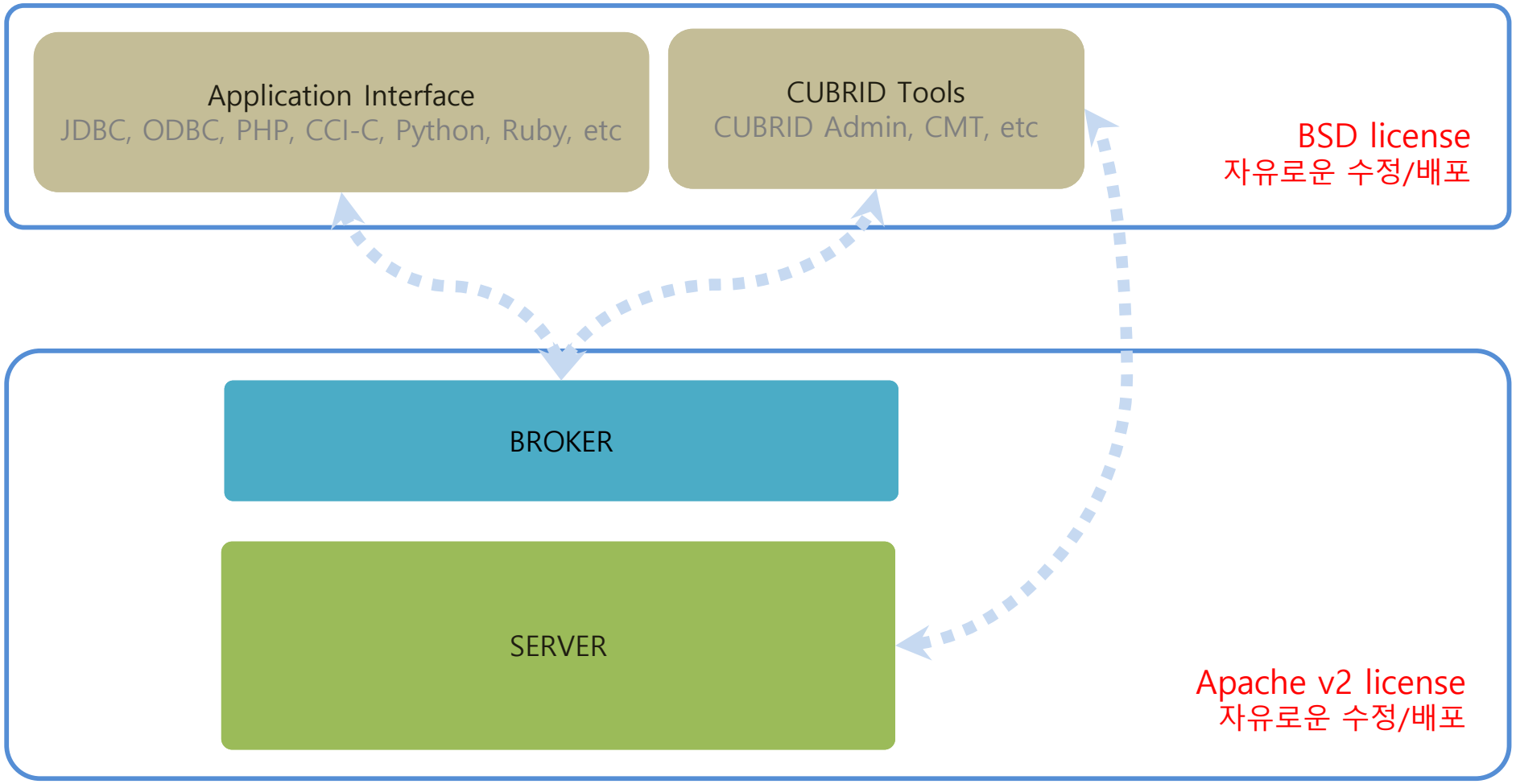
목차

CUBRID

1. CUBRID 소개

오픈소스 라이선스

- 이중 라이선스 채택



CUBRID 주요 기능

RDBMS 기본 기능	• 트랜잭션 완벽 보장(commit/rollback/savepoint) • 장애 발생/백업 복구시 트랜잭션 일치 보장 • HA 환경에서 트랜잭션 일치 보장 • ANSI SQL 표준 및 확장된 SQL 지원 • 계층형 쿼리, CTE (Common Table Expression)를 이용한 재귀적 쿼리 등 • VIEW/TRIGGER/PRIMARY KEY/FOREIGN KEY/SERIAL 등 지원
고성능	• MVCC (Multi-version Concurrency Control) 지원 • 브로커 미들웨어에 의한 connection pooling/load balancing/Proxy 기능 지원 • 비용 기반 옵티마이저 지원(CBO) • 질의 계획 및 결과 캐쉬 지원 • 고성능 인덱스(Multi-Range/Covered/Reverse/Skip-Scan/Function based/Filtered Index) 지원
대용량 및 확장성	• 멀티 볼륨 및 볼륨 자동 추가 기능 지원 • DB/테이블/컬럼/인덱스 개수 무제한 생성 가능 • 1:N 복제 구성을 통해 부하 분산 및 서비스 확장 가능 • 테이블 분할(Partitioning)을 통한 데이터 분할 관리 기능

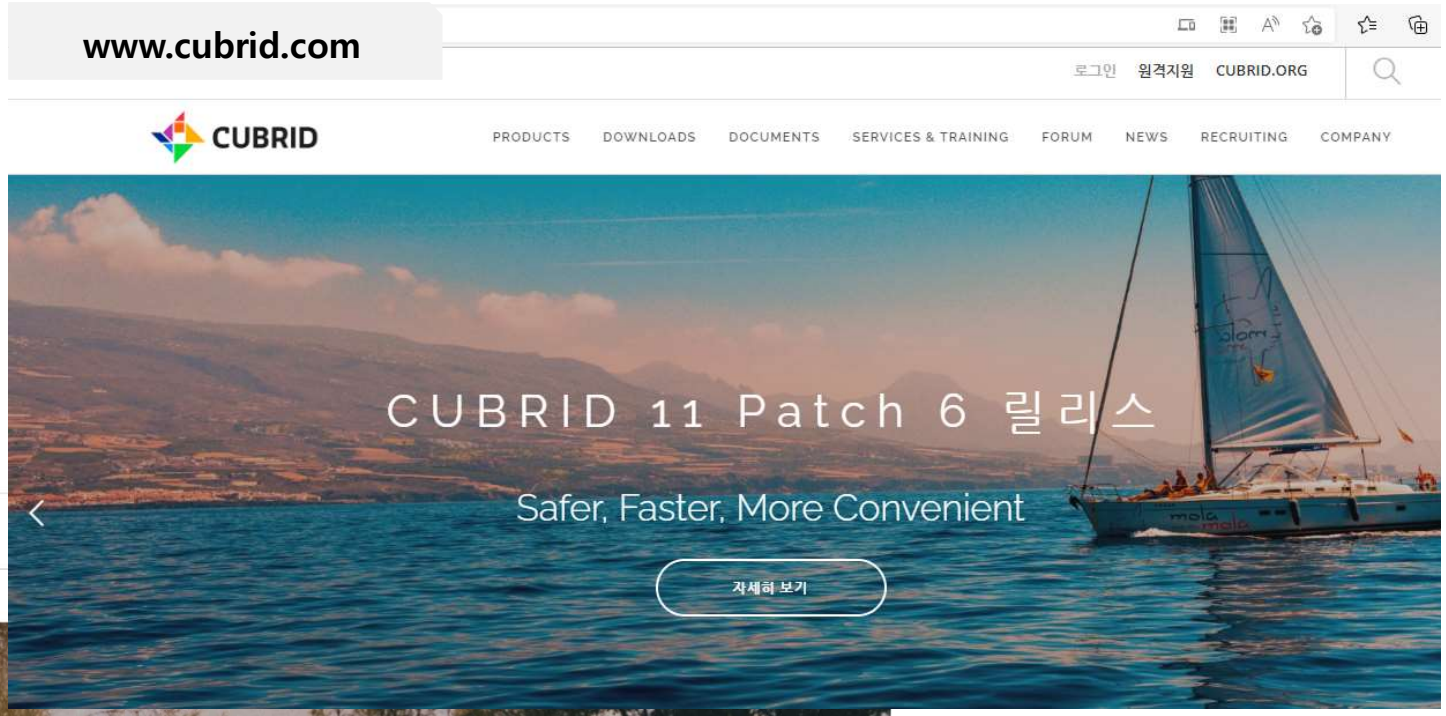
CUBRID 주요 기능 - 계속

안정성 및 운영 편의성	• 온라인/오프라인 백업 지원 • 증분 백업 및 병렬/압축 백업 지원 • 장애 발생 시점 또는 특정 시점으로의 복구 지원 • 권한 상속을 통한 사용자/그룹별 권한 관리 기능 • HA 환경에서 장애 발생 시 자동 절체(Auto-Failover) 지원
보안성	• 테이블 기반의 TDE(Transparent Data Encryption) 지원 • 드라이버와 서버 간의 패킷 암호화 지원 • 승인된 사용자/IP 접근 제어 지원

CUBRID

CUBRID 주요 웹사이트

www.cubrid.com

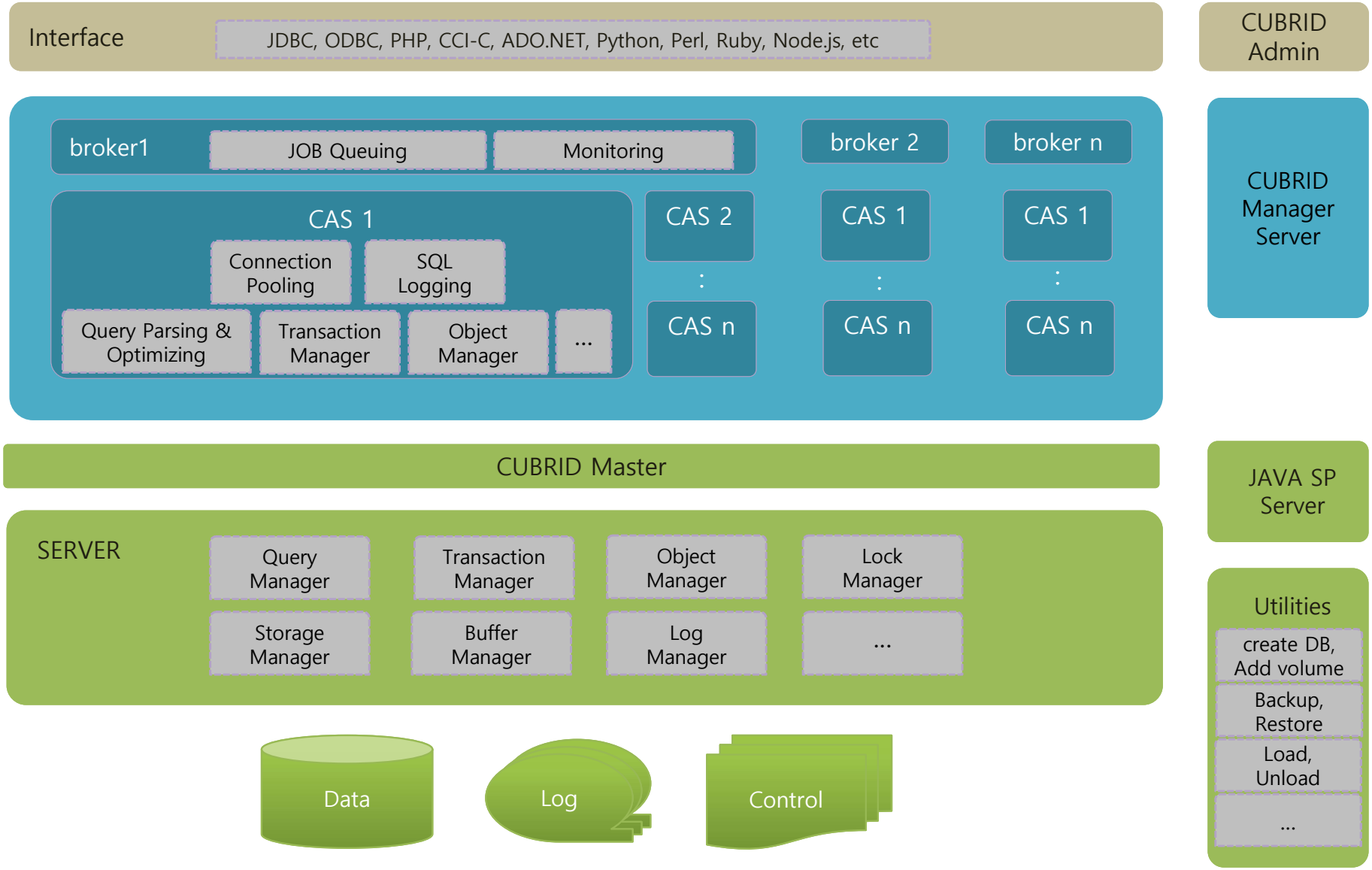


www.cubrid.org

CUBRID

2. CUBRID 시스템 구조

CUBRID 시스템 구조도



CUBRID

CUBRID 시스템 구조

- 데이터베이스 서버
  - 데이터의 저장 및 관리를 수행한다.
  - Multi thread기반 server/client 방식으로 동작한다.
  - 사용자가 입력한 질의를 처리하고 DB 내의 객체 관리한다.
  - 잠금(LOCK)과 로깅(Logging) 기법을 이용해 다수 사용자의 동시 트랜잭션을 지원한다.
  - 운영에 필요한 백업 및 복구 기능을 지원 한다.

- 브로커
  - CUBRID 전용 미들웨어로 외부 응용 프로그램의 요청을 데이터베이스 서버에 전달하고, 그 결과를 응용 프로그램에 전달하는 역할을 한다.
  - 브로커는 커넥션 풀링, 모니터링, 로그 추적 및 분석 기능 제공한다.

- JAVA SP 서버
  - JAVA 로 개발된 Stored Procedure/Function 을 실행시켜주는 JVM 이다.
    - JAVA class 또는 jar 를 데이터베이스에 등록하여야 한다.

CUBRID 시스템 구조 - 계속

- Interface
  - 외부 응용 프로그램을 위한 다양한 드라이버 제공
    - JDBC, ODBC, PHP, CCI-C, ADO.NET, Python, Perl, Ruby, Node.js 등
    - JDBC($CUBRID/jdbc/cubrid_jdbc.jar)와 CCI-C($CUBRID/incude/cas_cci.h, $CUBRID/lib/libcascci.so) 는 설치시 기본 제공되며, 나머지 드라이버는 다운로드(ftp.cubrid.org/CUBRID_Drivers/) 받아 사용한다.
  - 모든 드라이버는 브로커를 통해 데이터베이스에 접속한다.

- CUBRID Admin
  - 데이터베이스 서버와 브로커를 관리할 수 있는 GUI 기반의 관리자 도구이다.
  - CUBRID 설치 시 CUBRID Manager Server 는 기본적으로 같이 설치된다.
  - CUBRID Admin 은 다운로드(ftp.cubrid.org/CUBRID_Tools/CUBRID_Admin/) 받아 사용한다.

CUBRID 프로세스

- CUBRID 설치 후 서비스를 위한 구동이 완료되면 프로세스를 확인 할 수 있다.
  - CUBRID 관련 프로세스는 모두 cub_ 라는 prefix 를 가지고 있다.
    - 구동된 CUBRID 프로세스 확인 : ps –ef | grep cub_
  - CUBRID Service 구동시 기본적으로 Master, Broker, CAS, Manager Server 프로세스가 구동된다.

- Master Process (cub_master)
  - CUBRID 서비스 구동시 서버당 하나의 프로세스가 구동된다.
  - DB Client와 DB Server 프로세스 사이의 연결을 담당한다.
    - DB Client : CAS, csql 등
  - HA 환경에서 Master/Slave 간 상호 동작 상태를 확인한다.

- Database Server Process (cub_server)
  - 데이터베이스별 하나의 프로세스가 구동된다.
  - 여러 개의 데이터베이스를 사용하는 경우 그 개수만큼의 프로세스가 구동된다.
  - 데이터베이스 파일 및 로그 파일 등에 직접 접근하여 사용자 요청을 처리한다.

- 실행 모드
  - 클라이언트/서버 모드(Client-Server mode/CS mode)
    - Database Server 프로세스가 구동되어 있는 상태이며, 이때 모든 클라이언트 프로세스는 CS(클라이언트/서버) 모드로 데이터베이스에 접속이 가능하다.
  - 독립 모드(Stand-Alone mode/SA mode)
    - 독립 모드는 CSQL 등의 하나의 프로세스가 독립적으로 데이터베이스를 사용하는 것으로, 다른 프로세스는 데이터베이스에 접속할 수 없다.

CUBRID

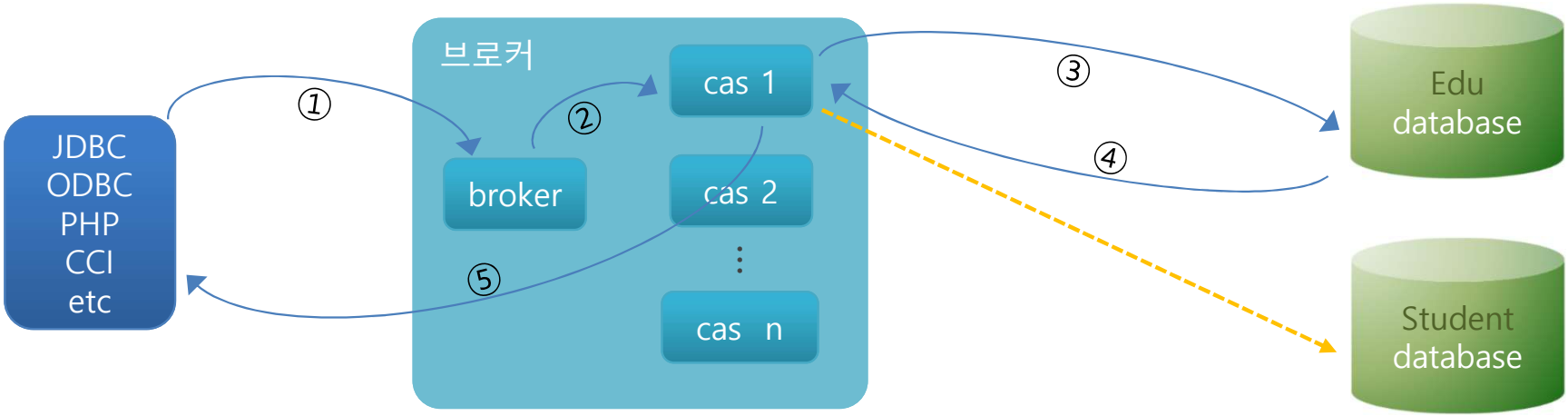
CUBRID 프로세스 - 계속

- BROKER Process (cub_broker)
  - 브로커 구동시 구동되며, 기본적으로 CUBRID 서비스 구동시 같이 구동된다.
    - Service 상태가 ON인 브로커만 구동된다.
  - 응용(JDBC, ODBC, PHP 등)과 cas 프로세스 사이의 연결을 중계하는 기능을 수행한다.
  - 브로커는 cas 프로세스의 상태를 관리하고 모니터링한다.

- CAS Process (cub_cas)
  - 브로커 구동시 브로커가 구동시킨다.
    - 만약 broker 없이 cas 만 구동되어있다면 비정상적인 프로세스이므로 종료(kill) 시켜야 한다.
  - DB에 연결하고자 하는 모든 종류의 응용이 사용하는 공용 응용 서버의 역할을 한다.
  - 응용 접속량에 따라 cub_broker 프로세스에 의해 cas 프로세스는 동적으로 조정된다.
    - cas 프로세스의 개수는 설정 값을 따른다.
  - SQL 분석이나 최적화, 실행 계획 생성 등의 작업이 수행된다.

- Manager Server Process (cub_manager)
  - 매니저 서버 구동시 구동되며, 기본적으로 CUBRID 서비스 구동시 이 구동된다.
  - 접속 기본 포트는 8001이며, 필요에 따라 설정을 통해 변경할 수 있다.
    - CUBRID 9.3.1 까지는 8001, 8002 2개의 포트를 사용한다.
  - 매니저 프로세스는 서비스 운영에 직접적인 영향이 없으므로, 필요 시 종료나 재구동이 가능하다.

CUBRID 요청 처리 과정



1. 응용(JAVA,PHP 등)에서 작업(질의) 처리를 broker로 요청
2. broker 는 관리하고 있는 cas프로세스에게 작업을 할당
   - 질의 처리는 cas가 수행(프로세스 단위로 작업 수행)
   - 응용은 cas에 직접 요청하는 것이 아니고, broker 에게 작업을 요청해 cas를 할당 받음
3. cas는 데이터베이스에 연결하고 질의처리를 요청
   1. 응용에서 connection URL 정보로 데이터베이스를 지정하며, cas는 지정된 데이터베이스로 연결
   2. cas는 작업 후 연결을 유지(disconnect 요청을 받아도), 동일한 데이터베이스로 요청이 들어오면 기존 연결을 재사용
   3. cas는 연결유지 상태에서 다른 데이터베이스로의 요청이 들어오면 현재 연결을 끊고 새로 연결
   4. 데이터베이스 별로 broker를 할당해서 사용하는 것이 연결유지관리와 성능에 유리
4. 데이터베이스는 cas로부터의 요청을 처리하고 결과를 해당 cas에게 전달
5. 데이터베이스의 작업결과(데이터)를 응용에 전달

※ cas와 트랜잭션

트랜잭션 처리 중에는 하나의 CAS가 하나의 응용에 종속되어 현재 처리중인 요청이 없더라도 다른 요청은 처리하지 않음

- ✓ cas프로세서 구동 수 만큼 동시 트랜잭션 처리
- ✓ 모든 SQL은 트랜잭션의 일부이며, 따라서 select 만 실행하더라도 트랜잭션 종료(commit/rollback) 필요

CUBRID

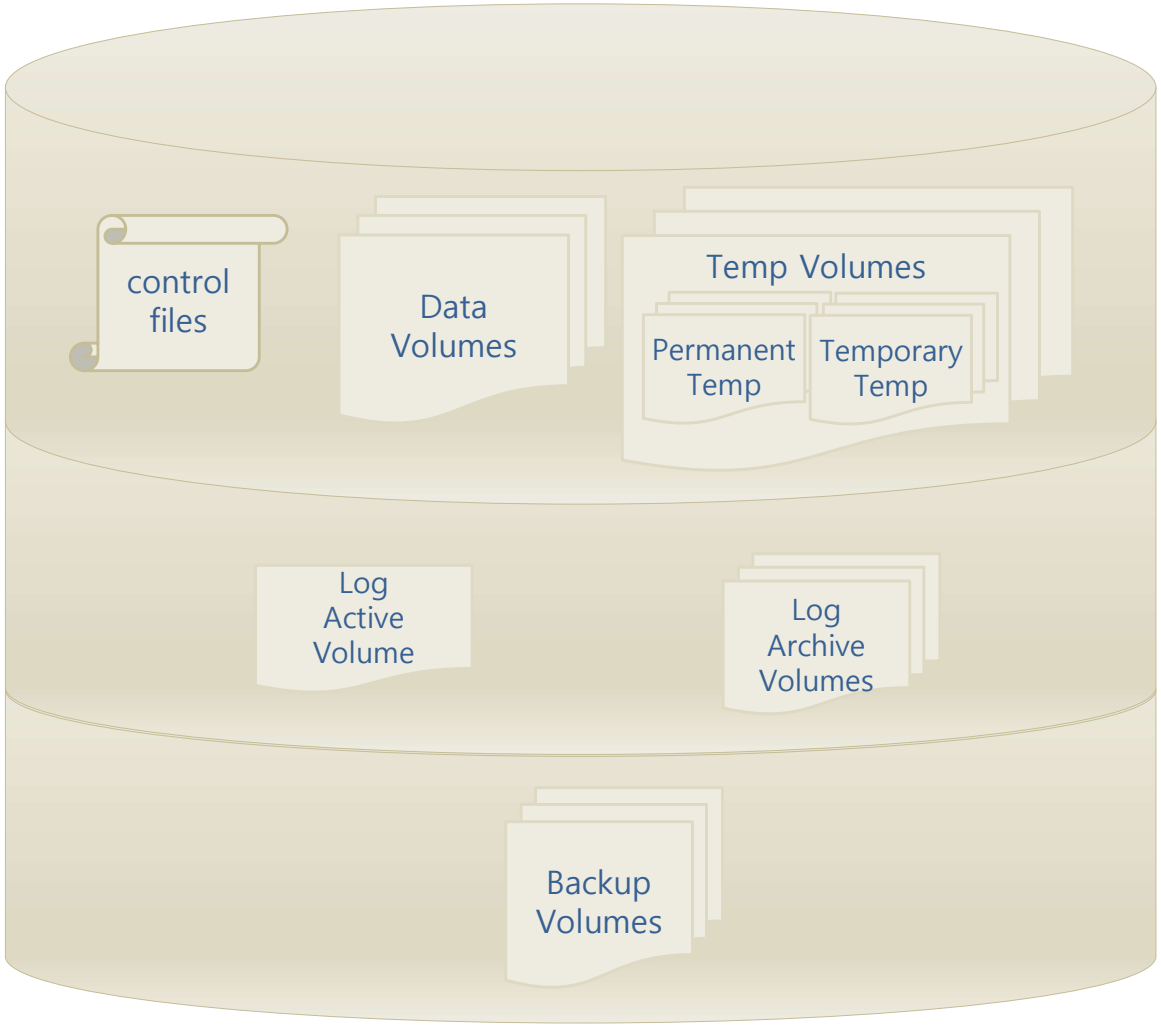
데이터베이스 구성 파일

- 데이터베이스 정보 화일
  - 데이터베이스 생성 시 데이터베이스에 관한 기본적인 위치 정보 등을 가지고 있다.
    - $CUBRID_DATABASES/databases.txt 에 저장된다.
    - DB이름, DB위치, 호스트명, LOG, LOB 정보가 기록된다.
    - 호스트명: 기본적으로 localhost 로 저장되나, HA 일 경우 Master/Slave 의 흐스트 명이 기록된다(예, db1:db2)

```
#db-name      vol-path        db-host      log-path        lob-base-path
demodb        /data/demodb    localhost    /data/demodb    /data/demodb//lob
```

  - 주의 사항
    - 데이터베이스 서버의 호스트 명이 변경될 경우 databases.txt 파일에 기록된 호스트명을 변경해야 한다.
    - CUBRID 사용자 계정은 파일에 대한 읽기/쓰기 권한이 있어야 한다.
    - 데이터베이스 삭제 등의 작업은 관련 명령을 이용하며, 직접적은 편집은 데이터베이스 사용에 심각한 문제가 발생할 수 있다.

데이터베이스 구성 파일 - 계속



데이터베이스 구성 파일 - 계속

- 제어/관리 파일(Control Files)
  - 제어 파일은 데이터베이스의 여러 볼륨의 정보와 로그 및 백업 정보를 기록한 파일들이다.
  - 볼륨정보(Volume information) 파일은 데이터베이스 볼륨에 대한 정보를 기록한다.
  - 로그정보(Log information) 파일은 로그 볼륨에 관한 정보를 기록한다.
  - 백업정보(Backup information) 파일은 백업 볼륨에 관한 정보를 기록한다.

- 데이터 볼륨(Data volume)
  - 스키마, 레코드, 멀티미디어 데이터와 같은 응용 프로그램의 데이터가 저장되는 공간이다.
  - 인덱스와 무결성 제약 조건(integrity constraints)의 인덱스 정보가 저장되는 공간이다.

- 임시 볼륨(Temp volume)
  - 질의 처리(join, group by, order by, sub-query, create index...)를 위해 사용되는 공간이다.
  - permanent 와 temporary 로 구분된다.
    - permanent temp 볼륨: 데이터베이스 생성시 볼륨 추가를 통해 생성된다.
    - temporary temp 볼륨: 질의 처리중 permanent temp 볼륨 이 부족한 경우 추가적으로 생성된다.
  - 서비스 운영 시 DB의 temporary temp 볼륨을 생성하는 비용은 상당히 클 수 있어 운영상황을 고려해 temp 볼륨 공간을 생성하는 것이 성능상 유리하다.
  - 한번 만들어진 temporary temp 볼륨은 재사용될 수 있으므로 운영 중엔 삭제되지 않으며, 데이터베이스 재시작시 삭제된다.

데이터베이스 구성 파일 - 계속

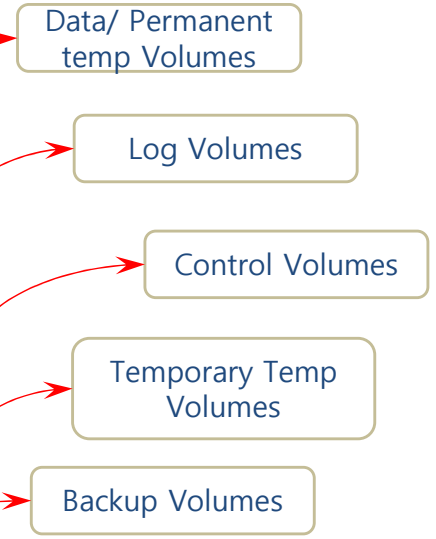
- 로그 볼륨(Log Volumes)
  - 활성 로그(Active Log)
    - 데이터베이스의 데이터에 대한 변경 정보를 기록한다. 커밋되었거나 롤백된 모든 트랜잭션에 대한 정보가 기록된다.
    - 디스크나 서버 장비의 장애가 발생할 경우 데이터베이스 복구를 위해 사용된다.
    - 각 데이터베이스는 한 개의 활성 로그를 가진다.
  - 보관 로그(Archive Log)
    - 활성 로그(Active log) 공간이 모두 사용되면 보관 로그(Archive log)로 복사하여 보관한다.
    - HA 환경에서는 서버간 동기화가 완료된 후, HA 가 아닌 경우 백업시 옵션을 통해 정리될 수 있다.
    - 서비스 처리량에 따라 일일 여러 개의 보관 로그(Archive log)가 생성된다.

- 백업 볼륨(Backup Volumes)
  - Backup 수행시 생성되는 파일이다.
  - Backup 볼륨은 백업 시점 데이터베이스의 스냅샷이다.
    - 온라인 백업의 경우 종료되지 않은 트랜잭션이 반영될 수 있으며, 종료되지 않은 트랜잭션은 해당 시점의 보관 로그에 기록된다.

데이터베이스 구성 파일 - 계속

- 데이터베이스 볼륨 파일 형식
  - Data volume 과 Permanent Temp volume 은 볼륨 파일 명으로는 구분이 어렵다.
  - 데이터베이스 사용량 확인을 통해 어느 볼륨 파일이 Permanent Temp volume 인지 확인할 수 있다.

```
-rw------- 1 cubrid cubrid 104857600 Jan  2 10:32 demodb
-rw------- 1 cubrid cubrid 536870912 Jan  3 11:09 demodb_x001
-rw------- 1 cubrid cubrid 536870912 Jan  3 11:09 demodb_x002
-rw------- 1 cubrid cubrid 536870912 Jan  3 11:09 demodb_x003
-rw------- 1 cubrid cubrid 104857600 Jan  2 10:32 demodb_lgat
-rw------- 1 cubrid cubrid 104857600 Jan  2 09:58 demodb_lgar_t
-rw------- 1 cubrid cubrid 104857600 Jan  2 11:32 demodb_lgar001
-rw------- 1 cubrid cubrid       263 Jan  2 15:37 demodb_vinf
-rw------- 1 cubrid cubrid       207 Jan  2 15:37 demodb_lginf
-rw------- 1 cubrid cubrid        57 Jan  2 18:26 demodb_bkvinf
-rw------- 1 cubrid cubrid   5472256 Jan  2 10:32 demodb_t32766
-rw------- 1 cubrid cubrid 213922816 Jan  2 18:26 demodb_bk0v000
```



3. CUBRID 설치

LINUX 서버에 설치 전 확인 사항

- Linux에서 설치
  - 공식지원 리눅스 버전
    - 32bit: CentOS 4 이상, Fedora 4, Ubuntu 6.10 이상, openSUSE 11 이상, Gentoo 2007 이상, Asianux 2 이상, Debian 4 이상
    - 64bit: CentOS 4 이상, Fedora 11, Ubuntu 9.04
  - 호환 CPU: Intel x86, Intel EM64T, AMD64
  - OS 버전 확인 : Linux Kernel 2.4과 glibc 2.3.4버전 이상만 지원한다.

```
$ uname –a; rpm –q glibc
Linux db1 2.6.18-53.1.14.el5xen #1 SMP Wed Mar 5 12:08:17 EST  x86_64 x86_64 x86_64 GNU/Linux
glibc-2.17-324.el7_9.x86_64
```

Kernel 2.6.18

glibc 2.17

  - 필요 라이브러리: ncurses, libgcrypt, libstdc++ 라이브러리가 필요하므로, 설치되었는지 확인이 필요하다.

```
$ rpm -q ncurses libgcrypt libstdc++
ncurses-5.9-14.20130511.el7_4.x86_64
libgcrypt-1.5.3-14.el7.x86_64
libstdc++-4.8.5-28.el7.x86_64
```

  - OS bit 확인: Linux OS bit(32/64bit) 를 확인하고 설치할 CUBRID 버전과 bit를 선택한다.

```
$ uname -a
Linux db1 3.10.0-862.el7.x86_64 #1 SMP Fri Apr 20 16:44:24 UTC 2018 x86_64 x86_64 x86_64
GNU/Linux
```

64 bit OS

  - 방화벽 설정
    - Linux 서버 환경에서 CUBRID 사용하기 위해서 다음 포트는 기본적으로 오픈 한다.
    - CUBRID Admin : 8001(CUBRID 9.3.1이하: 8001, 8002)
    - 브로커: 30000(질의편집기), 33000(응용)

CUBRID

LINUX 서버에 설치

- Linux 버전
  - CUBRID 설치 및 관리할 사용자 계정을 생성해서 CUBRID 설치를 권장한다.
  - CUBRID 데이터베이스 Server와 Client(CSQL, CUBRID Utilities, 브로커)는 동일 버전에서만 완전한 호환을 지원한다.

```
root# useradd cubrid
root# su - cubrid

$ sh CUBRID-11.0.6.0313-e1160bb-Linux.x86_64.sh
CUBRID Installer Version: 11.0.6.0313-e1160bb, Copyright (c)
...
Do you accept the license? [yN]: y
By default the CUBRID will be installed in:
  "/home/cubrid/CUBRID-11.0.6.0313-e1160bb-Linux.x86_64"
Do you want to include the subdirectory CUBRID-11.0.6.0313-e1160bb-Linux.x86_64?
Saying no will install in: "/home/cubrid" [Yn]: y
Using target directory: /home/cubrid/CUBRID-11.0.6.0313-e1160bb-Linux.x86_64

Since CUBRID broker and server versions should match, please make sure that you are running the same version if you
operate them in separate machines.
Do you want to continue? [Yn] : y
Extracting, please wait...

Unpacking finished successfully
demodb has been successfully created.

If you want to use CUBRID, run the following command to set required environment variables.
  $ . /home/cubrid/.cubrid.sh

$ . /home/cubrid/.cubrid.sh
$ cubrid_rel
CUBRID 11.0 (11.0.6.0313-e1160bb) (64bit release build for Linux) (Feb 11 2022 19:31:44)
```

CUBRID 11.0 이후 CUBRID 설치 디렉토리명에 버전 정보를 포함한다. 이는 업그레이드시 기존 버전을 보전하기 위한 방법이다. 원하는 위치에 설치하고 싶으면, 원하는 디렉토리를 만들고 그곳에서 설치한다. 설치중 위치를 묻는 질문에서 n 을 입력하면 현재 디렉토리에 설치된다.

CUBRID Engine 구조

디렉토리	설 명	용량
CUBRID/	CUBRID 의 홈 디렉토리	500M
bin/	실행 파일	19M
conf/	환경 설정 파일	100K
databases/	databases.txt 와 sample 데이터베이스(demodb 설치 시 1.5G 필요)	1K
demo/	demodb 생성 스크립트	1M
include/	C include 파일	500K
java/	JAVA SP 관련 파일	300K
jdbc/	CUBRID jdbc driver	700K
lib/	library 파일	455M
locales/	CUBRID 언어설정 정보 디렉토리	5M
log/	각종 log	1M
server/	데이터베이스 에러 로그	-
manager/	매니저 서버 로그	-
broker/	브로커 접속 로그	-
sql_log/	SQL 로그	-
error/	브로커 에러 로그	-
msg/	CUBRID 에서 사용되는 각종 메시지 파일	6M
share/	구동 및 종료에 필요한 기본 script	5M
tmp/	임시 파일	24K
var/	CUBRID 프로세스들이 서비스 중 사용되는 정보 파일	-

실습

1. CUBRID 를 설치한다.
    1. VMware Player 를 실행하고, cubrid 계정으로 접속
    2. cubrid 계정 홈 디렉토리에 있는 CUBRID-11...sh 을 실행하여 설치한다.
        1. 설치 위치는 /home/cubrid/CUBRID 로 한다.
            - upgrade 등을 위해 설치 위치가 CUBRID 버전 정보를 포함한다. 처음 설치이므로, CUBRID 설치 위치가 불편하지 않도록 간단하게 한다.

```
[cubrid@db1 ~]$ mkdir CUBRID; cd CUBRID
[cubrid@db1 CUBRID]$ sh ../CUBRID-11.0.6.0313-e1160bb-Linux.x86_64.sh
CUBRID Installer Version: 11.0.6.0313-e1160bb, Copyright (c) Search Solution Corporation
This is a self-extracting archive
The archive will be extracted to:

If you want to stop extracting, p
Copyright (C) 2008 Search Solutio
Copyright (C) 2016 CUBRID Corporat
CUBRID is registered trademark of

This Software is released under A
For brevity, CUBRID Server Engine
For details, please refer to the
```

```
Do you accept the license? [yN]:
y
By default the CUBRID will be installed in:
  "/home/cubrid/CUBRID/CUBRID-11.0.6.0313-e1160bb-Linux.x86_64"
Do you want to include the subdirectory CUBRID-11.0.6.0313-e1160bb-Linux.x86_64?
Saying no will install in: "/home/cubrid/CUBRID" [Yn]:
n

Using target directory: /home/cubrid/CUBRID

Since CUBRID broker and server versions should match, please make sure that you are running the same version if you operate
them in separate machines.
Do you want to continue? [Yn] :
y
Extracting, please wait...

Unpacking finished successfully
demodb has been successfully created.

If you want to use CUBRID, run the following command to set required environment variables.
  $ . /home/cubrid/.cubrid.sh
```

    3. CUBRID 환경 변수 등을 적용하기 위해 다시 로그인한다.
2. 정상 설치 여부 확인
    1. CUBRID 버전 확인 Utilities 인 cubrid_rel 을 수행하여 버전 정보가 올바로 나오는지 확인한다.

```
[cubrid@db1 ~]$ cubrid_rel

CUBRID 11.0 (11.0.6.0313-e1160bb) (64bit release build for Linux) (Feb 11 2022 19:31:44)
```

4. CUBRID Admin

CUBRID Admin

- CUBRID Admin
  - 데이터베이스 관리
    - 데이터베이스 구동 및 종료
    - 데이터베이스 생성 마법사를 이용한 데이터베이스 생성
    - 백업/복구, Unload/load(export/import) 등 여러 utilities 사용
  - 브로커 관리
    - 브로커 구동 및 종료
    - 브로커 상태 모니터링 등

- CUBRID Manager Server
  - CUBRID Admin 을 사용하기 위해서는 CUBRID Manager Server 가 구동되어 있어야 한다.
    - 기본적으로 CUBRID 서비스 구동시 같이 구동된다.
  - 별도 구동 및 종료는 다음과 같이 수행하면 된다.

```
$ cubrid manager start
@ cubrid manager server start
++ cubrid manager server start: success
$ cubrid manager stop
@ cubrid manager server stop
++ cubrid manager server stop: success
```

- CUBRID Manager Server 주요 설정 (cm.conf)
  - cm_port
    - CUBRID Admin 에서 Manager Server 연결을 위해 사용하는 포트로, 기본값은 8001 이다.
    - CUBRID 9.3.1 이하: cub_auto 가 사용하며, cub_js 는 주어진 값 +1 의 포트 사용

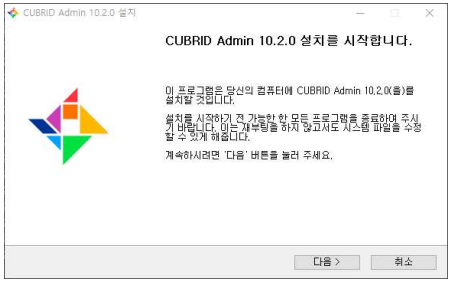
CUBRID Admin 설치

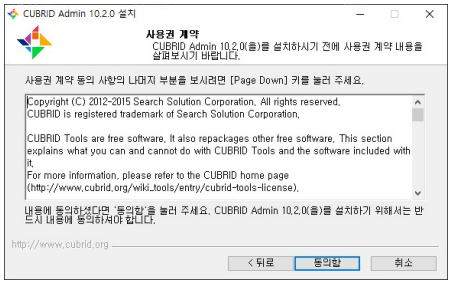
http://ftp.cubrid.org

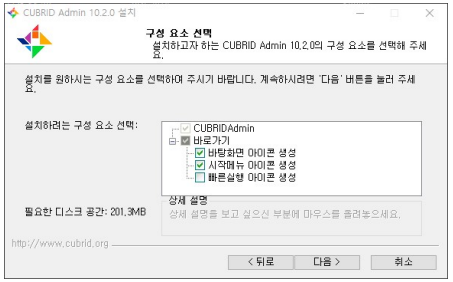
CUBRID_Engine/	CUBRID Database Engine installation files
CUBRID_Drivers/	CUBRID API Connectors
CUBRID_Tools/	CUBRID Tools installation files
CUBRID_Docs/	CUBRID Database documentation files
CUBRID_Repos/	CUBRID RPM packages for YUM
cubrid_repo_settings/	RPM for CUBRID repo

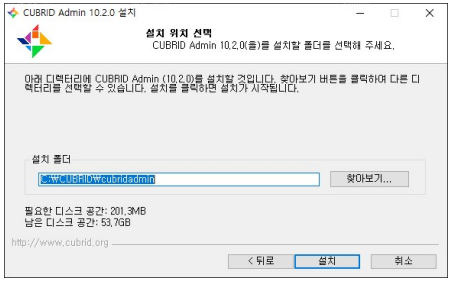
Index of /CUBRID_Tools

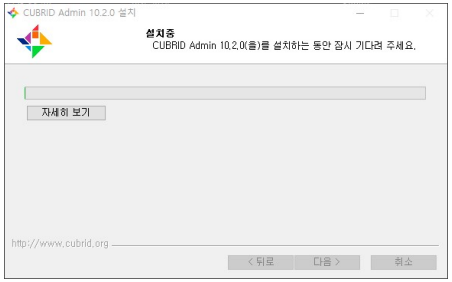
Name	Last modified	Size	Description
Parent Directory		-	
CUBRID_ACXEL/	2012-01-25 20:18	-	
CUBRID_Admin/	2020-04-01 09:08	-	
CUBRID_AutoSet/	2013-09-27 23:16	-	
CUBRID_Manager/	2019-12-13 18:17	-	
CUBRID_Manager_Server/	2014-06-18 18:31	-	
CUBRID_Migration_Toolkit/	2019-12-13 18:18	-	
CUBRID_Query_Browser/	2014-05-26 18:59	-	
CUBRID_Web_Manager/	2013-08-23 15:13	-	
CUBRID_Webquery/	2012-12-17 14:02	-	
Dev/	2017-09-20 18:12	-	











CUBRID

CUBRID Admin 설치와 접속

- CUBRID Admin 실행 환경
  - CUBRID Admin은 Java 기반으로 되어 있어 JVM 이 필요하며, 실행 환경은 JRE 만 설치하여도 된다.
  - CUBRID Admin은 Windows 32bit, 64bit 를 지원한다.

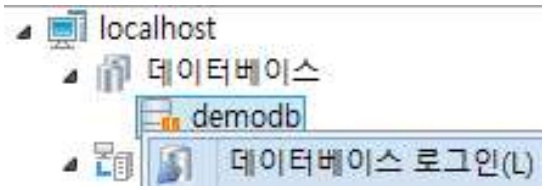
- CUBRID Admin 설치
  - 32/64bit OS에 해당하는 exe 파일을 다운로드 후 실행하여 설치한다.
  - Linux: 32/64bit OS에 해당하는 tar.gz 파일을 다운로드하고 압축을 해제하여 cubridmanager를 실행하거나, sh 파일을 다운로드하여 console에서 실행하여 설치한다.

- CUBRID Admin 실행
  - C:₩CUBRID₩cubridadmin/cubridadmin.exe 파일을 클릭해 실행한다.

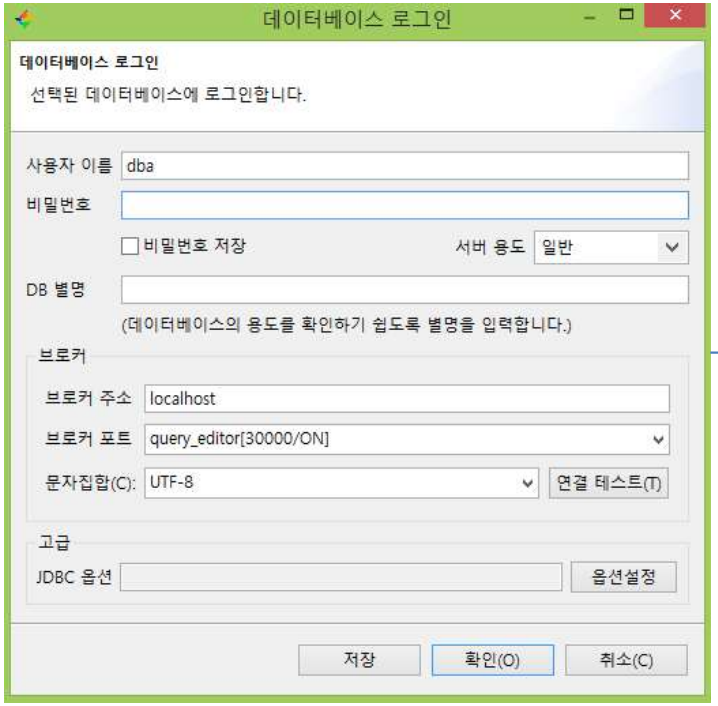
- CUBRID Admin 접속
  - CUBRID Admin 접속 포트는 8001번이며 CUBRID Server에서 포트 오픈이 필요하다.
  - 로그인(default 사용자인 admin의 비밀번호는 admin이다. 처음 로그인 후 비밀번호를 변경해야 한다)

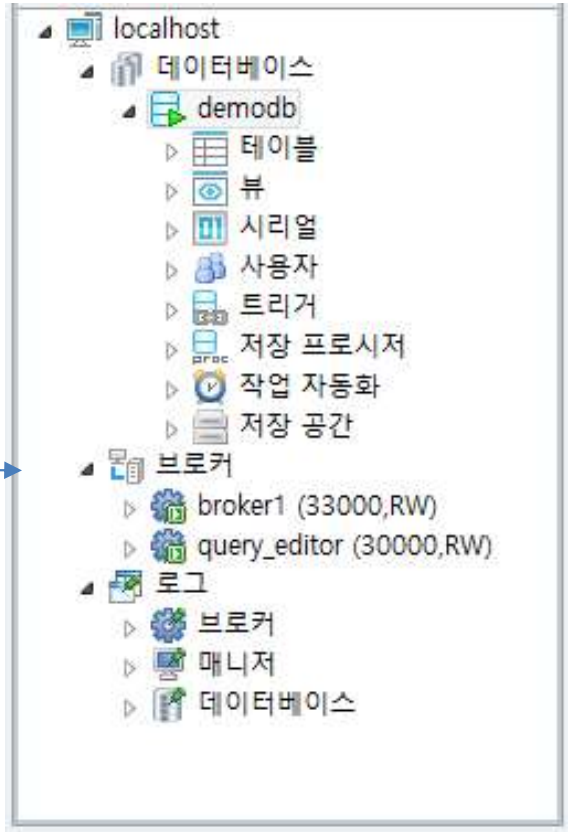
CUBRID Admin 설치와 접속 - 계속

- CUBRID Admin
  - 데이터베이스 로그인
    - 데이터베이스 별 사용자로 로그인 한다.

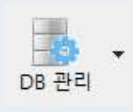
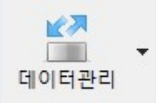


    - Default 계정: dba, public
    - Default 비밀번호는 없다.(공백으로 두고 접속)

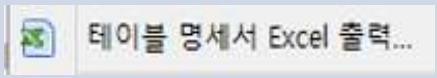




CUBRID Admin 도구 모음

버튼	설명	버튼	설명
	- 큐브리드 호스트 추가		- CM 사용자 관리 - DB 사용자 관리
	- 데이터베이스 시작(선택된 DB가 중지 상태일 경우에만 활성 상태)		- 데이터베이스 언로드
	- 테이블, 뷰, 시리얼, 트리거를 생성하는 화면 호출		- CMT(cubrid migariontool) 실행
	- 내보내기(SQL, CSV, TXT, loaddb 포맷 등) - 가져오기 화면을 호출		- 도움말 새로운 기능 - 버그 신고 - 서버, 클라이언트 버전 확인
	- DB 비교 마법사(스키마, 데이터) - ERD 편집기 - 다중호스트 설정 편집기		- Cubrid.org에서 검색

CUBRID Admin 고급기능 - 1

기능	메뉴위치	처리순서
	메뉴-동작 ➔ 테이블명세서 Excel 출력	1. "동작 ➔ 테이블명세서 Excel 출력" 클릭 2. 명세서 경로, 파일명, 문자집합, 문서스타일 입력후 "OK"버튼 클릭 3. 명세서 파일 확인

테이블 명세

시스템 이름		날짜	2020.05.18	작성자			
테이블명	athlete						
테이블 설명							
컬럼명	자료형	크기	NULL	PK	FK	기본값	컬럼 설명
code	INTEGER			Y			
name	VARCHAR	40					
gender	CHAR	1	Y				
nation_code	CHAR	3	Y				
event	VARCHAR	30	Y				

인덱스 정의

NO	인덱스명	컬럼 ID	순서	비고
1	pk_athlete_code	code	1	

DDL

```
CREATE TABLE athlete(
code integer AUTO_INCREMENT(16693,1) NOT NULL,
[name] character varying(40) NOT NULL,
gender character(1),
nation_code character(3),
event character varying(30)
) COLLATE iso88591_bin ;

ALTER TABLE athlete ADD CONSTRAINT pk_athlete_code PRIMARY KEY (code);
```

단순형

테이블 명세

시스템 이름		날짜	2020.05.18	작성자		
테이블명						
테이블 ID	athlete					
컬럼명	컬럼 ID	자료형	크기	NULL	PK	FK	비고
	code	INTEGER			Y		
	name	VARCHAR	40				
	gender	CHAR	1	Y			
	nation_code	CHAR	3	Y			
	event	VARCHAR	30	Y			

인덱스 정의

NO	인덱스명	컬럼 ID	순서	비고
1	pk_athlete_code	code	1	

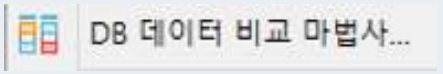
DDL

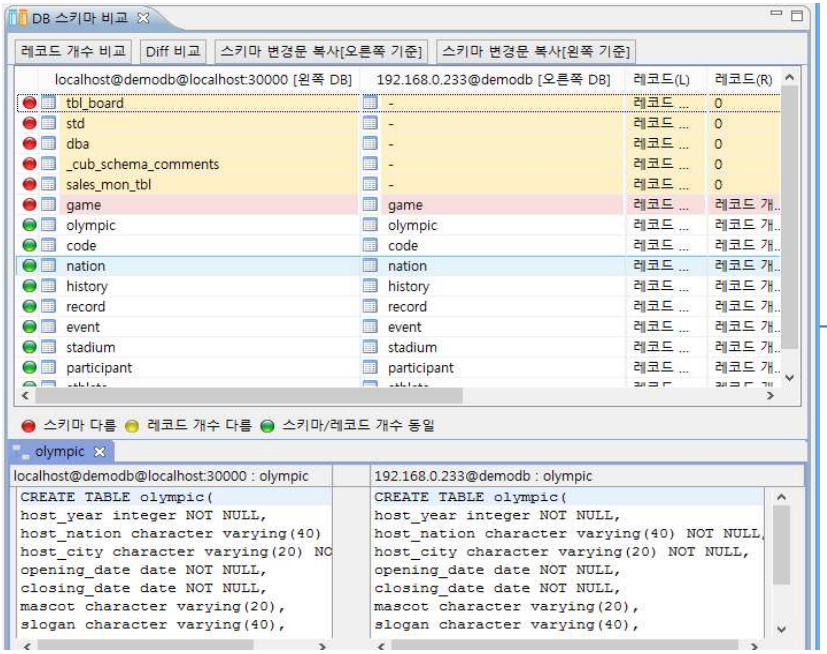
```
CREATE TABLE athlete(
code integer AUTO_INCREMENT(16693,1) NOT NULL,
[name] character varying(40) NOT NULL,
gender character(1),
nation_code character(3),
event character varying(30)
) COLLATE iso88591_bin ;

ALTER TABLE athlete ADD CONSTRAINT pk_athlete_code PRIMARY KEY (code);
```

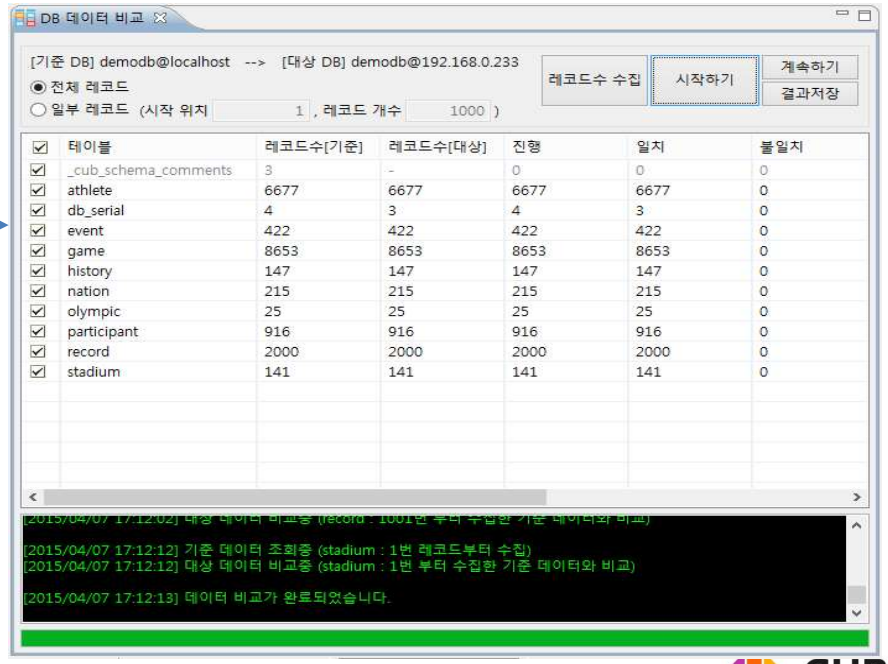
일반형

CUBRID

CUBRID Admin 고급기능 - 2

기능	메뉴위치	처리순서
	툴바-고급기능 ➔ DB스키마 비교 마법사	1. 스키마를 비교하고자 하는 DB서버에 각각 로그인 2. 각각의 DB서버를 선택 3. "고급기능 ➔ DB스키마 비교 마법사" 클릭 4. DB 스키마 비교 결과 화면 확인 5. 비교하고자 하는 테이블 목록 선택 후, "레코드 개수 비교", "Diff 비교", "스키마 변경문 복사 " 버튼을 클릭 4. 비교결과 확인
	툴바-고급기능 ➔ DB데이타 비교 마법사	1. 데이타를 비교하고자 하는 DB서버에 각각 로그인 2. 각각의 DB서버를 선택 3. "고급기능 ➔ DB데이타 비교 마법사" 클릭 4. DB 데이터 비교 결과 화면 확인 5. "레코드 수집" 버튼 클릭하여 레코드 수 비교 6. "시작하기" 버튼 클릭하여 데이터 비교 7. 비교결과 확인

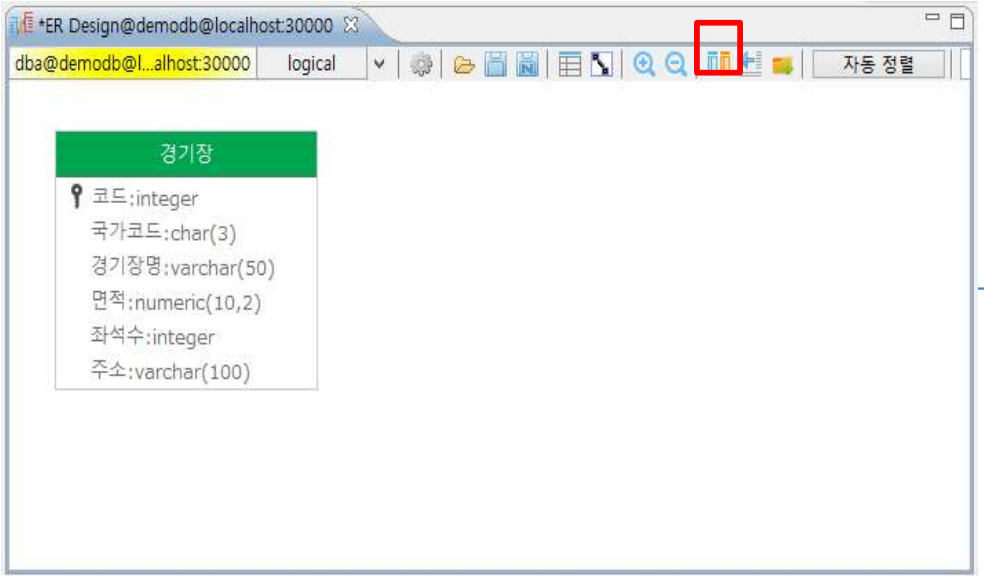


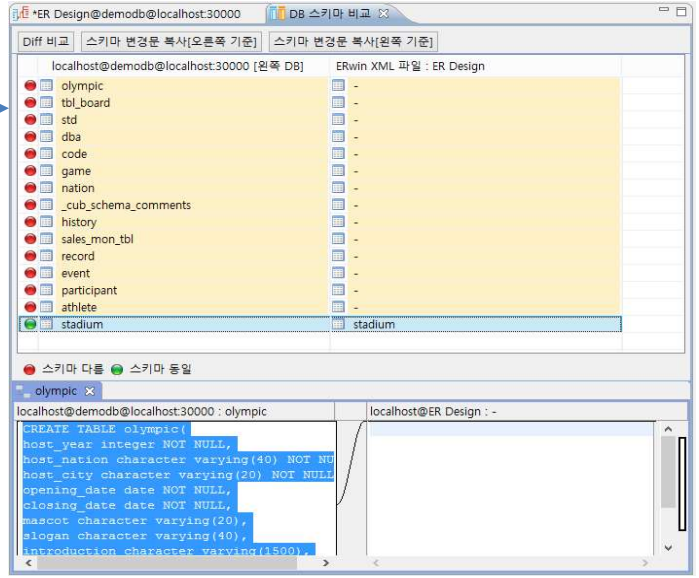


CUBRID

CUBRID Admin 고급기능 - 3

기능	메뉴위치	처리순서
	툴바-고급기능 ➔ ERD 편집기	○ DB 생성 및 테이블 생성 완료(테이블과 컬럼 설명 입력 완료) 1. DB서버에 로그인 2. "고급기능 ➔ ERD 편집기" 클릭 3. 작성하고자 하는 테이블을 선택하여 ERD 창에 옮기기 4. ERD 편집기 상단에 logical을 선택하면 테이블과 컬럼 설명으로 입력한 logical ERD 가 보여 짐 5. physical을 클릭하면 컬럼 이름이 보여 짐 ○ DB 생성 (신규 DB 생성하여 개발 시작 단계) 1. "고급기능 ➔ ERD 편집기" 클릭 2. ERD 작성(테이블 추가, 컬럼 추가, 테이블과 컬럼설명 입력 3. ERD 편집기 툴박스의 "ERD와 데이터베이스의 DDL 비교" 클릭 4. 테이블 선택 후 "Diff 비교", "스키마 변경문 복사" 버튼을 클릭 5. "Diff 비교" 버튼 클릭하여 스크립트 내용을 확인 6. 비교결과를 복사하여 질의편집기에서 실행하여 ERD에서 작성 된 DB를 생성 7. 결과 확인





5. CUBRID 구동 및 종료

CUBRID 서비스 구동과 종료

- 서비스 구동
  - CUBRID 운영에 필요한 기본 프로세스 구동
  - master, broker, manager server 구동
  - database server 는 별도 구동, 또는 설정을 통하여 서비스 구동 시 같이 구동 가능
  - HA 환경일 경우 15장 CUBRID HA 참고
- 명령어

```
$ 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 관련 모든 작업은 CUBRID 가 설치된 cubrid 계정을 사용할 것을 권장한다.
- 다른 계정에 환경을 추가 설정하여 여러 계정으로 사용시 구동/종료에 심각한 영향을 줄 수 있다.

CUBRID 서비스 종료

- 서비스 종료
  - CUBRID 관련 모든 프로세스 종료
    - master, broker, manager server 및 database server 종료
    - HA 환경일 경우 HA 도 종료된다.
  - HA 환경일 경우 15장 CUBRID HA 참고
- 명령어

```
$ cubrid service stop
@ cubrid broker stop
++ cubrid broker stop: success
@ cubrid manager server stop
++ cubrid manager server stop: success
@ cubrid master stop
++ cubrid master stop: success
```

CUBRID Server 구동

- 데이터베이스 구동
  - 사용하는 데이터베이스 별 서버 구동
  - HA 환경일 경우 15장 CUBRID HA 참고
    - 서버를 직접 구동하지 않고, HA 를 구동하는 형태
- 명령어

```
$ cubrid server start demodb
@ cubrid server start: demodb

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

CUBRID   R11.0

++ cubrid server start: success

→ 이미 구동된 경우 다음과 같은 메시지 출력
++ cubrid server 'demodb' is running.
```

- demodb 는?
  - CUBRID 설치시 생성되는 연습용 데이터베이스
  - CUBRID 처음 사용자를 위해 연습용 테이블 및 데이터가 들어가 있다.

CUBRID Server 종료

- 데이터베이스 종료
  - 사용하는 데이터베이스 별 서버 종료
  - HA 환경일 경우 15장 CUBRID HA 참고
    - 서버를 직접 종료하지 않고, HA 를 구동하는 형태
- 명령어

```
$ 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 종료된 경우 다음과 같은 메시지 출력
++ cubrid server 'demodb' is not running.
```

CUBRID 브로커 구동 및 종료

- 구동
  - 기본적으로 CUBRID service 구동 시 같이 구동된다.
  - 별도 구동 방법은 다음과 같다.

```
$ cubrid broker start
@ cubrid broker start
++ cubrid broker start: success

→ 이미 구동된 경우 다음과 같은 메시지 출력
++ cubrid broker is already running.
```

- 종료
  - CUBRID service 종료 시 같이 종료된다.
  - 별도 종료 방법은 다음과 같다.

```
$ cubrid broker stop
@ cubrid broker stop
++ cubrid broker stop: success

→ 이미 종료된 경우 다음과 같은 메시지 출력
++ cubrid broker is not running.
```

CUBRID Manager Server 구동 및 종료

- 구동
  - 기본적으로 CUBRID service 구동 시 같이 구동된다.
  - 별도 구동 방법은 다음과 같다.

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

→ 이미 구동된 경우 다음과 같은 메시지 출력
++ cubrid manager server is already running.
```

- 종료
  - CUBRID service 종료 시 같이 종료된다.
  - 별도 구동 방법은 다음과 같다.

```
$ cubrid manager stop
@ cubrid manager server stop
++ cubrid manager server stop: success

→ 이미 종료된 경우 다음과 같은 메시지 출력
++ cubrid manager server is not running.
```

실습

1. CUBRID 서비스를 구동한다.

```
[cubrid@db1 ~]$ 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
```

2. 정상 구동 여부 확인
    1. cubrid service status 를 이용하여 정상적으로 구동되었는지 확인한다.

```
[cubrid@db1 ~]$ cubrid service status
@ cubrid master status
++ cubrid master is running.
@ cubrid server status
@ cubrid broker status
  NAME                     PID  PORT    AS   JQ                  TPS                 QPS   SELECT   INSERT   UPDATE   DEL
ETE   OTHERS     LONG-T     LONG-Q         ERR-Q  UNIQUE-ERR-Q  #CONNECT   #REJECT
================================================================================================================================
================================================================================
* query_editor           2204 30000     5    0                     0                   0        0        0        0
  0        0     0/60.0     0/60.0            0             0         0         0
* broker1                2215 33000     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 running.
```

실습 - 계속

3. 실습용 데이터베이스 demodb 를 구동한다.

```
[cubrid@db1 ~]$ cubrid server start demodb
@ cubrid server start: demodb

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

CUBRID 11.0

++ cubrid server start: success
```

4. 정상 구동 여부 확인
   1. cubrid server status 를 이용하여 정상적으로 구동되었는지 확인한다.

```
[cubrid@db1 ~]$ cubrid server status
@ cubrid server status
 Server demodb (rel 11.0, pid 2349)
```

   2. cub_commdb -P 명령도 수행하여 본다.

```
[cubrid@db1 ~]$ cub_commdb -P
 Server demodb (rel 11.0, pid 2546)
```

6. 데이터베이스 생성

데이터베이스 생성

- 데이터베이스 생성
  - 데이터베이스 명은 최대 17자로 대소문자 구분하며, 특수문자로 시작할 수 없다.
  - 데이터 보존 기간을 고려하여 저장될 전체 데이터의 크기를 산정해 생성한다.
    - 생성 가능한 최대 크기는 20G이며, 보다 큰 용량의 데이터베이스를 생성하기 위해선 다음에 소개되는 볼륨 추가 기능을 이용하여야 한다.
      - 질의 수행중 사용될 수 있는 Perment Temp 볼륨 역시 볼륨 추가 기능을 이용하여 생성한다.
    - 생성된 데이터베이스의 크기는 줄일 수 없으므로 주의해야 한다.

- 데이터베이스 기본 정보
  - 데이터베이스의 입출력 단위는 페이지이며, 그 크기는 16 Kbytes 이다.
  - 로케일: 데이터베이스에서 기본적으로 사용할 문자 관련 지역 및 문자의 코드셋
    - 기본 지원 지역: 영어, 한국어, 터키어
    - 기본 지원 문자 코드셋: iso88591, euckr, utf8

  - 볼륨 : 스키마, 데이터 등이 저장되는 공간
    - 32bit 사용시 한 개 볼륨의 최대 크기 : 2G
    - 데이터 볼륨 : 각 데이터, 인덱스 저장됨. (10.x 미만은 데이터만 저장)
    - 로그 볼륨 : 데이터 변경에 관련된 정보 저장. 백업 복구 시 사용됨.
    - 템프 볼륨 : 질의 처리 및 정렬(sorting)을 수행할 때 중간, 최종 결과를 임시로 저장하는 공간

    - 범용 볼륨 (10.x 미만): 스키마정보, 카다로그 정보 저장됨.
    - 인덱스 볼륨 (10.x 미만): 인덱스정보 저장되며, 10.x 인 경우 반드시 추가해야 함

데이터베이스 생성 - 계속

- 사용 예

```
# testdb 를 한글 utf8 문자셋으로 데이터 볼륨 위치는(-F) /data/mydb 으로, 로그 볼륨 위치는(-L) /data/mydb/log 으로 하여 10G의 크기(--db-volume-size)로 생성한다.
$ cubrid createdb --db-volume-size=10G -F /data/mydb -L /data/mydb/log mydb ko_KR.utf8

# testdb 가 이미 존재하면 기존 데이터베이스를 삭제(-r)하고 기본크기(512M)로 현재 디렉토리에 생성한다.
$ cubrid createdb -r mydb ko_KR.utf8
```

볼륨 추가

- 데이터가 저장될 공간 추가
  - 데이터베이스 생성후 데이터와 인덱스가 저장될 공간 추가
  - 정렬 등의 질의 수행을 위해 사용될 permanent temp 볼륨 추가
    - 데이터 공간의 10% 정도로 생성하나, 질의의 복잡도에 따라 적절히 조정하여야 한다.
    - 효율적인 질의 처리를 위해 반드시 추가하여야 한다.
    - 부족할 경우 자동으로 temporary temp 볼륨이 추가되나, 그로 인한 I/O 급증, 디스크 공간 부족 등이 야기될 수 있으므로 자동으로 추가된 경우 관련 질의 튜닝 또는 permanent temp 볼륨을 추가해야 한다.
  - 운영중 서비스 중단 없이 볼륨 추가 가능
    - 볼륨의 여유 공간이 10% 미만으로 될 경우 자동으로 추가됨. (설정을 따르며 기본 512M)
      - 사용량이 많은 시간대에 자동으로 추가될 수 있으므로 모니터링을 통해 자동 확장전에 적정 용량으로 추가하는 것으로 권장
    - 운영자가 임의로 볼륨 추가 가능
    - 볼륨 추가시 디스크 I/O 가 급증하게 되므로 사용량이 적은 시간대에 너무 크지 않게 만들어야 함.
  - 볼륨이 만들어질 위치(-F)를 지정하지 않으면 데이터베이스가 생성된 위치에 만들어 진다.
    - 다만 volume_extension_path(cubrid.conf) 가 설정된 경우 그곳에 생성되나, 보통은 설정하지 않는다.

- 사용 예

```
# mydb 에 서버가 중단된 상태(-S)에서 데이터베이스가 있는 위치에 data 공간(-p data) 20G 를 추가한다.
# 서비스 중에 추가할때는 -C(-S 대신) 옵션 사용
$ cubrid addvoldb -S -p data --db-volume-size=20G mydb

# 정렬등 일부 질의 처리를 위한 permanent temp 볼륨을(-p temp) 데이터 영역의 10% 정도로 반드시 생성한다.
$ cubrid addvoldb -S -p temp --db-volume-size=2G mydb

# CUBRID 10.x 미만일 경우 인덱스가 저장될 공간(-p index) 을 추가하여야 한다.
$ cubrid addvoldb -S -p index --db-volume-size=1G mydb
```

실습

1. 다음의 조건으로 데이터베이스를 생성한다.
    1. /data/DB 아래에 1G 의 크기를 가지는 한글 utf8 을 사용하는 testdb 를 생성한다.
        1. /data/DB 는 실습 환경에서 존재하지 않으므로 생성하여야 하며, 최소한 DB 디렉토리는 cubrid 계정에 사용 권한을 주어야 한다.

```
[cubrid@db1 ~]$ su -
Password:
[root@db1 ~]# mkdir /data ; mkdir /data/DB
[root@db1 ~]# chown cubrid. /data/DB
[root@db1 ~]# ls -l /data
total 0
drwxr-xr-x. 2 cubrid cubrid 6 May 15 16:54 DB
```

        2. 데이터베이스를 생성한다.

```
[cubrid@db1 ~]$ cubrid createdb --db-volume-size=1G -F /data/DB testdb ko_KR.utf8
Creating database with 1.0G size using locale ko_KR.utf8. The total amount of disk space needed is 2.0G.

CUBRID 11.0
```

2. 다음의 조건으로 데이터베이스의 볼륨을 추가한다.
    1. 방금 생성한 testdb 에 data 가 저장될 1G 크기의 볼륨을 기존과 동일한 위치에 추가한다.

```
[cubrid@db1 ~]$ cubrid addvoldb -S -p data --db-volume-size=1G testdb
[cubrid@db1 ~]$
```

    2. 실습용 데이터베이스 demodb 에 정렬 등의 질의 처리를 위해 512M 크기의 permanent temp 볼륨을 demodb 의 다른 볼륨들과 동일한 위치에 추가한다.

```
[cubrid@db1 ~]$ cubrid addvoldb -C -p temp demodb
[cubrid@db1 ~]$
```

볼륨 추가시 볼륨의 크기를 지정하지 않으면 기본 512M 로 추가된다.
-C 는 생략 가능하다.

7. SQL Interpreter (csql)

SQL Interpreter (csql)

- csql
  - CSQL 은 CUBRID와 함께 설치되며, 대화형(interactive) 방식과 일괄 수행(batch) 방식으로 SQL 을 수행하고 결과를 조회할 수 있는 도구이다.
  - CSQL 은 독립 모드(Standalone Mode), 클라이언트/서버 모드(Client/Server Mode)를 제공한다.

- 기본 접속 방법

```
# HA 환경이 아닌 single 환경인 경우
# 서비스 중에(-C) db_user 계정으로(-u db_user) 계정 암호를 이용하여(-p 'qwe123') demodb 에 접속한다.
$ csql -C -u db_user -p 'qwe123' demodb

# HA 환경이고 서비스 구동중일 경우 데이터베이스 이름 뒤에 @ 와 접속할 서버 이름을 붙여준다.
# 현재 접속한 서버(localhost) 에 구동된 demodb 에 접속한다.
$ csql -C -u db_user -p 'qwe123' demodb@localhost
# 특정 서버(db1) 에 구동된 demodb 에 접속한다.
$ csql -C -u db_user -p 'qwe123' demodb@db1

# 서비스가 중단된 상태에서(-S) 일반 트랜잭션 모드로 db_user 계정으로 demodb 에 접속한다.
# 암호를 입력하지 않으면 사용자 계정의 암호를 물어온다,
$ csql -S -u db_user demodb
Password:
```

SQL Interpreter (csql) - 계속

- 사용 예

```
# 서비스 중에 자동 commit 이 아닌 일반 트랜잭션 모드(--no-auto-commit)로 demodb 에 접속한다.
# csql 사용시 SQL 수행후 commit/rollback 을 처리하지 않아 lock 으로 인해 서비스 장애를 유발할 수 있어 기본적으
   로 SQL 수행후 commit 이 수행되도록 auto commit 을 사용하고 있음.
   - 트랜잭션이 필요할 경우, 즉 update/delete 에 실수가 있을 수 있어 rollback 이 필요한 경우 선별적으로 사용하
     여야 하며, 실수를 방지하기 위해선 해당 옵션의 사용보단 다음과 같이 접속후 설정 변경을 권장한다.
   - csql 접속후 ;au off 로 트랜잭션모드로 변경할 수 있으며, ;au on 으로 auto commit 모드로 변경 가능
# -C 옵션은 기본 적용되어 서비스가 구동 중일 경우 사용할 필요는 없다.
# 암호 입력시 암호에 특수문자가 들어가 있지 않다면 ' ' 를 사용할 필요 없다.
$ csql -u db_user -p qwe123 --no-auto-commit demodb

# SQL로 이루어진 파일(sql.list)을 읽어 들여(-i sql.list) 수행한다. 이때 SQL문장의 끝은 반드시 ; 이 있어야 한다.
$ csql -u db_user -p qwe123 -i sql.list demodb

# SQL 문장을 넘겨주어 바로 실행(-c "select ...")하고 그 결과를 sql.out 파일로 받는다(-o sql.out).
# SQL 문장의 끝은 반드시 ; 이 있어야 한다.
$ csql -u db_user -p qwe123 -c "select * from record;" -o sql.out demodb
```

SQL Interpreter (csql) - 계속

- SQL 수행
  - SQL buffer 사용없이 바로 수행 (질의 끝에 ; 추가)
  - 예시

```
csql> select * from olympic;

...
 1900  'France'              'Paris'               05/14/1900    10/28/1900    NULL                  NULL                  'The
1900 Paris Games allowed women to compete and a total of nineteen countries participated at the Games. The
Games were held at the same time as the World Exhibition in Paris and were dragged out over five months. As a
result the 1900 Paris Games were a flop.'
         1896  'Greece'              'Athens'              04/06/1896    04/15/1896    NULL                  NULL
'Thirteen countries participated at the 1896 Athens Games with around 300 athletes taking part in the competition.
There were 43 events contested which fell into the following categories; athletics (track and field), cycling, fencing,
gymnastics, shooting, swimming, tennis, weightlifting and wrestling. Competition was open to any man who
wanted to participate. All medals were silver and they were only awarded to the winner of an event.'

25 rows selected. (0.560000 sec) Committed.

1 command(s) successfully processed.
csql>
```

SQL Interpreter (csql) - 계속

- SQL 수행
  - SQL buffer 사용 (질의 끝에 ; 추가하지 않음)
    - SQL 문을 buffer 에 넣고, buffer 의 내용 수행
    - 입력되는 내용은 buffer 에 자동 저장
  - 명령어 (대문자 부분만 입력 가능, 대소문자 구분 없음)
    - ;RUn : buffer 내용 수행
    - ;Xrun : buffer 내용 수행 후 buffer 지움
    - ;EDIT : buffer 내용 편집 (LINUX 의 경우 vi 사용)
    - ;CLear : buffer 내용 삭제
  - 예시

```
csql> select * from olympic
csql> ;run
...
 1900  'France'              'Paris'               05/14/1900    10/28/1900    NULL                  NULL                  'The
1900 Paris Games allowed women to compete and a total of nineteen countries participated at the Games. The
Games were held at the same time as the World Exhibition in Paris and were dragged out over five months. As a
result the 1900 Paris Games were a flop.'
         1896  'Greece'              'Athens'              04/06/1896    04/15/1896    NULL                  NULL
'Thirteen countries participated at the 1896 Athens Games with around 300 athletes taking part in the competition.
There were 43 events contested which fell into the following categories; athletics (track and field), cycling, fencing,
gymnastics, shooting, swimming, tennis, weightlifting and wrestling. Competition was open to any man who
wanted to participate. All medals were silver and they were only awarded to the winner of an event.'

25 rows selected. (0.560000 sec) Committed.

1 command(s) successfully processed.
csql>
```

SQL Interpreter (csql) - 계속

- SQL 수행
  - SQL 튜닝에 필요한 SQL 수행계획 및 통계 정보 확인 기능
    - 뒤 튜닝 부분에서 다룸
  - 명령어 history 기능
    - ↑ ↓ 키를 이용하여 사용했던 명령어(SQL 및 csql 명령어) 재사용
    - 단, line 제한이 있어 SQL이 길 경우 잘릴 수 있으므로, 이를 방지하기위해 buffer 사용 권장

- 테이블 정보 확인

```
-- record 테이블에 대한 정보(컬럼,인덱스 등)를 확인한다.
csql> ;SChema record
```

- 트리거 정보 확인

```
-- 현재 접속한 데이터베이스의 모든 트리거 확인
csql> ;TRIgger
-- tr1 이라는 이름의 트리거에 대한 정보 확인
csql> ;TRIgger tr1
```

- CSQL 종료

```
csql> ;EXit
```

8. 사용자와 권한

사용자 생성 및 암호 변경

- DBA(관리자) 와 PUBLIC(일반사용자) 계정이 기본적으로 제공된다.
  - PUBLIC 계정은 삭제 불가능하며, 암호 설정후 사용하지 않을 것을 권장함.
- DBA, DBA 그룹만 사용자 생성/삭제 가능하다.
- 사용자 명은 대소문자 구분하지 않으나, 암호는 대소문자 구분함
- 사용자 계정명은 변경할 수 없으나, 암호는 변경 가능

```
-- db_user 계정의 암호를 user#pass 로 설정하여 계정 생성
create user db_user password 'user#pass'

-- db_user 계정의 암호를 passWd 로 변경
alter user db_user password 'passWd'
```

권한 관리

- 테이블에 대한 사용자 계정별 접근 권한 관리
  - 권한 부여 및 제거는 DBA, DBA 권한 계정, 테이블 권한을 가지고 있는 계정만 가능
- 권한 부여

```
-- 테이블 nation과 athlete 에 대한 select, update 권한을 db_user 계정에 부여
grant select, update on nation, athlete to db_user

-- 테이블 record 에 대한 모든 권한(권한 부여까지) 을 db_user 계정에 부여
grant all privileges on record to db_user with grant option
```

권한 관리 - 계속

- 권한 제거

```
-- db_user 계정이 가지고 있던 테이블 nation과 athlete 에 대한 select, update 권한을 제거
revoke select, update on nation, athlete from db_user

-- db_user 계정이 가지고 있던 테이블 record 에 대한 모든 권한을 제거
revoke all privileges on record from db_user
```

소유자 변경

- 테이블에 대한 소유자 변경을 할 수 있다.
  - 데이터베이스 관리자(**DBA**) 또는 **DBA** 그룹의 멤버만 가능
  - 테이블, 뷰, 트리거, Stored Procedure/Function 에 대한 소유자 변경이 가능

```
-- record 테이블에 대한 소유자를 db_user 로 변경
alter table record owner to db_user

-- record 테이블에 대한 소유자를 dba 로 변경
alter table record owner to dba
```

사용자 삭제

- 사용자 삭제 시 소유한 테이블이 없어야 삭제 가능

```
-- db_user 계정 삭제
drop user db_user

-- db_user 계정 삭제시 소유한 테이블이 있는 경우 에러 발생
drop user db_user
ERROR: Cannot drop the user who owns database objects(class/trigger/serial etc).
```

9. DDL

기본 사항

- 테이블
  - 동일한 데이터베이스에서 테이블명은 중복 될 수 없다
  - 생성 가능한 테이블의 개수 제한은 없다.
  - 테이블 명에 대한 대소문자 구분은 없으며, 최대 254자까지 사용할 수 있다.
  - 이름에 한글, 영문자, 숫자, _, # 를 사용할 수 있으며, 첫 글자는 반드시 문자(한글/영문)여야 한다.
  - 레코드의 개수 제한은 없다.
- 컬럼
  - 컬럼 명에 대한 대소문자 구분은 없으며, 최대 254자까지 사용할 수 있다.
  - 이름에 한글, 영문자, 숫자, _, # 를 사용할 수 있으며, 첫 글자는 반드시 문자(한글/영문)여야 한다.
- 인덱스
  - Primary key와 Foreign key 는 인덱스로도 사용되므로, key 생성후 별도로 인덱스 만들 필요 없다.
- 예약어 사용 시 [] 또는 “”로 감싸주어야 한다.

```
select [name] from nation
```

- 데이터베이스의 기본 문자셋과 다르게 테이블/컬럼 별 문자셋을 지정할 수 있다.
  - 지원 : iso88591, UTF8, EUC-KR

기본 사항 - 계속

- 제약조건
  - NOT NULL
  - UNIQUE
  - DEFAULT
  - Primary Key : 모든 컬럼에 대하여 NULL 허용 상관없이 NOT NULL 로 설정된다.
  - Foreign Key
    - 참조대상의 값이 있거나 모두 NULL 이어야 한다.
      - multi column 인 경우 어느 한 컬럼만 NULL 인 경우는, 기본키가 모두 NOT NULL 이므로 허용이 되지 않는다.
    - action
      - on update : no action, restrict, set null
      - on delete : no action, restrict, set null, cascade
    - option
      - restrict/no action: 기본키의 삭제/변경 제한
      - set null: 기본키가 삭제/변경되면 참조하는 외래키의 값을 null 로 변경
      - cascade: 기본키가 삭제되면 참조하는 외래키도 삭제
  - check 지원하지 않으므로, 응용에서 처리하거나 trigger 를 사용해야 한다.
- dummy table (dual)

```
select sysdatetime from dual
select sysdate
select sysdate from db_root
```

데이터 타입

- 숫자형
  - unsigned 지원 안 함.
  - smallint: 2bytes, -32768 ~ 32767
  - integer: 4bytes, -2147483648 ~ 2147483647
  - bigint: 8bytes, -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
  - numeric: numeric(p,s), 1<= p <= 38, 0 <= s <= p
  - float: -3.402823466E+38 ~ -3.402823466E+38 (유효자리: 7)
  - double: - 1.7976931348623157E+308 ~ 1.7976931348623157E+308 (유효자리: 15)
  - float, double은 유효 자리 수를 초과하는 경우 값이 달라질 수 있다.
  - **'' 값 입력 시 0으로 입력되나 numeric 은 NULL 로 입력됨.**
- 문자형
  - char : 고정 길이형. 1 ~ 256M (CUBRID 10.x 까지는 1G)
    - 문자열 저장시 남는 공간은 오른쪽에 공백문자가 채워진다.
      예) 컬럼 길이가 10, 'CUBRID' 입력하면 'CUBRID□□□□'로 입력됨 (□는 공백)
  - varchar : 가변 길이형, 1 ~ 1G
    - 문자열과 문자열 길이가 같이 저장. update 시 길이가 변하는 경우 저장 위치 변경으로 overhead 발생.
    - 문자열 끝의 공백도 문자열로 본다. 단, CUBRID 10.x 까지는 공백은 절삭한다.
  - **입력되는 문자열의 길이가 컬럼 길이를 초과하면 에러 발생**
    - CUBRID 10.x 까지는 절삭되어 저장된다. 예) varchar(4) 에 'cubrid' 저장하면 'cubr' 만 저장.
  - 오라클 LONG 타입은 varchar(1G) 로 사용.
    - string 은 varchar(1G) 의 alias 이다. 1G 를 숫자 1073741823 로 표현하기 어려워서 만들어졌다.
  - '' (empty string) ≠ NULL
    - NULL 값에 대한 비교는 is null 또는 is not null 을 이용해야 한다.

데이터 타입 - 계속

- 날짜/시간형
  - '' 값 입력 시 에러, NULL 또는 표현 가능한 값을 입력해야 한다.
  - date: 날짜만 저장, 기본형 mm/dd/yyyy
  - time: 시간만 저장, 기본형 hh:mi:ss
  - datetime: 1/1000초, 'mm/dd[/yyyy] hh:mi[:ss.ff]' 또는 '[yyyy-]mm-dd hh:mi[:ss.ff]'
  - timestamp:1970/01/01 00:00:00 (GMT) 이후 경과한 초, 최대 2038/01/19 03:14:07 (4 byte)

- ENUM
  - 열거형 데이터 타입은 문자열 상수들의 중복 없는 순서 집합으로 구성되어진다.
  - 최대 512개를 가질 수 있다.
  - 색인만 저장함으로 2바이트 저장 공간 사용

- 비트열
  - 고정길이 비트열(BIT), 가변길이 비트열(BIT VARYING) 이 있으며 최대 1G까지 가능하다.
  - 비트열은 0과 1로 이루어진 이진 값이며, 1G는 파일 기준 128M-1byte 의 크기이다.
  - 타입 지정시 bit varying 으로 길이 없이 지정하면 bit varying(1G) 로 생성된다.

- LOB
  - BLOB/CLOB 을 저장하며, 크기 제한은 없다.
  - 메타 정보를 데이터베이스에 보관하며, 실 데이터는 <database directory>/lob 아래 파일로 저장된다.

CUBRID

테이블 생성

- 일반
  - 테이블 및 컬럼 주석 설정 가능
  - REUSE_OID: 데이터 볼륨의 한번 사용한 공간을 재사용하는 설정으로 CUBRID11.0 이후 기본 적용된다.
    - delete/insert 가 많은 경우 볼륨 공간 사용 효율이 좋아진다.
  - DONT_REUSE_OID: 사용한 공간을 재사용하지 않는 설정으로, OID 를 이용한 연산(예, update object, delete object) 을 사용하는 경우엔 설정을 해주어야 한다.

```
-- 일부 주요 사항에 대하여만 설명

-- 테이블 user_tbl 에 '사용자 정보 테이블' 이라는 주석을 넣고, OID를 재사용하지 않도록 생성한다.
-- user_id 를 정수 값의 기본키로 설정하고, user_name 은 NULL 값을 허용하지 않고 중복된 값도 가지면 않된다.
-- age 의 기본값은 0으로 하고 nation_code 에 대한 주석을 넣는다.
-- 한 사람(user_name) 이 여러 국가(nation_code) 에 속하지 않도록 설정한다.
   -- user_name 이 unique 여서 user_name 에 nation_code 가 중복되지는 않지만, 다중 컬럼 unique 설정 참고용
-- nation_code 는 nation 테이블의 code 를 참조하고, nation 에서 삭제시 같이 삭제되도록 설정한다.
create table user_tbl (
    user_id int,
    user_name char(10) not null unique,
    addr varchar(50) null,
    age int not null default 0,
    nation_code char(3) comment '국적',
    unique(user_name, nation_code),
    constraint pk_id primary key(user_id),
    constraint fk_nation foreign key (nation_code) references nation(code) on delete cascade
) comment '사용자 정보 테이블' dont_reuse_oid [ charset euckr ]
```

user_id int constraint pk_id primary key(user_id) : 지원 안됨

테이블 생성시 데이터베이스 설정과 다른 문자셋을 지정할 수 있다.
단, 참조하는 테이블과 동일한 문자셋이어야 한다.
다를 경우 foreign key 생성 오류가 발생한다.

테이블 생성 - 계속

- 스키마 복제
  - 자동 증가 속성이 없는 경우만 사용 가능
  - Constraints, 인덱스 동일한 이름으로 복사된다.

```
-- user_tbl 을 동일하게 복제하여 tml_user_tbl 을 만든다.
create table tmp_user_tbl like user_tbl
```

- 스키마, 데이터 복제
  - create 절에 없는 select 절의 필드는 자동 생성되며, select 절에 없는 필드엔 null 값이 들어간다.
  - Constraints, 인덱스는 복사되지 않으므로 별도로 생성해야 한다.
  - 타입이 다른 경우 자동 형 변환되어 입력되며, 중복되는 값이 있는 경우 중복에러가 발생한다.
    - replace 옵션을 사용하면 기존 값을 변경하면서 입력된다.

```
-- user_tbl 과 동일하게 tbl 을 만들고, user_tbl 모든 컬럼 데이터를 tbl 에 입력한다.
create table tbl as select * from user_tbl

-- tbl1 을 id, name 컬럼으로 만들고, 이중 id 는 user_tbl 의 user_id 값으로 입력한다. name 컬럼은 우선 비워둔다.
create table tbl1 ( id int, name char(10) ) as select user_id from user_tbl

-- tbl2 를 중복되지 않는 nation_code 컬럼으로 만들고, 이 값은 user_tbl 의 값으로 입력한다.
create table tbl2 ( nation_code char(3) unique) replace as select nation_code from user_tbl
```

키 추가

- 기본키 추가

```
-- tbl 에 id 를 기본키로 추가하고 이름을 pk_id 로 만든다.
alter table tbl add constraint pk_id primary key(user_id)
```

- 외래키 추가

```
-- tbl2 의 nation_code 컬럼이 nation 테이블의 code 를 참조하게 하고, 그 이름을 fk_code 로 만든다.
alter table tbl2 add constraint fk_code
                foreign key(nation_code)  references nation(code) on delete cascade
```

- UNIQUE 추가

```
-- tbl 에 사용자 이름과 국적이 중복되지 않도록 unique 를 설정하고, 그 이름을 u_name 으로 한다.
create unique index u_name on tbl(user_name, nation_code)
-- 또는 다음과 같이 만들 수 있다.
alter table tbl add constraint u_name unique (user_name, nation_code)
```

키 삭제

- 기본키 삭제

```
-- tbl 의 기본키를 삭제한다.
alter table tbl drop primary key
-- 또는 다음과 같이 constraint name pk_id 를 이용하여 삭제할 수 있다.
alter table tbl drop constraint pk_id
```

- 외래키 삭제

```
-- tbl2 의 외래키 fk_code 를 삭제한다.
alter table tbl2 drop constraint fk_code
```

- UNIQUE 삭제

```
-- tbl 의 unique u_name 을 삭제한다.
alter table tbl drop constraint u_name
```

인덱스

- 컬럼별 오름차순/내림차순 지정 가능
  - 검색 조건 및 정렬 조건에 적절히 활용할 경우, 인덱스를 이용한 검색 순서와 정렬 순서가 같은 경우 정렬을 하지 않게 되어 성능을 개선 할 수 있다.
  - 오름차순으로만 생성하여도 정렬시 내림차순일 경우 해당 인덱스 사용 가능. 단, 모든 컬럼을 내림차순으로 정렬할 경우만 사용된다.
- 컬럼에 대한 특정 조건의 값들만 인덱스 구성 가능
- 일부 함수에 대하여 함수 결과를 인덱스로 만드는 function based 인덱스 생성 가능
  - upper(), lower(), to_char(), to_date(), nvl(), trim() 등

```
-- 예를 위한 것으로, 예시의 모든 경우를 위해 반드시 모든 인덱스가 필요하지는 않습니다.

-- tbl 테이블 검색시 user_name 과 age 를 이용한 검색을 위해 인덱스 idx1 생성
create index  idx1 on tbl(user_name, age)
-- tbl 테이블에 대하여 age 또는 nation_code 로 검색을 위해 인덱스 생성
alter table tbl add index idx2(age), add index idx3(nation_code)
-- tbl 테이블에 대하여 user_name 과 age 로 검색하면서 user_name 은 순방향에 나이 많은 순으로 검색을 위해 인덱스 생성
create index idx4 on tbl(user_name, age desc)
-- tbl 테이블에 대하여  age 가 20 미만일 경우만 주로 검색하여 이를 위한 인덱스 생성
create index idx5 on tbl(age) where age < 20
-- tbl 테이블에 대하여 user_name 을 소문자로만 검색하는 경우를 위한 인덱스 생성
create index idx6 on tbl(lower(user_name))
```

인덱스 - 계속

- 인덱스 재구성

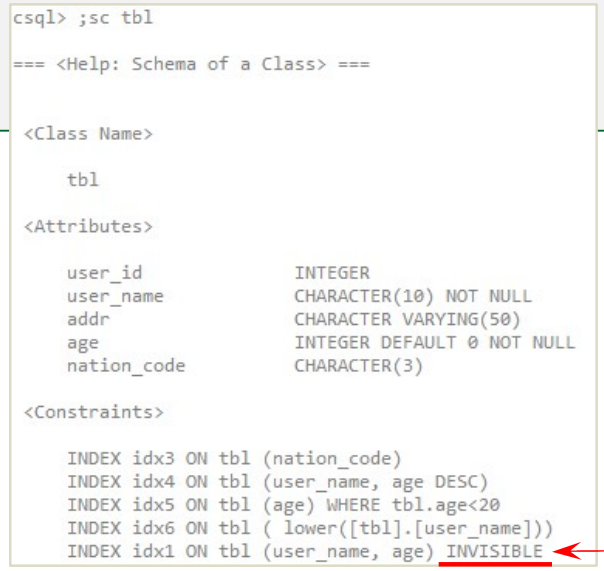
```
-- tbl 의 인덱스 idx1 을 재구성한다.
alter index idx1 on tbl rebuild
```

- 인덱스 삭제

```
-- tbl 의 인덱스 idx2 를 삭제한다.
drop index idx2 on tbl
-- 또는
drop index idx2 on tbl(age)
```

- 인덱스 숨김 및 숨김 해제
  - 인덱스를 숨겨 해당 인덱스가 없는 것처럼 SQL 이 동작하게 한다.

```
-- tbl 의 인덱스 idx1 를 숨겨 SQL 수행시 참조하지 않도록 한다. 테이블 정보에서 INVISIBLE 로 보인다.
alter index idx1 on tbl invisible
-- idx 을 다시 보이도록 한다.
alter index idx1 on tbl visible
```



```
csql> ;sc tbl

=== <Help: Schema of a Class> ===


 <Class Name>

     tbl

 <Attributes>

     user_id              INTEGER
     user_name            CHARACTER(10) NOT NULL
     addr                 CHARACTER VARYING(50)
     age                  INTEGER DEFAULT 0 NOT NULL
     nation_code          CHARACTER(3)

 <Constraints>

     INDEX idx3 ON tbl (nation_code)
     INDEX idx4 ON tbl (user_name, age DESC)
     INDEX idx5 ON tbl (age) WHERE tbl.age<20
     INDEX idx6 ON tbl ( lower([tbl].[user_name]))
     INDEX idx1 ON tbl (user_name, age) INVISIBLE
```


CUBRID

테이블 변경

- 이름 변경

```
-- tml_user_tbl 을 user_tbl_bak 로 변경한다.
rename table tmp_user_tbl as user_tbl_bak
```

- 주석 변경

```
-- user_tbl_bak 의 주석을 '사용자 테이블 백업' 으로 변경한다.
alter table user_tbl_bak comment = '사용자 테이블 백업'
```

- 삭제

```
-- 테이블 tbl1, tbl2 를 삭제한다.
drop table tbl1, tbl2
```

테이블 컬럼 변경

- 컬럼 추가

```
-- tbl 테이블에 가변문자열 20자리의 컬럼 phone 을 추가한다. 테이블의 컬럼 목록 제일 뒤에 추가된다.
alter table tbl add column phone varchar(20)
-- tbl 테이블 컬럼 목록 제일 앞부분에 기본값 '' 을 가지는 컬럼 nick_name 을 추가한다.
alter table tbl add column nick_name char(10) default '' first
-- tbl 테이블 user_name 컬럼 다음에 birthday 를 추가한다
alter table tbl add column birthday date after user_name
```

- 컬럼 이름 및 속성 변경

```
-- tbl 테이블phone 컬럼의 이름을 mobile 로 변경한다.
alter table tbl rename column phone as mobile
-- tbl 테이블 birthday 컬럼의 기본값을 sysdate 로 변경한다.
alter table tbl alter column birthday set default sysdate
-- tbl 테이블 mobile 컬럼을 mobile_phone 으로 변경하고 길이를 varchar(15)로 변경한다.
alter table tbl change mobile mobile_phone varchar(15)
-- tbl 테이블 nick_name 컬럼 타입을 varchar(50)으로 변경한다.
alter table tbl modify nick_name varchar(50)
```

  - 컬럼 크기를 줄이는 경우 기존 값이 줄어드는 값보다 긴 경우 에러 발생.
  - change/modify 로 컬럼 속성을 변경 가능하며, change 의 경우 이름까지 변경할 수 있다.

- 컬럼 삭제

```
-- tbl 테이블의 nickname 컬럼을 삭제한다.
alter table tbl drop column nick_name
```

일련번호 - 계속

- serial
  - 일련번호를 생성하는 데 사용되며, 트랜잭션의 영향을 받지 않는다.
    - 트랜잭션이 롤백되더라도 증가된 serial 값은 줄어들지 않는다.
  - 게시물 번호와 같이 중간에 번호가 없어도 상관없는 경우 사용한다.
  - 기본적으로 최대값은 10^{37}이다.
    - 최대값 도달 후 다음 값은 nocycle 인 경우 에러가 발생한다. cycle 인 경우 minvalue 부터 다시 시작한다.

```
-- seq 라는 이름을 가진 serial 을 생성한다.
create serial seq
-- 100 부터 시작해서 10씩 증가하며 최대 100000 까지의 값을 얻을 수 있는 serial seq1 을 생성한다. 최대값에 도달하면 10부터 다시 시작한다.
create serial seq1 start with 10 increment by 10 minvalue 10 maxvalue 100000 cycle

-- seq 의 최대값을 10000으로,  증가값은 10으로 조정한다.
alter serial seq increment by 10 maxvalue 10000

-- seq1 의 다음 값을 얻어온다. 번호 입력시 사용하는 일반적인 방법이다.
select seq1.nextval

-- seq1 의 현재값을 확인하고, 현재값을 1000으로 변경하고, 다음 값을 얻어온다.
select seq1.currval
alter serial seq1 start with 1000
select seq1.nextval

-- seq 를 삭제한다.
drop serial seq
```

변경한 값부터 다시 가져온다

```
csql> select seq1.currval;

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

  serial_current_value(seq1)
======================
  10

1 row selected. (0.004524 sec) Committed.

1 command(s) successfully processed.
csql> alter serial seq1 start with 1000;
Execute OK. (0.000746 sec) Committed.

1 command(s) successfully processed.
csql> select seq1.nextval;

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

  serial_next_value(seq1, 1)
======================
  1000

1 row selected. (0.005092 sec) Committed.
```

자동증가

- auto increment
  - 레코드 입력 시 설정된 컬럼의 값이 기본 1씩 자동으로 증가하며, 임의의 값을 입력할 수도 있다.
    - 단, 임의로 입력된 값은 자동 증가되는 값과 중복될 수 있다.
  - 테이블에 하나의 컬럼에만 설정 가능하며, 정수형 타입(smallint, int, bigint, 소수점없는 numeric) 만 가능
  - 설정한 타입의 최대값을 넘어가는 경우 에러 발생
  - serial 사용과 유사하다. 즉 롤백되더라도 숫자는 1만큼 증가되어 있다.

```
-- 1부터 시작해서 1씩 자동으로 증가하는 컬럼을 가진 테이블 ai_tbl 을 만든다.
create table ai_tbl (
  id int auto_increment,
  name varchar(10)
)
-- 10 부터 시작해서 2씩 자동으로 증가하는 컬럼을 가진 테이블 ai_tbl2 를 만든다.
create table ai_tbl2 (
  id int auto_increment(10, 2),
  name varchar(10)
)
```

자동증가 - 계속

```
csql> insert into ai_tbl(name) values('name1');
1 row affected. (0.002581 sec) Committed.

1 command(s) successfully processed.
csql> insert into ai_tbl(name) values('name2');
1 row affected. (0.002034 sec) Committed.

1 command(s) successfully processed.
csql> select * from ai_tbl;

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

            id  name
=====================================
             1  'name1'
             2  'name2'

2 rows selected. (0.006653 sec) Committed.

1 command(s) successfully processed.
csql> insert into ai_tbl values(3, 'name3');
1 row affected. (0.001044 sec) Committed.

1 command(s) successfully processed.
csql> insert into ai_tbl(name) values('name4');
1 row affected. (0.001675 sec) Committed.

1 command(s) successfully processed.
csql> select * from ai_tbl;

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

            id  name
=====================================
             1  'name1'
             2  'name2'
             3  'name3'
             3  'name4'

4 rows selected. (0.004386 sec) Committed.
```

id 컬럼은 자동 증가되므로 name 컬럼만 값을 입력한다.

id 는 1, 2 로 1씩 자동 증가되는 값으로 입력되었다.

id 에 3의 값을 넣어서 한 건 입력한다.

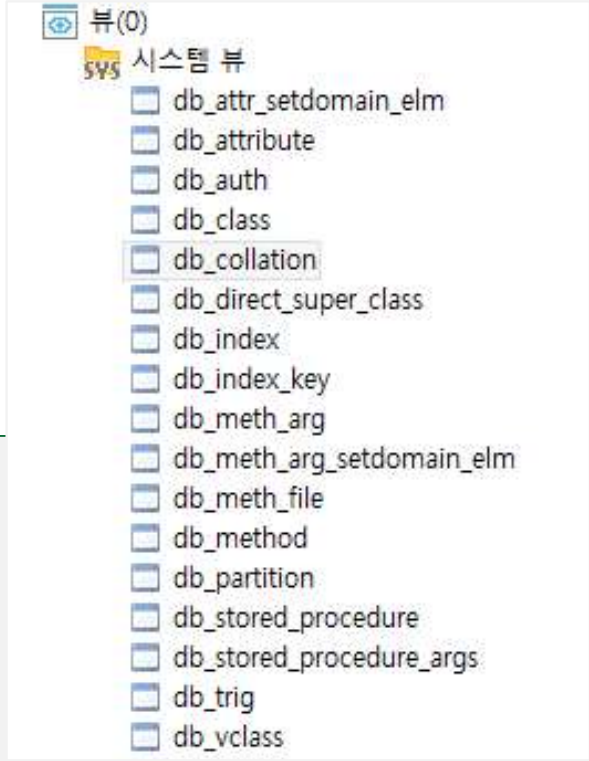
name 에만 값을 넣어서 한 건 입력한다.

검색결과 직접 값을 넣은 id 3은 자동증가로 인식되지 못해, 그 뒤에 넣은 name4 의 레코드는 이전에 자동 증가된 값 2에서 자동 증가하여 3이 입력된 것을 알 수 있다.

CUBRID

카달로그

- 스키마 정보를 가진 시스템 테이블
  - 테이블/컬럼 명: 소문자, 소유자 명: 대문자로 저장
  - 테이블 정보 : db_class
    - 테이블명: class_name, 소유자: owner_name
  - 컬럼 정보 : db_attribute
    - 테이블명: class_name, 컬럼명 : attr_name, 데이터타입: data_type



- 카달로그 활용 예

```
-- Primary key 없는 테이블 검색
select owner_name, class_name
from  db_class
where class_type = 'CLASS' and is_system_class = 'NO'
and class_name not in
  ( select class_name from db_index  where is_primary_key ='YES'  )
order by owner_name, class_name

-- LOB 를 사용하는 테이블/컬럼 검색
select class_name, attr_name, data_type
from  db_attribute
where data_type like '%LOB'
order by class_name, attr_name
```


CUBRID

SHOW

- 스키마 정보 조회

```
-- 생성된 모든 테이블 목록 확인
show tables
-- 테이블 이름이 user 로 시작하는 모든 테이블 확인 (테이블명은 소문자로 되어 있음. 대문자로는 검색않됨)
show tables like 'user%'

-- user_tbl 의 모든 컬럼 확인 (아래 두 SQL 수행 결과는 같다)
show columns from user_tbl
desc user_tbl

-- user_tbl 의 모든 인덱스 확인
show indexes from user_tbl

-- user_tbl 에 대한 테이블 생성 명령어 확인
show create table user_tbl
```

실습

1. demodb 에 사용자 dbuser 를 암호없이 생성한다.

```
[cubrid@db1 ~]$ csql -u dba demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.

csql> create user dbuser;
Execute OK. (0.010875 sec) Committed.
```

2. 사용자 dbuser 의 암호를 'passWd' 로 변경한후, dbuser 계정으로 로그인한다.

```
csql> alter user dbuser password 'passWd';
Execute OK. (0.001364 sec) Committed.

1 command(s) successfully processed.
csql> ;ex
[cubrid@db1 ~]$ csql -u dbuser -p passwd demodb

ERROR: Incorrect or missing password.

[cubrid@db1 ~]$ csql -u dbuser -p passWd demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.
```

암호에 대하여 대소문자 구분됨을 알 수 있다.

실습 - 계속

3. 다음의 조건으로 테이블을 생성하고, 생성된 테이블 정보를 확인한다.

테이블명: user_info
컬럼1: 이름:ID, 정수형으로 기본키
컬럼2: 이름:name, 최대 50자리의 가변 문자열로 반드시 값이 있어야 하고, 중복되어도 않된다.
컬럼3: 이름:phone, 최대 50자리의 가변문자열로 값이 없을 수도 있다.
컬럼4: 이름:age, 정수형으로 반드시 값이 있어야 하며, 초기값은 0 이다.
컬럼5: 이름:birth, 날자형으로 반드시 값이 있어야 하며, 초기값은 오늘 이다.
나이로 검색할 수 있고, 이름과 생일을 같이 검색할 수 있으니 관련 인덱스가 필요하다.

```
csql> create table user_info (
csql> id int primary key,
csql> name varchar(50) not null unique,
csql> phone varchar(50),
csql> age int not null default 0,
csql> birth date not null default sysdate,
csql> index idx1(age)
csql> )
csql> ;ru
Execute OK. (0.039191 sec) Committed.
```

```
csql> ;cl
csql> create index idx2 on user_info(name, birth)
csql> ;x
Execute OK. (0.012681 sec) Committed.
```

```
csql> ;sc user_info

=== <Help: Schema of a Class> ===


 <Class Name>

     user_info

 <Attributes>

     id                  INTEGER NOT NULL
     name                CHARACTER VARYING(50) NOT NULL
     phone               CHARACTER VARYING(50)
     age                 INTEGER DEFAULT 0 NOT NULL
     birth               DATE DEFAULT SYS_DATE NOT NULL

 <Constraints>

     PRIMARY KEY pk_user_info_id ON user_info (id)
     UNIQUE u_user_info_name ON user_info (name)
     INDEX idx1 ON user_info (age)
     INDEX idx2 ON user_info (name, birth)
```

실습 - 계속

4. demodb 에 사용자 dbuser2 를, dbuser2 를 암호로 하여 생성한다.

```
[cubrid@db1 ~]$ csql -u dba demodb -c "create user dbuser2 password 'dbuser2';"
Execute OK. (0.004152 sec) Committed.
```

5. demodb 에 dbuser2 로 접속하여 user_info 테이블을 검색해본다.

```
select * from user_info
```

```
[cubrid@db1 ~]$ csql -u dbuser2 -p dbuser2 demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.

csql> select * from user_info;

In line 1, column 24,

ERROR: SELECT is not authorized on user_info.
```

6. dbuser2 에 user_info select 권한을 부여하고, dbuser2 사용자로 다시 검색해 본다.

```
[cubrid@db1 ~]$ csql -u dbuser -p passWd demodb -c "grant select on user_info to dbuser2;"
Execute OK. (0.001850 sec) Committed.
[cubrid@db1 ~]$ csql -u dbuser2 -p dbuser2 demodb -c "select * from user_info;"

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

There are no results.
0 row selected. (0.009593 sec) Committed.
```

실습 - 계속

7. 다음의 조건으로 테이블을 생성하고, 생성된 테이블 정보를 확인한다.
   1. dba 계정으로 생성후, dbuser 를 소유자로 변경한다.

테이블명: dept_info
컬럼1: 이름:ID, 정수형으로 기본키
컬럼2: 이름:name, 최대 50자리의 가변 문자열로 반드시 값이 있어야 하고, 중복되어도 않된다.
컬럼3: 이름:manager, 정수형으로 반드시 값이 있어야 한다. 사용자 ID 로 user_info 를 참조한다.
컬럼4: 이름:manager_sub, 정수형으로 반드시 값이 있어야 한다. 사용자 ID 로 user_info 를 참조한다.
부서의 부서장(manager)과 부-부서장(manager-sub) 는 같은 2명이 한 개의 부서만 맡을 수 있다. 즉, 1명이 여러 부서의 부서장이나 부-부서장을 맡을 수는 있지만, 동일한 부서장과 동일한 부-부서장이 2개의 부서를 맡을 수는 없다.

```
[cubrid@db1 ~]$ csql -u dba demodb -c "alter table dept_info owner to dbuser;"
Execute OK. (0.001877 sec) Committed.
[cubrid@db1 ~]$ csql -u dbuser -p passWd demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.

csql> ;sc dept_info

=== <Help: Schema of a Class> ===


 <Class Name>

     dept_info

 <Attributes>

     id                   INTEGER NOT NULL
     name                 CHARACTER VARYING(50) NOT NULL
     manager              INTEGER NOT NULL
     manager_sub          INTEGER NOT NULL

 <Constraints>

     PRIMARY KEY pk_dept_info_id ON dept_info (id)
     UNIQUE u_dept_info_name ON dept_info (name)
     UNIQUE u_dept_info_manager_manager_sub ON dept_info (manager, manager_sub)
     FOREIGN KEY fk1 ON dept_info (manager) REFERENCES user_info ON DELETE RESTRICT, ON UPDATE RESTRICT
     FOREIGN KEY fk2 ON dept_info (manager_sub) REFERENCES user_info ON DELETE RESTRICT, ON UPDATE RESTRICT
```

```
[cubrid@db1 ~]$ csql -u dba demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.

csql> create table dept_info (
csql> id int primary key,
csql> name varchar(50) not null unique,
csql> manager int not null,
csql> manager_sub int not null,
csql> unique(manager, manager_sub),
csql> constraint fk1 foreign key (manager) references user_info(id),
csql> constraint fk2 foreign key (manager_sub) references user_info(id) );
Execute OK. (0.023323 sec) Committed.

1 command(s) successfully processed.
csql> ;ex
[cubrid@db1 ~]$ csql -u dbuser -p passWd demodb

        CUBRID SQL Interpreter


Type `;help' for help messages.

csql> ;sc dept_info

ERROR: SELECT authorization failure.
```

실습 - 계속

7. ...

```
csql> show columns from dept_info;

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

  Field                 Type                  Null                  Key                   Default               Extra

=============================================================================================================================
============
  'id'                  'INTEGER'             'NO'                  'PRI'                 NULL                  ''

  'name'                'VARCHAR(50)'         'NO'                  'UNI'                 NULL                  ''

  'manager'             'INTEGER'             'NO'                  'UNI'                 NULL                  ''

  'manager_sub'         'INTEGER'             'NO'                  'MUL'                 NULL                  ''


4 rows selected. (0.005604 sec) Committed.

1 command(s) successfully processed.
csql> show indexes from dept_info;

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

  Table                 Non_unique  Key_name              Seq_in_index  Column_name           Collation             Car
dinality     Sub_part  Packed                Null                  Index_type            Func                  Comment
          Visible
=============================================================================================================================
=============================================================================================================================
================================
  'dept_info'                    1  'fk1'                            1  'manager'             'A'
       0         NULL  NULL                  'NO'                  'BTREE'               NULL                  NULL
          'YES'
  'dept_info'                    1  'fk2'                            1  'manager_sub'         'A'
       0         NULL  NULL                  'NO'                  'BTREE'               NULL                  NULL
          'YES'
  'dept_info'                    0  'pk_dept_info_id'               1  'id'                  'A'
       0         NULL  NULL                  'NO'                  'BTREE'               NULL                  NULL
          'YES'
  'dept_info'                    0  'u_dept_info_manager_manager_sub'            1  'manager'             'A'
                 0         NULL  NULL                  'NO'                  'BTREE'               NULL
    NULL                'YES'
  'dept_info'                    0  'u_dept_info_manager_manager_sub'            2  'manager_sub'         'A'
                 0         NULL  NULL                  'NO'                  'BTREE'               NULL
    NULL                'YES'
  'dept_info'                    0  'u_dept_info_name'              1  'name'                'A'
       0         NULL  NULL                  'NO'                  'BTREE'               NULL                  NULL
          'YES'
```


CUBRID

실습 - 계속

8. 사용자 정보에서 생일을 시간까지 포함하도록 변경한다.

```
csql> alter table user_info modify birth datetime;
Execute OK. (0.010161 sec) Committed.

1 command(s) successfully processed.
csql>
csql> ;sc user_info

=== <Help: Schema of a Class> ===


 <Class Name>

     user_info

 <Attributes>

     id                  INTEGER NOT NULL
     name                CHARACTER VARYING(50) NOT NULL
     phone               CHARACTER VARYING(50)
     age                 INTEGER DEFAULT 0 NOT NULL
     birth               DATETIME

 <Constraints>

     PRIMARY KEY pk_user_info_id ON user_info (id)
     UNIQUE u_user_info_name ON user_info (name)
     INDEX idx1 ON user_info (age)
     INDEX idx2 ON user_info (name, birth)
```

```
csql> alter table user_info modify birth datetime not null default sysdatetime;
Execute OK. (0.006422 sec) Committed.

1 command(s) successfully processed.
csql> ;sc user_info

=== <Help: Schema of a Class> ===


 <Class Name>

     user_info

 <Attributes>

     id                  INTEGER NOT NULL
     name                CHARACTER VARYING(50) NOT NULL
     phone               CHARACTER VARYING(50)
     age                 INTEGER DEFAULT 0 NOT NULL
     birth               DATETIME DEFAULT SYS_DATETIME NOT NULL

 <Constraints>

     PRIMARY KEY pk_user_info_id ON user_info (id)
     UNIQUE u_user_info_name ON user_info (name)
     INDEX idx1 ON user_info (age)
     INDEX idx2 ON user_info (name, birth)
```


CUBRID

10. DML

연산자

- 논리연산자

AND, &&	OR, ||	XOR	NOT, !

- 비교연산자

=	<=>	!=, <>	<	>	<=	>=	is [not] <true | false | null>

- 산술연산자

+	-	*	/, DIV	%, MOD

  - 숫자형 연산
    - 정수와 정수 연산시 정수형 결과. 예) 1/2 = 0 (0.5 에서 소수점 이하 버림)
    - 다른 타입의 숫자 연산시 큰 타입으로 형 변환. 예) 1/2.0 = 0.500000 (정수와 실수 연산으로 실수로 형변환됨)
  - 문자형과 숫자형 연산
    - double 타입으로 형 변환. 예) 1 + '1' = 2.000000000000000e+00

연산자 - 계속

- 산술연산자
  - 날짜/시간형 연산

연산 형태	설명	예시 (결과는 SQL 수행 결과)
time + time, date + date	지원하지 않음	
time - time = int	초 단위 두 시간의 간격	time '23:30:30' - time '23:29:00' = 90
time ± int = time	몇 초 전/후	time '23:30:30' + 90 = 11:32:00 PM
date – <date | timestamp | datetime> = int	일자 단위 두 날자의 간격	date '2022-12-01' - date '2022-12-25' = -24
date ± int = date	몇 일 전/후	date '2022-12-01' + 24 = 12/25/2022
timestamp – <date | timestamp | datetime> = int	두 일시의 간격(초 또는 1/1000초). date 일 경우 시각은 00:00:00로 본다.	systimestamp - sysdate = 39066 systimestamp - sysdatetime = -799
datetime – <date | timestamp | datetime> = int	두 일시의 1/1000초 간격. date 일 경우 시각은 00:00:00로 본다.	sysdatetime - sysdate = 39066799 sysdatetime - systimestamp = 799
datetime ± int = datetime	일시 기준 1/1000초 전/후	datetime '2022-12-25 12:30:20' + 100 = 12:30:20.100 PM 12/25/2022

연산자 - 계속

- 집합연산자

UNION	DIFFERENCE	INTERSECT

- 비트연산자

&	|	^	~	<<	>>

- 문자열 연산자

+, ||	LIKE

  - like : 우절단의 경우 인덱스 검색 가능, %, _ 사용 , 문자열 내의 %, _ 검색위한 escape 문자 지정

- 정규 표현식

REGEXP	RLIKE	

  - 패턴 : "."은 문자 하나와 매칭, [...]은 대괄호 안의 문자 중 하나와 매칭, "*"는 앞의 문자 또는 문자열이 연속으로 나열 된 문자열과 매칭
  - 패턴이 존재하면 1을 반환하고, 그렇지 않은 경우 0을 반환
  - REGEXP BINARY : 대소문자 구분하여 패턴 검색

- 기타

BETWEEN	EXISTS	IN

데이터 입력

- INSERT
  - update 형태의 insert 문 사용 가능
  - select SQL을 사용하여 입력 가능
  - ON DUPLICATE KEY UPDATE : unique, Primary key 에 중복 값 입력시 새로운 값으로 갱신

```
-- nation 테이블에 code 컬럼에 'cd0', name 컬럼에 '신설국' 이라는 값을 입력한다.
INSERT INTO nation SET code = 'cd0', name = '신설국'

-- nation 테이블에 앞서 입력한 값을 검색하여 code 는 'cd1' 을 name 은 검색한 값에 1을 붙여 입력한다.
INSERT INTO nation(code, name) SELECT 'cd1', name || '1' from nation where code = 'cd0'

-- nation 테이블 (code, name) 에  'cd2', '신설국2' 와 'cd3', '신설국3' 총 2건을 입력한다.
INSERT INTO nation(code, name) values ('cd2', '신설국2'), ('cd3', '신설국3')

-- nation 테이블에 code='cd3', name='후설국'을 입력한다. 만약 값이 중복되면 name 의 값을 바꾸고, capital 컬럼에
'설도' 를 넣어준다.
INSERT INTO nation(code, name) VALUES ('cd1', '후설국')
  ON DUPLICATE KEY UPDATE name = '후설국', capital = '설도'
```

- REPLACE
  - Primary key, Unique 조건에 중복 값 입력 시 삭제 후 입력(INSERT문을 사용하는 것이 성능 유리)

```
-- nation 테이블에 code='cd3', name='대설국'을 입력한다. 만약 값이 중복되면 기존값을 바꾼다.
replace nation set code = 'cd3', name = '대설국'
```

데이터 수정

- UPDATE
  - LIMIT : 레코드 개수 제한
  - Set 절에 SELECT 조회한 결과로 갱신
  - ORDER BY 이용 해당 컬럼의 순서로 레코드를 갱신
  - 다른 테이블과 조인을 통해 갱신

```
-- nation 테이블에서 원하는 조건의 레코드 중 code 값으로 정렬하여 처음 1개만 갱신
UPDATE nation SET (continent, capital) = (SELECT 'asia', '수도' FROM dual)
WHERE code like 'cd%'
ORDER BY code limit 1

-- 2000년 올림픽 기록에서 'Peel Ian' 선수의 메달을 금메달로 변경한다.
-- 올림픽 기록은 record 테이블, 선수 정보는 athlete 테이블에 있다.
UPDATE record r, athlete a SET medal = 'G'
WHERE r.host_year = 2000 and r.athlete_code = a.code and a.name = 'Peel Ian'
```

MERGE

- 주어진 조건에 따라 데이터의 입력/수정/삭제 작업을 하나의 질의로 처리

```
-- game 테이블에 record 테이블의 값이 있으면 game 테이블의 값을 수정하고 없으면 입력한다.
MERGE INTO game g USING record r
ON   g.host_year = r.host_year AND g.event_code = r.event_code AND g.athlete_code = r.athlete_code
WHEN MATCHED THEN
        UPDATE SET g.medal = r.medal WHERE r.medal in ('G', 'S', 'B')
        DELETE WHERE r.medal not in ('G', 'S', 'B')
WHEN NOT MATCHED THEN
        INSERT VALUES(r.host_year, r.event_code, r.athlete_code, '000', '000', r.medal, sysdate)
```

데이터 삭제

- DELETE
  - from 절 생략 가능 (CUBRID 9.0 이상이고 대상 테이블이 한 개일 경우)
  - LIMIT : 레코드 개수 제한
  - 다른 테이블과 조인을 통해 삭제

```
-- nation 테이블에서 원하는 조건의 레코드 중 1개만 삭제
DELETE from nation WHERE code like 'cd%' limit 1

-- 2000년 올림픽 기록에서 'Peel Ian' 선수의 모든 기록을 삭제한다.
-- 올림픽 기록은 record 테이블, game 테이블, 선수 정보는 athlete 테이블에 있다.
DELETE g, r FROM game g, record r, athlete a
WHERE a.name = 'Peel Ian'  and a.code = r.athlete_code and r.host_year = 2000 and r.host_year =
g.host_year and r.athlete_code = g.athlete_code
```

- TRUNCATE
  - DELETE 보다 빠르게 모든 데이터 삭제
  - delete trigger 동작하지 않음.
  - auto increment 사용중인 경우 truncate 후엔 초기값부터 생성, delete 의 경우 삭제 전 값 기준으로 증가

```
-- 테이블 user_tbl_bak 의 모든 데이터 삭제
truncate [table] user_tbl_bak
```

ORDER BY

- 질의 결과 정렬
  - 정렬 대상이 select 절에 없어도 된다.
  - NULL 은 가장 작은 값으로 처리되며, NULL 값의 위치를 지정할 수 있다.

```
-- nation 테이블에서 code 명이 cd 로 시작하는 레코드를 capital 순서대로 정렬
select * from nation where code like 'cd%' order by capital

-- nation 테이블에서 code 명이 cd 로 시작하는 레코드를 capital 순서대로 정렬하되 NULL 을 제일 나중에 나오도록 한다. 또한 검색 결과는 code 와 name 만 가져온다.
select code, name from nation where code like 'cd%' order by capital NULLS LAST
```

  - 인덱스와 병행 사용시 성능 개선
    - 인덱스를 이용하여 검색되는 경우, 결과의 순서가 인덱스 순서와 동일하므로 정렬을 하지 않아도 된다.
    - 검색 조건과 정렬 조건을 적절히 활용하여 인덱스를 생성하면, 질의 수행시 정렬을 하지 않는다.

```
-- nation 테이블에서 code 명이 cd 로 시작하는 레코드를 code 순서대로 정렬
select * from nation where code like 'cd%' order by code

Query plan:
 Index scan(nation nation, pk_nation_code, (nation.code>= ?:0  and nation.code< ?:1 ))
 skip ORDER BY
```

GROUP BY

- 검색 결과를 특정 컬럼(들)을 기준으로 그룹화
  - 결과 정렬 방법 제공 : ASC/DESC
  - group by 절에 모든 컬럼 명시하지 않아도 되나, 선택되어지는 값은 임의의 값
  - 중간 집계 방법 제공 : WITH ROLLUP
  - having절 : 그룹 결과 선택을 위한 조건 설정

```
-- record 테이블에서 메달별 획득 개수를 구한다.
SELECT medal, count(*) FROM record GROUP BY medal

-- record 테이블에서 1988년도 결과 종목별, 메달별 로 2개 이상 획득한 종목/메달/개수를 종목 역순으로 구해온다.
SELECT event_code, medal, count(*) FROM record WHERE host_year = 1988
GROUP BY event_code desc, medal HAVING count(*) > 1

-- record 테이블에서 메달별 획득 개수를 구하는데, 메달을 획득한 연도를 같이 구한다.
SELECT medal, host_year, count(*) FROM record GROUP BY medal

-- record 테이블에서 메달별, 연도별 획득 개수를 구한다. 이때 메달별 누계와 전체 개수를 구한다.
SELECT medal, host_year, count(*) FROM record GROUP BY medal, host_year with rollup
```

```
csql> SELECT medal, host_year, count(*) FROM record GROUP BY medal;

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

  medal                 host_year     count(*)
================================================
  'B'                        2000          668
  'G'                        2000          668
  'S'                        2000          663

3 rows selected. (0.005143 sec) Committed.
```

```
csql> SELECT medal, host_year, count(*) FROM record GROUP BY medal, host_year with rollup;

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

  medal                 host_year     count(*)
================================================
  'B'                        1988          120
  'B'                        1992          126
  'B'                        1996          128
  'B'                        2000          147
  'B'                        2004          147
  'B'                        NULL          668
  'G'                        1988          118
  'S'                        2004          147
  'S'                        NULL          663
  NULL                       NULL         1999

19 rows selected. (0.005429 sec) Committed.
```


CUBRID

일련번호

- rownum
  - 질의 결과로 생성될 각 레코드에 대한 순서를 나타내는 번호
  - 질의 결과 일부만 검색/갱신/삭제 가능
  - 1부터가 아닌 임의의 번째 사용 가능. 예) where rownum between 11 and 15
  - order by, group by 전에 생성
    - 정렬후 순서: orderby_num(), group by 후 순서: groupby_num()
- limit
  - 질의 최종 결과에서의 개수 제한
  - limit 10 : 처음부터 10개
  - limit 0, 10: 첫번째(0부터 시작) 부터 10개

```
-- record 테이블에서 2000 년도의 기록을 11번째부터 20번째까지 구하고, 그 번호와 연도, 선수, 메달 종류를 구한다.
SELECT rownum, host_year, athlete_code, medal FROM record WHERE rownum between 11 and 20

-- record 테이블에서 금메달을 딴 선수들을 최근 올림픽부터 상위 10 명으로, 그 순서를 포함하여 구한다.
SELECT orderby_num(), host_year, athlete_code FROM record WHERE medal = 'G'
    ORDER BY host_year desc limit 10

-- record 테이블에서 금메달을 2개 이상 획득한 선수와 개수 상위 10명으로, 그 순서를 포함하여 구한다.
SELECT groupby_num(), athlete_code, count(medal) FROM record WHERE medal = 'G'
    GROUP BY athlete_code HAVING count(medal) > 1 limit 0, 10
```

click counter

- incr() / decr()
  - 컬럼 값을 1만큼 증가/감소 시키는 함수
  - 트랜잭션의 영향을 받지 않음
  - 검색시 특정 컬럼의 값을 증가/감소 시킬 수 있으므로, update 질의 처리를 줄일 수 있다.
    - 게시물 조회건수 증가에 사용 효과적이다.
  - update 가 발생하므로, HA 환경에서 부하 분산시 master 로만 접속하여야 한다.
  - 검색 결과 1건일 경우에만 가능하며, 2건이상일 경우 에러 처리

```
-- 게시판 테이블(board) 의 본문 보기시 조회 건수(view_cnt) 를 자동 증가 시킨다.
SELECT incr(view_cnt), ... FROM board WHERE ...
```

FOR UPDATE

- FOR UPDATE
  - 검색한 레코드에 대하여 update/delete 를 위해 X_LOCK(write lock) 을 선점한다.
  - 트랜잭션 종료시 잠금 해제
  - CUBRID 9.3 이상 지원
- 제약사항
  - 부질의 안에서 FOR UPDATE 절 사용 불가. 단, FOR UPDATE 절이 부질의 참조 가능.
  - GROUP BY, DISTINCT 또는 집계 함수를 가진 질의문에서 사용 불가
  - UNION을 참조 불가능

```
-- nation 테이블의 code 가 'cd01' 인 레코드에 잠금을 선점하고, capital 을 'seoul' 로 수정한다.
SELECT * FROM nation WHERE code = 'cd1' FOR UPDATE
UPDATE nation SET capital = 'seoul' WHERE code = 'cd1'
```

조인

- 검색 대상 테이블이 2개 이상인 경우 관련 테이블들로 부터 검색
- Inner join
  - 테이블간 조인 조건을 만족하는 데이터만 검색

```
-- nation 테이블에 등록된 국가중 올림픽을 개최한 경우
개최한 연도를 국가명과 개최 연도 순으로 검색
SELECT n.name, o.host_year
FROM nation n, olympic o
WHERE n.name = o.host_nation
ORDER BY n.name, o.host_year
```

```
  name                      host_year
=====================================
  'Australia'                    1956
  'Australia'                    2000
  'Belgium'                      1920
  'Canada'                       1976
  'Finland'                      1952
  'France'                       1900
  'France'                       1924
  'Germany'                      1936
  'Germany'                      1972
  'Greece'                       1896
  'Greece'                       2004
  'Italy'                        1960
  'Japan'                        1964
  'Korea'                        1988
  'Mexico'                       1968
  'Netherlands'                  1928
  'Spain'                        1992
  'Sweden'                       1912

18 rows selected. (0.011237 sec)
```

- Outer Join
  - 어느 한 테이블을 기준으로 조인되는 테이블에서 조인 조건을 만족하는 값이 없으면 NULL 로 처리
  - ANSI SQL 권장. ORACLE 형식으로 사용시 조인 조건이 많은 경우 결과가 달라질 수 있음
  - Left Outer Join: 왼쪽 테이블을 기준으로 오른쪽 테이블을 조인
  - Right Outer Join: 오른쪽 테이블을 기준으로 왼쪽 테이블을 조인
  - Full Outer Jon: 양방향으로 outer join. 현재 지원되지 않음.

```
-- nation 테이블에 등록된 국가에 대하여 올림픽을 개최한 연도를 검색 (ANSI SQL)
SELECT n.name, o.host_year
FROM nation n LEFT OUTER JOIN olympic o ON n.name = o.host_nation
ORDER BY n.name, o.host_year
-- 또는 ORACLE 형식으로.
SELECT n.name, o.host_year
FROM nation n, olympic o
WHERE n.name = o.host_nation(+)
ORDER BY n.name, o.host_year
```

```
  name                      host_year
=====================================
  'Afghanistan'                  NULL
  'Albania'                      NULL
  'Algeria'                      NULL
  'American Samoa'               NULL
  'Andorra'                      NULL
  'Angola'                       NULL
  'Antigua & Barbuda'            NULL
  'Arab Republic of Egypt'       NULL
  'Argentina'                    NULL
  'Armenia'                      NULL
  'Aruba'                        NULL
  'Australia'                    1956
  'Australia'                    2000
  'Austria'                      NULL
  'Azerbaijan'                   NULL
  'Costa Rica'                   NULL
223 rows selected. (9.973794 sec)
```

계층형 SQL

- 상하관계가 있는 컬럼 값을 가지는 테이블에 대하여 상하 관계를 기준으로 검색
- 루트 노드: 상하 관계에서 제일 상위에 해당하는 노드(레코드)
- START WITH: 검색시 루트 노드를 찾기 위한 조건을 지정
- CONNECT BY: 하위 관계의 노드를 검색하기 위한 조건을 지정.
  - PRIOR : 하위 노드 검색을 위한 현재 노드의 컬럼을 의미
- 계층 질의 의사 컬럼
  - LEVEL: 검색 결과 노드의 순서. 루트 노드가 LEVEL 1 이며 1씩 증가
  - CONNECT_BY_ISLEAF: 최 하위 노드이면 1을 넘겨준다.
  - CONNECT_BY_ISCYCLE: 현 노드의 상위 노드인데, 현 노드의 하위 노드가 되는 경우 1을 넘겨준다.
  - CONNECT_BY_ROOT : 루트 노드의 컬럼 값을 넘겨준다.
  - SYS_CONNECT_BY_PATH : 루트 노드 부터 현재 노드까지의 컬럼 값을 주어진 구분자를 이용하여 패스처럼 묶어서 넘겨준다.

```
-- 다음의 테이블을 가정한다. 한사람의 관리자는 mgr_id 에 저장되며, 최상위 관리자는 mgr_id 가 NULL 값을 가진다.
  create table employee ( id int, name varchar(10), mgr_id int )
  insert into employee values(1, 'name1', NULL), (2, 'name2', NULL), (3, 'name3', 1), (4, 'name4', 2), (5, 'name5', 3), (6, 'name6', 3)

-- 최상위 관리자로부터 관리 관계를 검색한다. 상하관계에 대한 레벨과 최상위 관리자로부터 현 사람까지의 이름을 / 로 구분하는 패스 형태로 출력한다. 이때 최하위 노드인지도 같이 확인한다.
SELECT id, name, sys_connect_by_path(name, '/'), mgr_id, connect_by_isleaf FROM employee
START WITH mgr_id is null
CONNECT BY prior id = mgr_id
```


CUBRID

실습

1. athlete 테이블에서 이름에 _ 를 포함한 선수를 검색한다.
    1. _ 는 like 의 wild card 이므로, escape 문자를 지정해서, escape 다음의 문자는 일반 문자로 인식하게 한다.

```
csql> select name from athlete where name like '%\_%' escape '\';

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

  name
======================
  'Said_Guerni Aissa Djabir'

1 row selected. (0.015938 sec) Committed.
```

2. record 테이블에서 금메달을 2개 이상 획득한 선수중 메달 개수 상위 10명을 구한다.
    1. 메달수 기준 순번과 선수 이름, 메달 개수를 함께 구한다.

```
csql> SELECT orderby_num(), a.name, count(medal) cnt FROM record r, athlete a
csql> WHERE medal = 'G' AND r.athlete_code = a.code
csql> GROUP BY athlete_code HAVING count(medal) > 1
csql> ORDER BY cnt desc limit 10;

=== <Result of SELECT Command in Line 4> ===

         orderby_num()  name                          cnt
============================================================
                     1  'Thompson Jenny'                6
                     2  'Otto Kristin'                  6
                     3  'Biondi Matthew'                5
                     4  'Egerszegi Krisztina'           5
                     5  'Lewis Carl'                    5
                     6  'Thorpe Ian'                    5
                     7  'Van Dyken Amy'                 5
                     8  'Phelps Michael'                5
                     9  'Darnyi Tamas'                  4
                    10  'De Bruijn Inge'                4

10 rows selected. (0.009474 sec) Committed.
```

실습

3. 계층형 질의 설명에 사용된 테이블을 생성하고, 기초 데이터를 입력한다.

```
create table employee ( id int, name varchar(10), mgr_id int )
insert into employee values(1, 'name1', NULL), (2, 'name2', NULL), (3, 'name3', 1), (4, 'name4', 2), (5, 'name5', 3), (6, 'name6', 3)
```

4. 최상위 관리자로부터 관리 관계를 검색한다. 상하관계에 대한 레벨과 최상위 관리자로부터 현 사람까지의 이름을 / 로 구분하는 패스 형태로 출력한다. 이때 최하위 노드인지도 같이 확인한다.

```
csql> create table employee ( id int, name varchar(10), mgr_id int )
csql> insert into employee values(1, 'name1', NULL), (2, 'name2', NULL), (3, 'name3', 1), (4, 'name4', 2), (5, 'name5',
3), (6, 'name6', 3)
csql> ;x
Execute OK. (0.002748 sec) Committed.
6 rows affected. (0.003172 sec) Committed.

2 command(s) successfully processed.
csql> SELECT id, name, sys_connect_by_path(name, '/'), mgr_id, connect_by_isleaf FROM employee
csql> START WITH mgr_id is null
csql> CONNECT BY prior id = mgr_id;

=== <Result of SELECT Command in Line 3> ===

           id  name                  sys_connect_by_path(name, '/')       mgr_id  connect_by_isleaf
============================================================================================
            1  'name1'               '/name1'                        NULL                    0
            3  'name3'               '/name1/name3'                     1                    0
            5  'name5'               '/name1/name3/name5'               3                    1
            6  'name6'               '/name1/name3/name6'               3                    1
            2  'name2'               '/name2'                        NULL                    0
            4  'name4'               '/name2/name4'                     2                    1

6 rows selected. (0.007798 sec) Committed.
```

11. 백업과 복구

데이터베이스 백업 시 참고사항

- 데이터베이스 백업 시 참고사항
  - 데이터베이스 백업은 데이터베이스의 이미지를 파일 등으로 덤프하는 것이다.
  - 백업이 수행되지 않았거나 백업 화일이 없는 경우 복구할 수 없으므로 반드시 백업을 하여야 한다.
  - OS 레벨의 copy나 ftp 등을 이용한 데이터베이스 볼륨(파일) 백업은 권장하지 않는다.
    - 파일 복사 시점에 변경되는 부분이 깨질 수 있다.
    - 특히 시점 복구가 불가능하다.
    - CUBRID export 도구인 unloaddb 역시 백업으로 권장하지 않는다. 단순 export 로 시점 복구가 불가능하기 때문이다.
  - 백업은 매일 받는 것을 권장하며, 사용량이 적은 새벽 시간 등에 자동 백업을 권장한다.
  - 백업 볼륨은 서버에 1일치만 보관하고, 외부 저장장치에 별도 보관하는 것을 권장한다.
  - 데이터베이스를 새로운 버전으로 업그레이드한 경우 새로 생성한 데이터베이스를 즉시 백업해야 한다.
    - 데이터베이스 백업은 버전이 다르면 복구가 불가능하다.
  - 데이터베이스 백업 시점에 불필요한 보관 로그들을 정리(-r option) 할 수 있다.
    - 보관 로그들의 내용은 대부분 데이터 볼륨에 반영되어 보관할 필요가 없다.
    - 단, HA 환경에서는 master/slave 간 데이터 동기에 사용되므로 이 옵션을 사용하면 않된다. 데이터 동기가 완료되면 설정에 따라 적절히 정리된다.
  - 백업 볼륨을 이동하거나 이름을 변경하지 않으면 이후의 백업 시에 이전 볼륨을 덮어쓴다.
    - 백업시 이전 백업이 있는 경우 덮어쓸지를 물어온다.
  - 백업 시점 이후로의 시점 복구는 데이터베이스 로그를 이용하여 이루어진다.

데이터베이스 백업 시 참고사항 - 계속

- 데이터베이스 백업 정책
  - 백업할 데이터의 선택
    - 데이터베이스의 전체 또는 변경된 부분에 대한 증분 백업을 할 것인가?
      - 데이터베이스 사이즈가 매우 큰 경우 전체 백업과 증분 백업을 같이 수행하면 백업 시간을 줄일 수 있다.
      - 단, 백업시 압축을 수행할 수 있기에 전체 백업과 증분 백업의 크기나 시간의 차이가 별로 없다면 전체 백업을 수행하는 것이 복구시 유리하다.
    - 데이터 보존 기간은 얼마로 할 것인가?
    - 데이터베이스와 함께 백업되어야 할 다른 파일은 있는가?
      - lob 사용시 lob 파일의 백업 (데이터베이스 디렉토리/lob (기본설정의 경우))
      - JAVA SP 에 사용된 class 또는 jar 파일의 백업
  - 백업 방법
    - On-line/Off-line 백업
    - 전체(level 0)/ 증분(level 1 또는 2) 백업
      - 증분은 현재 level 이전의 level 백업 이후 변경된 내용을 백업한다. 이전 백업 내용 이후가 아님.
      - 증분 백업이 이전 level 의 백업이 있어야한 가능하며, 없는 경우 에러가 발생한다.
      - 예) 일요일: 전체(0 level) 백업, 월요일: 증분(1 level) 백업, 화요일: 증분(1 level) 백업
        - » 화요일 백업은 전날 월요일 이후 변경된 부분이 아닌, 이전 level 인 일요일 백업 이후 변경된 내용이 백업된다.
  - 백업 시기
    - 데이터베이스의 활동이 가장 적은 시기
    - 배치 작업이 있는 경우 배치작업과 시간을 다르게 설정하는 것이 좋다

백업

- 데이터베이스 백업
  - Disk와 tape 등의 device 지정이 가능하다.
  - 백업이 완료되면 백업 볼륨 및 백업정보 파일을 생성한다. (예, demodb_bk0v000, demodb_bkvinf)
  - 백업시 데이터베이스의 일관성 검사를 진행한다. 이때 다음의 상황이 발생할 수 있으므로 일관성 검사는 필요에 따라 선택하여야 한다.
    - 일관성 검사를 위해 테이블에 락이 발생할 수 있고 이로 인해 서비스 지연이나 locktimeout 이 발생할 수 있음.
    - backupdb 에서 lock 획득을 실패한 경우 백업 수행이 중단될 수도 있음
  - HA 환경일 경우 -r 옵션을 사용하면 master/slave 간 동기화되지 않은 정보가 지워질 수 있어 사용하지 않아야 한다.

- 사용 예

```
# 서비스 중에(-C) 압축을 수행하여(-z) 전체 백업을 /backup/db 디렉토리에(-D) 수행한다.
# 전체 백업인 0 level 은 기본 값이므로 구지 옵션을 주지 않아도 된다.
$ cubrid backupdb -C -z -D /backup/db demodb

# 서비스가 중단된 상태에서(-S) 압축을 수행하면서 불필요한 보관 로그를 삭제하고(-r) 일관성 검사없이 (--no-check)
   1 level 의 증분 백업을(-l 1) /backup/db 디렉토리에 수행한다.
# HA 환경이 아닐 경우 -r 옵션을 사용할 수 있다.
$ cubrid backupdb -S -z -r --no-check -D /backup/db -l 1 demodb

### HA 환경일 경우 데이터베이스 이름 뒤에 @localhost 를 붙여준다.
$ cubrid backupdb  -C -z -D /backup/db demodb@localhost
```

복구

- 데이터베이스 복구
  - 서비스를 중단한 상태에서만 복구가 가능하다.
  - 전체 복구만 가능하다.
  - 백업 시점 또는 백업 이후 원하는 시점을 복구 가능하다.
  - 최근 백업의 level 을 기준으로 복구가 된다.
    - 최근 1 level 백업을 한 경우, 1 level 백업을 이용하나 증분이므로 전체 백업인 0 level 백업도 필요하다.
    - 1 level 백업을 받고 다시 0 level 백업을 받은 경우, 최근 백업이 0 level 이므로 0 level 백업을 이용한 복구만 가능하다.
  - restoredb 명령어 수행 시 –d 옵션을 사용하지 않으면 복구 가능한 가장 최근 시점으로 복구한다.
    - 사용자 실수로 복구가 필요한 경우 반드시 –d 로 해당 시점을 명시하여야 한다.

- 사용 예

```
# demodb 에 대하여 실수로 삭제된 데이터를 복구하기 위해 최근 삭제 시점 이전인 2020년5월19일 17시39분0초로 (-d 19-05-2020:17:39:00) 복구한다.
$ cubrid restoredb -d 19-05-2020:17:39:00 demodb

# 백업된 시점으로 복구한다. 최근 백업이 1레벨 증분 백업으로 1레벨 백업을 이용하여(-l 1) 복구한다.
$ cubrid restoredb -d backuptime -l 1 demodb

# 백업 파일의 백업 시점 등 백업 정보를 확인(--list)한다.
$ cubrid restoredb --list demodb


# 복구시 보관로그가 없어 복구 에러 발생시 보관 로그없이(-p) 복구한다. on-line 백업시 진행중이던 트랜잭션이 백업에 완전히 보관되지 않은 경우 해당 트랜잭션에 대한 정보는 보관로그에 기록되기 때문이다.
$ cubrid restoredb -p demodb
```

실습

1. demodb 를 on-line 전체 백업을 압축으로 하고, 백업 화일의 위치를 확인한다.

```
[cubrid@db1 ~]$ cubrid backupdb -z demodb
Backup Volume Label: Level: 0, Unit: 0, Database demodb, Backup Time: Mon May 23 12:39:24 2022
[cubrid@db1 ~]$ more $CUBRID_DATABASES/databases.txt
#db-name        vol-path                db-host         log-path                lob-base-path
testdb          /data/DB        localhost       /data/DB        file:/data/DB/lob
mydb            /home/cubrid    localhost       /home/cubrid    file:/home/cubrid/lob
demodb          /home/cubrid/CUBRID/databases/demodb    localhost       /home/cubrid/CUBRID/databases/demodb    file:/ho
me/cubrid/CUBRID/databases/demodb/lob
[cubrid@db1 ~]$ more /home/cubrid/CUBRID/databases/demodb/demodb_bkvinf
  0 0 /home/cubrid/CUBRID/databases/demodb/demodb_bk0v000
```

2. 현재 시간을 확인하고, 앞서 입력한 employee 테이블의 데이터를 모두 삭제한다.

```
csql> select sysdatetime;

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

   SYS_DATETIME
==============================
  12:45:51.156 PM 05/23/2022

1 row selected. (0.004936 sec) Committed.

1 command(s) successfully processed.
csql> delete from employee;
6 rows affected. (0.005555 sec) Committed.
```

실습 - 계속

3. 백업을 이용하여 복구한 후, employee 테이블을 검색한다.

```
[cubrid@db1 ~]$ cubrid server stop demodb
@ cubrid server stop: demodb

Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
[cubrid@db1 ~]$ cubrid restoredb demodb

CUBRID 11.0

[cubrid@db1 ~]$ csql -u dba -S -c "select * from employee;" demodb

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

There are no results.
0 row selected. (0.009971 sec) Committed.
```

4. 만약 employee 테이블에 데이터가 없다면, 복구 시점을 삭제 전 시간으로 하여 복구하고, employee 테이블을 검색해본다.

```
[cubrid@db1 ~]$ cubrid restoredb -d '23-05-2022:12:45:51'  demodb

CUBRID 11.0

[cubrid@db1 ~]$ csql -u dba -S -c "select * from employee;" demodb

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

           id  name                       mgr_id
============================================
            1  'name1'                      NULL
            2  'name2'                      NULL
            3  'name3'                         1
            4  'name4'                         2
            5  'name5'                         3
            6  'name6'                         3

6 rows selected. (0.003273 sec) Committed.
```

12. 데이터베이스 재구성

데이터베이스 재구성

- 데이터베이스 재구성 이유
  - 운영 중 데이터베이스 장애 복구 또는 최적화를 위해 재구성을 진행할 경우가 있다.
  - CUBRID 버전 업그레이드시 볼륨 호환이 되지 않는 경우 재구성을 진행해야 한다.

- 데이터베이스 재구성 방법
  - 백업(backup) : 작업 오류를 대비해 backupdb/copydb/renamed 를 사용한 백업
  - 언로드(export) : 원본 데이터베이스를 파일로 내려 받기
  - Rebuilding : 기존의 데이터베이스 삭제 및 새로운 데이터베이스 생성
    - renamedb 를 한 경우 이름을 변경한 것이므로 삭제할 필요 없다.
  - 로드(import) : 언로드 받은 파일을 새로운 데이터베이스에 적재
  - 통계정보 갱신(optimizedb) : 질의 수행 계획 생성시 필요한 통계 정보를 만들어 줌
  - 백업본 정리: 재구성이 완료되면 백업했던 부분을 삭제한다.
    - 필요에 따라 일정기간 데이터 이상여부를 확인 후 삭제할 수 있다.

- 데이터베이스 재구성을 위한 유틸리티 : unloaddb(export), loaddb(import), renamedb

데이터베이스 복사

- 현재의 데이터베이스를 다른 이름으로 복사
  - 데이터베이스 재구성 시 백업 방법 중 하나로 활용 가능
  - 데이터베이스를 다른 경로에 복사 가능
  - 데이터베이스 종료후 사용 가능
  - 로그 볼륨 및 확장 볼륨의 경로 지정 가능
  - 사용자 계정과 암호는 동일하게 복사

- 사용 예

```
# demodb 를 demodb_bk 라는 이름으로 /data/DB 디렉토리(-F) 로 복사하면서 데이터베이스 로그는 /data/DB/log 디렉토리로(-L) 복사한다.
$ cubrid copydb -F /data/DB -L /data/DB/log demodb demodb_bk

# demodb_bk 를 demodb_bak 라는 이름으로 /data/DB 디렉토리로 복사하면서 동일한 이름의 데이터베이스가 존재하면 삭제(-r) 하고 복사후 원본은 제거(-d) 한다.
$ cubrid copydb -F /data/DB -r -d demodb_bk demodb_bak
```

데이터베이스 이름 변경

- 데이터베이스 이름 변경(renamedb)
  - 데이터베이스 재구성시 백업 방법의 하나로 활용 가능.
    - 문제 발생시 기존 데이터베이스로 이름 변경만으로 빨리 복구 할 수 있다.
    - 백업/복구 보다 빠르므로, 데이터베이스 변경 작업시 백업 용도로 활용 가능하다.
  - 단, 이름만 변경하는 것이므로 데이터베이스가 위치한 디렉토리의 여유 공간이 충분해야 한다.
  - 데이터베이스 서버 종료후 사용 가능.
  - 사용 예

```
# 데이터베이스 demodb_bak 의 이름을 demodb_bk 로 변경
$ cubrid renamedb demodb_bak demodb_bk
```

데이터베이스 언로드

- 데이터베이스 언로드(export)
  - 현재 데이터베이스를 파일로 내려 받는다.(계정 암호 반드시 필요. 계정 미지정시 DBA)
  - 데이터베이스 재구성 또는 특정 테이블만 다른 DB로 옮기거나 전체 스키마 확인 등을 위해 사용한다.
  - 생성 파일 (현재 디렉토리에 생성)
    - 스키마 파일 : 데이터베이스의 스키마 정의 포함하는 파일(<DB명>_schema)
    - 객체 파일 : 데이터베이스의 데이터를 포함한 파일 (<DB명>_objects)
    - 인덱스 파일 : 데이터베이스에 정의된 인덱스를 포함하는 파일 (<DB명>_indexes)
    - 트리거 파일 : 데이터베이스에 정의된 트리거를 포함하는 파일 (<DB명>_triger)
  - 사용 예 (데이터베이스 이름: demodb_bk)

```
# dba 계정(-u)으로 'dba123' 암호(-p)를 이용하여 off-line(-S) 으로 진행상태 확인(-v)하면서 export
$ cubrid unloaddb -S -u dba -p 'dba123' -v demodb_bk

# on-line(-C) 으로 명시된 테이블에 대하여(-i, --input-class-only) 데이터만(-d) export
# table.list: 테이블 이름을 라인단위로 입력한 화일
$ cubrid unloaddb -C -d -i table.list --input-class-only demodb_bk

# on-line 으로 dbuser 계정의 테이블 목록(-s)만 /data/tmp 디렉토리에(-O) 생성
# 계정 암호를 입력하지 않아 암호를 물어온다
$ cubrid unloaddb -C -s -u dbuser -O /data/tmp demodb_bk
Password:

# HA 환경으로 서비스 구동중에 dba 계정으로 'dba123' 암호를 이용하여 진행상태 확인(-v)하면서 export
$ cubrid unloaddb -C -u dba -p 'dba123' -v demodb_bk@localhost
```

데이터베이스 로드

- 데이터베이스 로드(import)
  - unloaddb 를 이용하여 export 받은 파일들을 데이터베이스에 import한다.
    - unloaddb 형식으로 작성된 파일도 사용 가능하며 해당 테이블에 대한 권한을 가진 계정으로 수행한다.
    - --data-file-check-only 옵션을 이용하거나, -I 옵션 없이 사용하여 파일의 오류가 있는지 확인 필요
  - 특정 테이블만 export 받은 경우가 아니라면, 시스템 테이블 정보가 포함되어 있으므로 반드시 off-line 상태로 DBA 계정으로 import 하여야 한다
    - 사용자 정보도 export 되어 있어, 사용자의 암호가 변경되거나 사용자가 추가되거나 할 수 있다.
    - 특정 테이블만 export 받은 경우(--input-class-only 사용)는 해당 테이블을 사용할 계정에서 import 한다.
  - export 받은 데이터가 import 될 충분한 공간이 있어야 한다. 부족할 경우 볼륨을 추가하여야 한다.
  - 이미 데이터가 존재할 경우 기본키/unique/외래키 참조 등의 오류가 발생할 수 있으므로 데이터가 적합한지 확인할 필요가 있다.

데이터베이스 로드 - 계속

– 사용예 (데이터베이스 이름: testdb)

```
# dba 계정(-u)으로 스키마 정보(-s <스키마 화일명))를  상태 확인(-v)하면서 import
$ cubrid loaddb -u dba -s demodb_bk_schema -v testdb

# dba/dba123 계정으로 데이터 포맷 검사 없이(-I) 데이터 정보만(-d <데이터 화일명>)를 1만건 단위 커밋(-c)수행하면서 통계정보를 갱신(--no-statistics)하지않고 oid 참조(--no-oid) 없이 상태 확인하면서 import
  -c: 데이터 impor시 적정 건수 단위로 커밋을 해주는 것이 import 속도에 유리하다. 1만-10만 정도를 시스템 성능에 따라 조정한다.
  --no_statistics: 질의 플랜 수립에 필요한 통계 정보를 만들지 않는다. 인덱스 추가 시 하거나 명령(optimizedb) 을 이용하여 별도로 하는 것이 좋다
  --no-oid: OBJECT 개념 (object data type, foreign key "on cache object" option) 을 사용하지 않는 경우
$ cubrid loaddb -u dba -p 'dba123' -c 10000 --no-statistics --no-oid -I -v -d demodb_bk_objects testdb

# dba 계정으로 인덱스 정보만(-i <인덱스 화일명>) 상태 확인하면서 import. 암호가 있는 경우 암호를 입력하지 않으면 암호를 물어온다
$ cubrid loaddb -u dba -i demodb_bk_indexes testdb
Password:
```

데이터베이스 재구성 절차

- 재구성 절차
  1. 서비스 종료

```
$ cubrid service stop
```

  2. 데이터베이스 백업
     - 데이터베이스 백업, 데이터베이스 복사 중 데이터베이스 이름 변경이 가장 빠르다.

```
# demodb 데이터베이스의 이름을 demodb_bak 로 변경한다.
$ cubrid renamedb demodb demodb_bak
```

  3. 스키마/데이터 내려 받기
     - 스키마 파일 : <데이터베이스 이름>_schema
     - 데이터 파일 : <데이터베이스 이름>_objects
     - 인덱스 파일 : <데이터베이스 이름>_indexes
     - 트리거 파일 : <데이터베이스 이름>_trigger

```
# demodb_bak 의 내용을 /data/unload 아래(-O /data/unload) export 한다.
$ cd /data; mkdir unload; cd unload
$ cubrid unloaddb -S -u dba demodb_bak
# dba 계정에 암호가 있는 경우 암호를 물어오니 입력하면 된다.
```

데이터베이스 재구성 절차 - 계속

- 재구성 절차
  4. 데이터베이스 생성
     - 데이터베이스 용량을 재 산정해 데이터베이스 생성한다.

```
# new_db 를 한글 utf8 문자셋으로 /data/new_db 아래 10G의 크기로 생성한다.
$ cd /data; mkdir new_db; cd new_db
$ cubrid createdb --db-volume-size=10G new_db ko_KR.utf8
# permanent temp 볼륨을 1G 크기로 추가한다.
$ cubrid addvoldb -S -p temp --db-volume-size=1G new_db
```

  5. 스키마 / 데이터 올리기
     - 앞서 내려 받은 스키마/데이터 등을 새로 생성한 데이터베이스에 올린다.
     - 재구성이므로 dba 계정으로 import 하여야 한다.

```
# 스키마 정보를 먼저 생성한다.
$ cubrid loaddb -v -u dba -l -s /data/unload/demodb_bak_schema new_db
# --no-oid : 객체개념을 사용하지 않았거나, foreign key option 에 on cache object 미사용 시에만 사용
# 재구성전 데이터베이스 dba 계정에 암호가 있는 경우, import 시 물어오니 dba 암호를 입력하면 된다.
$ cubrid loaddb -v -u dba --no-statistics --no-oid -l -c 10000 -d /data/unload/demodb_bak_objects
new_db
# 인덱스가 있는 경우 인덱스를 등록한다.
$ cubrid loaddb -v -u dba --no-statistics -l -i /data/unload/demodb_bak_indexes new_db
# 트리거가 있는 경우 트리거를 등록한다.
$ cubrid loaddb –v -u dba -l -s /data/unload/demodb_bak_trigger new_db
# 통계 정보를 갱신한다.
$ cubrid optimizedb new_db
```

  6. CUBRID 서비스 구동후 데이터 이상여부를 확인한다.

CUBRID

데이터베이스 재구성 절차 - 계속

- 재구성 절차
  4. 데이터베이스 생성
     - 데이터베이스 용량을 재 산정해 데이터베이스 생성한다.

```
# new_db 를 한글 utf8 문자셋으로 /data/new_db 아래 10G의 크기로 생성한다.
$ cd /data; mkdir new_db; cd new_db
$ cubrid createdb --db-volume-size=10G new_db ko_KR.utf8
# permanent temp 볼륨을 1G 크기로 추가한다.
$ cubrid addvoldb -S -p temp --db-volume-size=1G new_db
```

  5. 스키마 / 데이터 올리기
     - 앞서 내려 받은 스키마/데이터 등을 새로 생성한 데이터베이스에 올린다.
     - 재구성이므로 dba 계정으로 import 하여야 한다.

```
# 스키마 정보를 먼저 생성한다.
$ cubrid loaddb -v -u dba -l -s /data/unload/demodb_bak_schema new_db
# --no-oid : 객체개념을 사용하지 않았거나, foreign key option 에 on cache object 미사용 시에만 사용
# 재구성전 데이터베이스 dba 계정에 암호가 있는 경우, import 시 물어오니 dba 암호를 입력하면 된다.
$ cubrid loaddb -v -u dba --no-statistics --no-oid -l -c 10000 -d /data/unload/demodb_bak_objects
new_db
# 인덱스가 있는 경우 인덱스를 등록한다.
$ cubrid loaddb -v -u dba --no-statistics -l -i /data/unload/demodb_bak_indexes new_db
# 트리거가 있는 경우 트리거를 등록한다.
$ cubrid loaddb –v -u dba -l -s /data/unload/demodb_bak_trigger new_db
# 통계 정보를 갱신한다.
$ cubrid optimizedb new_db
```

데이터베이스 재구성 절차 - 계속

- 재구성 절차
  6. 데이터 정합성 확인
     - 데이터 로드후 화면상의 메시지와 언로드 로그를 이용하여 간단히 확인할 수 있다.
       - 로드시 insert 된 개수와 언로드시 dump 된 개수를 비교하면 된다.
       - 로드 로그로도 확인 가능하나, 인덱스 등록도 loaddb 를 이용하여 기존 로그를 덮어쓴다.

```
Total 19209 object(s) inserted, 0 object(s) failed.
...
$ cd /data/unload; more demodb_bak_unload.log
19209 objects dumped.
```

  7. CUBRID 서비스 구동후 주요 업무를 통해 데이터 이상여부를 추가로 확인할 수 있다.

데이터베이스 삭제

- 서비스를 중단한 상태에서만 삭제 가능
  - 백업이 있는 경우 백업은 같이 삭제되지 않는다.
  - HA 환경일 경우 복제 로그(기본위치: $CUBRID_DATABASES/<db이름>_<대상 서버이름>)는 삭제되지 않는다.
- 사용 예

```
# new_db 를 삭제한다.
$ cubrid deletedb new_db

# demodb_bk 를 삭제한다. 만약 백업이 있는 경우 백업까지 삭제한다.
$ cubrid deletedb -d demodb_bk
```

13. 모니터링

CUBRID 서비스 상태 확인

- CUBRID 전체 서비스 상태 확인
  - 빨간색은 HA 환경에서만 추가로 보여진다.

```
$ cubrid service status
@ cubrid master status
++ cubrid master is running.
@ cubrid server status
 HA-Server demodb (rel 10.2, pid 6301)
@ cubrid broker status
  NAME              PID  PORT   AS   JQ      TPS      QPS    .....
==========================================================
* query_editor     5344 30000    5    0        0        0    .....
* broker1          5355 33000    5    0        0        0    .....
@ cubrid manager server status
++ cubrid manager server is running.
@ cubrid heartbeat status

 HA-Node Info (current db1, state master)
   Node db2 (priority 2, state slave)
   Node db1 (priority 1, state master)

 HA-Process Info (master 5304, state master)
   Applylogdb demodb@localhost:/CUBRID/DATABASES/demodb_db2 (pid 6320, state registered)
   Copylogdb demodb@db2:/CUBRID/DATABASES/demodb_db2 (pid 7842, state registered)
   Server demodb (pid 6301, state registered_and_active)

 HA-Ping Host Info (PING check enabled)
   192.168.0.1          SUCCESS
```

구동된 데이터베이스 서버 정보(버전 등) 목록

구동된 브로커의 기본 상태

매니저 서버 구동 여부
CUBRID Admin 이 사용한다

HA 상태를 보여준다.
세부 내용은 15장 HA 에서 설명

CUBRID

브로커 상태 확인

- SQL 처리 상태 확인
  - 응용에서 요청한 SQL은 모두 broker 를 통해 수행되므로, broker 의 상태를 확인함으로써 SQL 처리 상태를 확인할 수 있다.
- 기본 정보

```
$ cubrid broker status  broker1
@ cubrid broker status
% broker1
--------------------------------------------
ID   PID   QPS   LQS PSIZE    STATUS
--------------------------------------------
 1  2685     9     0    52388 IDLE
 2  2686    12     0    56124 BUSY
 3  2687     0     0    48780 CLIENT_WAIT
 4  2688     0     0    48780 CLOSE_WAIT
 5  2689     0     0    48780 IDLE
```

  - ID : 브로커에서 관리하는 CAS의 일련 번호
  - PID : 브로커 내 CAS process id
  - QPS : 초당 처리된 SQL의 수
  - LQS : 초당 처리되는 장기 실행 질의의 수
  - PSIZE : CAS 프로세스 크기
  - STATUS : 응용 서버의 현재 상태
    - IDLE: 연결되어 있지 않으며, 아무 작업도 수행 중이지 않은 상태
    - BUSY: SQL 수행 중이거나, 응용에서 요청한 작업을 처리중인 상태
    - CLIENT_WAIT: 요청한 작업은 완료되었으나 트랜잭션이 끝나지 않은 상태
    - CLOSE_WAIT: 트랜잭션이 완료되었으며 연결은 유지되어 있는 상태
      - Connection Pool 사용 시 CLOSE_WAIT 상태로 유지 됨

브로커 상태 확인 - 계속

- 브로커 별 상태 확인

```
# broker1 의 상태를 1초 단위로(-s 1) 지속적으로 보여준다.
$ cubrid broker status -b -s 1 broker1

NAME      PID    PORT  AS  JQ  TPS  QPS  SELECT  INSERT  UPDATE  DELETE  OTHERS  LONG-T  LONG-Q
ERR-Q  UNIQUE-ERR-Q  #CONNECT
================================================================================
==
broker1 3269 33000    5    0   70   60      10      20      10      10      10      0/60.0  0/60.0
   30           10          213
```

  - NAME : 브로커 이름
  - PID : 브로커의 프로세스 ID
  - PORT : 브로커의 포트 번호
  - AS : 응용 서버 개수
  - JQ : 작업 큐에서 대기 중인 작업 개수
  - TPS : 초당 처리된 트랜잭션의 수(옵션이 " -b -s <sec> " 일 때만 계산됨)
  - QPS : 초당 처리된 질의의 수(옵션이 " -b -s <sec> " 일 때만 계산됨)
  - SELECT/INSERT/UPDATE/DLETE : 브로커 시작 후 초당 처리되는 개수로 -b -s 옵션 지정한 초 동안 갱신
  - LONG-T : LONG_TRANSACTION_TIME 시간을 초과한 트랜잭션 수 / 파라미터 설정 값
  - LONG-Q : LONG_QUERY_TIME 시간을 초과한 SQL 수 / LONG_QUERY_TIME 파라미터의 값
  - ERR-Q : 에러가 발생한 SQL 수
  - UNIQUE-ERR-Q: Unique 에러가 발생한 SQL 개수로 -b -s 옵션 지정한 초 동안 갱신
  - #CONNECT: 브로커 시작 후 응용 클라이언트가 CAS에 접속한 회수

브로커 상태 확인 - 계속

- broker cas들의 자세한 정보 출력

```
# 자세한 정보를 위해 -f 옵션 추가. 2초 동안 client wait 또는 busy 상태인 cas 의 개수를 2초 간격으로 확인
$ cubrid broker status -b -f -s 2 -I 2

NAME            PID      PSIZE     PORT      AS(T   W   B   2s-W   2s-B) JQ TPS QPS          .....                  CANCELED
....
================================================================================================
==
query_editor 6784  56700  30000          5   1   2      1      2 ..........................................    7
```

  - 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
    - 오랜 시간동안 client waiting 상태로 있는 것은 SQL 수행이 끝난후 응용에서 다른 작업을 오랫동안 하고 있는 것으로 질의로 인한 lock 이 길어지거나 cas 1개가 특정 응용에 종속되어 전체 서비스에 영향을 줄 수 있음.
  - AS(Ns-B): N초 동안 클라이언트 수행(Busy) 상태였던 CAS의 개수
    - 오랜 시간동안 SQL이 수행중일 경우 해당 응용의 서비스 지연에 영향을 줄 수 있음.
  - CANCELED: 브로커 시작 이후 사용자 인터럽트로 인해 취소된 SQL 개수 (-I N 옵션과 함께 사용하면 N초 동안 누적된 개수)

브로커 상태 확인 - 계속

- 브로커 cas 별 상태 확인
  - 상태가 BUSY 일 경우 응용에서 요청한 SQL 수행이 시작된 시간(LAST ACCESS TIME)을 알 수 있다.
    - 이 시간동안 SQL이 수행중이므로 SQL 에 대한 개선이 필요할 수 있다.

```
$ cubrid broker status -f broker1
% 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    5168      BUSY      2015/03/11 16:23:42   demodb     localhost     2015/03/11 16:23:40   192.168.0.10
   10.2.13.0003            NONE           2015/03/11 16:23:42          0            0

2 26947     0     0   51172       IDLE      2015/03/11 16:23:34         -             -                  -               0.0.0.0
      -                       -                              -                   0            0
:
```

- 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. 접속이 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
- TRANSACTION STIME: 트랜잭션 시작 시간
- #CONNECT: 브로커 시작 후 응용 클라이언트가 CAS에 접속한 회수
- #RESTART: 브로커 시작 후 CAS의 재구동 회수

브로커 상태 확인 - 계속

- 브로커 별 cas 상태 확인

```
# broker1 의 상태를 1초 단위로(-s 1) 지속적으로 보여준다.
$ cubrid broker status -f -s 1 broker1
% broker1
--------------------------------------------------------------------------------------------------
ID   PID   QPS    LQS    PORT PSIZE  STATUS        CPU    LAST ACCESS TIME               DB
  HOST   LAST CONNECT TIME       CLIENT IP    SQL_LOG_MODE   TRANSACTION STIME # CONNECT # RESTART
--------------------------------------------------------------------------------------------------
1  317 2     3     0  33001  9596 CLIENT WAIT   0.00 2019/08/15 10:43:38           testdb
localhost 2019/08/15 10:43:02    127.0.0.1         -       2019/08/15 10:43:38         3         0
SQL:
 2  3180     2     0  33002  9704 BUSY          0.00 2019/08/15 10:44:44           testdb
localhost 2019/08/15 10:43:03    127.0.0.1         -       2019/08/15 10:44:44         2         0
SQL: execute update a set i = 1;
 3  3188     2     0  33003  9524 CLOSE WAIT    0.00 2019/08/15 10:42:42           testdb
localhost 2019/08/15 10:41:52    127.0.0.1         -                          -         1         0
SQL:
 4  3196     0     0  33004  1896 IDLE          0.00 2019/08/11 05:42:57            -
    -                   -          0.0.0.0         -                          -         0         0
 5  3204     0     0  33005  2028 IDLE          0.00 2019/08/11 05:42:57            -
    -                   -          0.0.0.0         -                          -         0         0
```

해당 cas의 sql log($CUBRID/log/broker/sql_log/**broker1_2**.sql.log)를 통해 수행중인 SQL 확인
broke이름: broker1, cas ID: 2

데이터베이스 트랜잭션 확인

- 데이터베이스에 접속중인 클라이언트의 트랜잭션 상태를 확인한다.
  - DBA 또는 DBA그룹 사용자만 가능하다.
  - 데이터베이스 클라이언트로 cas 또는 csql 등의 트랜잭션 상태 정보를 확인한다.

- 사용 예

```
# demodb 에 현재 접속중인 클라이언트의 트랜적션 상태를 확인한다. HA 환경일 경우 demodb@localhost
$ cubrid tranlist demodb

Tran_index  User_name  Host_name  Process_id  Program_name  Query_time  Tran_time  Wait_for_lock_holder
SQL_ID  SQLText
------------------------------------------------------------------------------------------------------------------------
1(ACTIVE)       public       test-server       1681       broker1_cub_cas_1  0.00        0.00                -1
*** empty ***
:
4(ACTIVE)       public       test-server       1684       broker1_cub_cas_4  1.80        1.80                -1
*** empty ***
------------------------------------------------------------------------------------------------------------------------
```

  - Tran index: 트랜잭션 인덱스
  - User name: 데이터베이스 사용자 이름
  - Host name: 접속한 클라이언트의 호스트 이름
  - Process id: 클라이언트 프로세스 ID
  - Program name: 클라이언트 프로그램 이름 (예, broker1_cub_cas_1: broker1 이 관리하는 cas 중 1번째 cas)
  - Query time: 수행중인 SQL의 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 트랜잭션 인덱스
  - SQL_ID: 수행중인 SQL ID. cubrid killtran 명령의 --kill-sql-id 옵션에서 사용될 수 있다.
  - SQL Text: 수행중인 SQL(최대 30자).

데이터베이스 트랜잭션 확인 - 계속

- 트랜잭션 확인

```
# demodb 에 접속중인 클라이언트들에 대하여 트랜잭션이 긴 순서대로(즉, 출력 항목 중 7번째(Tran_time)에 대하여(--sort-key=7) 역순으로(--reverse)) 출력
$ cubrid tranlist --sort-key=7 --reverse demodb

Tran_index  User_name  Host_name  Process_id  Program_name  Query_time  Tran_time  Wait_for_lock_holder  SQL_ID  SQLText
---------------------------------------------------------------------------------------------------------------------------
1(ACTIVE)  public  test-server  1681  broker1_cub_cas_1  0.00  169.97  -1
*** empty ***
 2(ACTIVE)  public  test-server  1681  broker1_cub_cas_2  9.80  19.80  1
c71aa435391bc  UPDATE nation SET NAME = 'Repub
---------------------------------------------------------------------------------------------------------------------------

# 위 정보를 2초 간격으로 갱신
$ watch -n 2 cubrid tranlist --sort-key=7 --reverse demodb
```

  - 2번 트랜잭션의 Wait_for_lock_holder 가 -1 이 아닌 1로 보여진다. 이는 2번 트랜잭션이 현재 1번 트랜잭션이 lock 을 잡고 있어 그 lock 이 풀리길 기다리고 있다는 의미이며, 현재 실행중인 질의문의 일부(UPDATE nation ...) 를 보여준다.
  - 1번 트랜잭션에서 Tran_time 이 0 이 아니므로 현재 트랜잭션이 진행중(169.97초 동안)인 상태이다.
  - 또한 질의문이 보이질 않는 이유는 질의 수행을 완료하였으나 commit/rollback 이 수행되지 않은 상태이다. 즉, 현재 수행중인 질의는 없기에 보이지 않을 뿐이다.
  - 수행중인 질의의 전체 모습은 현재 cas 를 통해서 수행 중에 있으므로, cas 의 sql_log ($CUBRID/log/broker/sql_log/**broker1_2**.sql.log) 를 통해 확인할 수 있다.

데이터베이스 트랜잭션 확인 - 계속

- 트랜잭션 상태 정보

```
$ cubrid tranlist demodb

Tran_index  User_name  Host_name  Process_id  Program_name  Query_time  Tran_time  Wait_for_lock_holder  SQL_ID
SQLText
------------------------------------------------------------------------------------------------------------------------
1(ACTIVE)           public       test-server       1681       broker1_cub_cas_1     0.00          0.00                    -1
*** empty ***
2(COMMITTING) public       test-server       1684       broker1_cub_cas_4     1.80          1.80                    -1
*** empty ***
3(ABORTED)          public       test-server       1684       broker1_cub_cas_4     1.80          1.80                    -1
*** empty ***
------------------------------------------------------------------------------------------------------------------------
```

  - ACTIVE : 활성
  - RECOVERY : 복구중인 트랜잭션
  - COMMITTED : 커밋 완료되어 종료될 트랜잭션
  - COMMITTING : 커밋중인 트랜잭션
  - ABORTED : 롤백되어 종료될 트랜잭션
  - KILLED : 서버에 의해 강제 종료 중인 트랜잭션

데이터베이스 잠금 확인

- 주어진 데이터베이스에 연결된 트랜잭션들이 테이블 및 레코드에 대한 잠금(lock) 상태를 확인
  - 서버 및 연결된 클라이언트들의 잠금 설정 정보
  - 잠금이 설정된 테이블 및 레코드에 대한 정보

- 사용 예

```
# demodb 에 대한 잠금 상태를 lockdb.log 파일에 출력(-o lockdb.log) 한다.
# HA 환경일 경우 demodb@localhost
$ cubrid lockdb -o lockdb.log demodb
```

데이터베이스 잠금 확인 - 계속

- 잠금 정보의 의미

```
$  cubrid lockdb demodb

*** Lock Table Dump ***
 Lock Escalation at = 100000, Run Deadlock interval =  1.00

Transaction (index  1, broker1_cub_cas_1, DBA@db1|123289)
Isolation COMMITTED READ
State TRAN_ACTIVE
Timeout_period :10.00

Transaction (index  2, csql, DBA@db1|123706)
Isolation COMMITTED READ
State TRAN_ACTIVE
Timeout_period : Infinite wait
:

OID =  0|  3393|  2
Object type: Instance of class ( 0|  203|  3) = tbl.
MVCC info: insert ID = 7, delete ID = 8.
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, Count =  2
LOCK WAITERS:
   Tran_index =  2, Blocked_mode =  X_LOCK
                    Start_waiting_at = Thu Aug 15 16:32:15 2019
                    Wait_for_secs = 10.00
```

잠금 설정: 잠금 에스컬레이션, deadlock 상태 탐지 간격

연결된 모든 클라이언트 정보: 트랜잭션 인덱스, 프로그램 이름, 데이터베이스 사용자 ID, 격리 수준, 잠금 타임아웃 정보 등

객체 잠금 정보: 클라이언트가 가진 잠금 정보, 잠금을 얻기 위해 기다리는 클라이언트 정보

트랜잭션이 얼마간 기다리고 있는지에 대한 확인 가능. 기다리는 시간이 길어지면 잠금을 가지고 있는 트랜잭션을 강제 종료 여부 판단 필요.

데이터베이스 사용량

- 데이터베이스 전체 사용량 확인
  - 데이터베이스의 전체 볼륨에 대한 사용량을 확인한다.

- 사용 예

```
# demodb 에 대한 데이터베이스 사용량을 확인
$ cubrid spacedb 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             3        35.7 G         24.8 G         60.5 G
PERMANENT  TEMPORARY DATA             1        63.0 M         39.9 G         40.0 G
TEMPORARY  TEMPORARY DATA             0         0.0 M          0.0 M          0.0 M
-                     -               4        35.7 G         64.8 G        100.5 G

Space description for all volumes:
volid  type         purpose        used_size free_size total_size      volume_name
    0   PERMANENT   PERMANENT DATA   512.0 M     0.0 M   512.0 M   /DATA/testdb/testdb
    1   PERMANENT   PERMANENT DATA    35.2 G     4.8 G    40.0 G   /DATA/testdb/testdb_x001
    2   PERMANENT   PERMANENT DATA     1.0 M    20.0 G    20.0 G   /DATA/testdb/testdb_x002
    3   PERMANENT   TEMPORARY DATA    63.0 M    39.9 G    40.0 G   /DATA/testdb/testdb_x003

LOB space description file:/clob/rms_db_clob

# 서버가 종료된 상태에서(-S) demodb 에 대한 데이터베이스 사용량을 확인
$ cubrid spacedb -S testdb

# HA 환경에서 현재 서버에 구동된 demodb 에 대한 데이터베이스 사용량을 확인
$ cubrid spacedb testdb@localhost
```

볼륨 종류별 요약 정보

볼륨별 요약 정보

CUBRID

데이터베이스 주요 로그

- CUBRID 운영 로그
  - 데이터베이스 운영부터 각종 유틸리티까지 CUBRID 명령어 사용에 대한 로그를 기록한다. 이를 통해 데이터베이스 서버 종료 등의 유틸리티 사용과 그 수행 시간을 확인할 수 있다.
  - $CUBRID/log/cubrid_utility.log

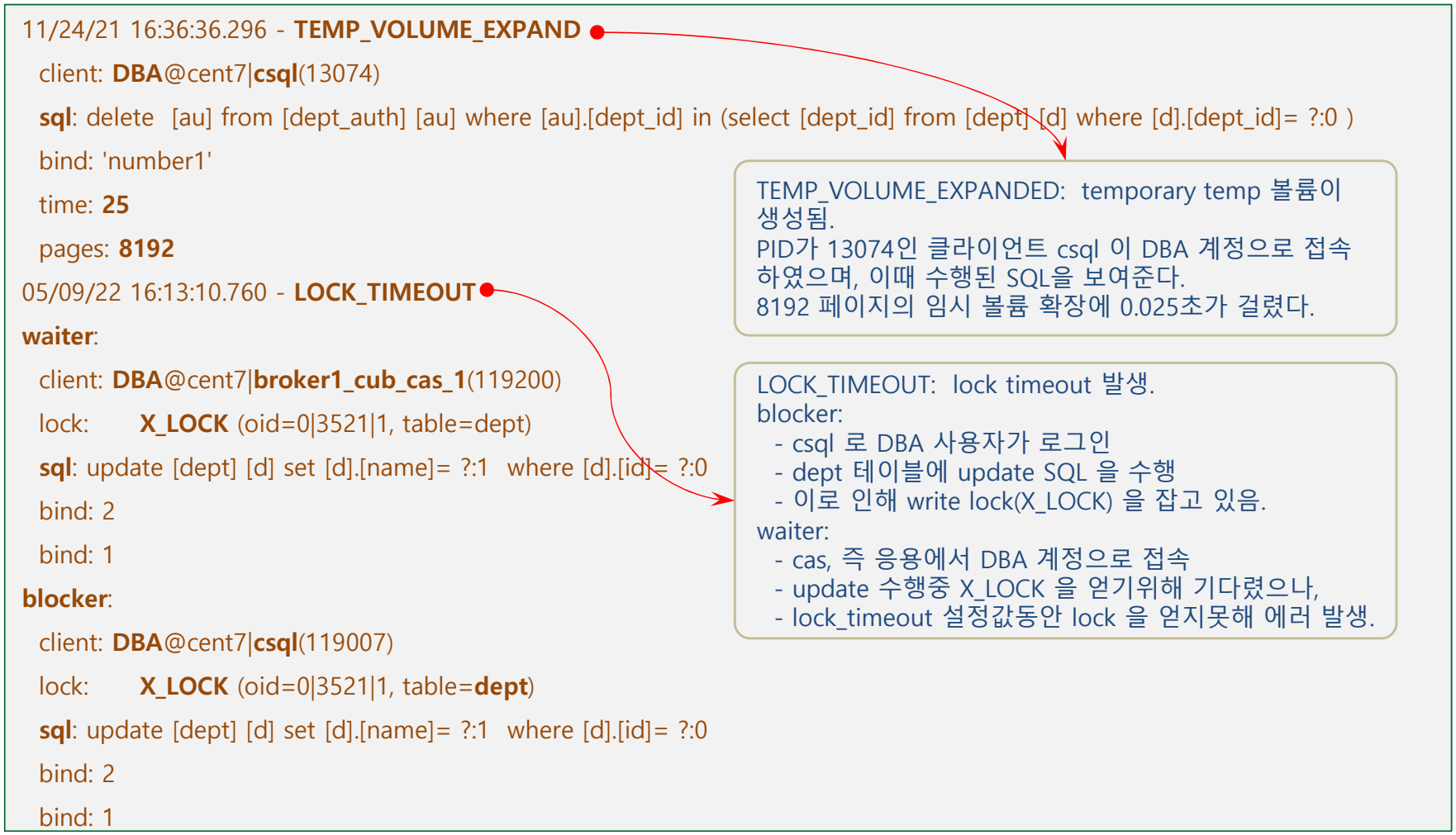
```
22-05-01 12:17:42.143 (114739) cubrid server start demodb
22-05-01 12:17:46.247 (114739) SUCCESS
22-05-01 12:34:06.616 (115217) cubrid broker start
22-05-01 12:34:07.136 (115217) SUCCESS
22-05-01 12:37:25.114 (115299) cubrid tranlist demodb
22-05-01 12:37:25.124 (115299) FAILURE: tranlist: The hostname on the database connection string should be
      specified when multihost is set in "databases.txt".
```

- 데이터베이스 서버 에러 로그
  - 데이터베이스 서버 운영중 발생한 에러 기록한다. 이를 통해 데이터베이스 서버 운영중 발생한 장애 원인을 파악하는데 도움을 얻을 수 있다.
  - $CUBRID/log/server/<db_이름>_<구동일자(yyyymmdd)>_<server_PID>.err

```
Time: 01/05/22 19:32:03.680 - ERROR *** file /home/jenkins/workspace/cubrid_release_11.0/src/storage/btree.c, line
27894 ERROR CODE = -670, Tran = 3, CLIENT = cent7:csql(63630), EID = 5
Operation would have caused one or more unique constraint violations. INDEX pk_tbl_id(B+tree: 1|2560|2561) ON
CLASS tbl (CLASS_OID: 0|195|12). key: 1(OID: 1|2625|3).
```

데이터베이스 주요 로그 - 계속

- CUBRID 이벤트 로그
  - 데이터베이스 서버 운영중 발생한 일부 이벤트를 기록한다. 이를 통해 데이터베이스 서버 운영중 발생한 몇 몇 이벤트에 대한 원인을 파악하는데 도움을 얻을 수 있다.
  - $CUBRID/log/server/<db_이름>_<구동일자(yyyymmdd)>_<server_PID>.event



데이터베이스 주요 로그 - 계속

- 브로커 sql log
  - 응용에서 처리를 요청한 SQL 에 대하여 SQL 및 트랜잭션 별 수행 시간 기록
  - SQL 로그 분석을 통한 처리 성능 분석
  - $CUBRID/log/broker/sql_log/<broker 이름>_<cas ID>.sql.log

CUBRID

데이터베이스 주요 로그 - 계속

- 브로커 slow log
  - sql log 와 동일하나 특정 시간 이상 수행된 SQL에 대하여 별도의 로그를 남긴다.
    - LOG_QUERY_TIME(cubrid_broker.conf) 로 설정하며, 로그 화일은 .sql.log 가 아닌 .slow.log 로 남는다.
  - 이를 통해 수행이 오래 걸리는 SQL 을 별도로 확인할 수 있다.

- 브로커 CAS 에러 로그
  - SQL 수행중 발생한 에러가 기록되어 있으며, 에러를 발생시킨 SQL은 동일한 broker-cas 의 SQL log 에 기록되어 있으므로, 발생 시점을 확인하여 관련 SQL 을 찾을 수 있다.
  - $CUBRID/log/broker/error_log/<broker 이름>_<cas ID>.err

```
Time: 20-02-28/10 13:45:17.687 - SYNTAX ERROR *** ERROR CODE = -493, Tran = 1, EID = 38
Syntax: Unknown class "unknown_tbl". select * from unknown_tbl
```


CUBRID

14. 환경 설정

CUBRID 기본 환경설정

- 기본 환경 정보 (.cubrid.sh)
  - CUBRID 운영에 필요한 CUBRID 엔진의 위치 및 명령 수행을 위한 파일 path, 라이브러리 path 등을 설정한다.
  - 사용자 계정 홈 디렉토리에 위치하며, 기본적으로 .bash_profile 을 통해 로그인시 사용된다.

```
CUBRID=/home/cubrid/CUBRID
CUBRID_DATABASES=$CUBRID/databases
if [ "x${LD_LIBRARY_PATH}x" = xx ]; then
  LD_LIBRARY_PATH=$CUBRID/lib
else
  LD_LIBRARY_PATH=$CUBRID/lib:$LD_LIBRARY_PATH
fi
SHLIB_PATH=$LD_LIBRARY_PATH
LIBPATH=$LD_LIBRARY_PATH
PATH=$CUBRID/bin:$PATH
      :
```

데이터베이스 환경설정

- 데이터베이스 환경설정 파일 ($CUBRID/conf/cubrid.conf)
  - 데이터베이스 운영에 필요한 환경설정 값들이다.
  - 데이터베이스별로 별도의 값을 설정할 수 있으며, 메모리 사용이나 접속 가능한 클라이언트의 개수 등은 개별로 설정하는 것이 좋다.

```
# 모든 데이터베이스 공통 설정
[commom]
data_buffer_size=5G

# demodb 만을 위한 설정
[@demodb]
data_buffer_size=10G

# 독립 모드(stand-alone)로 접속시 사용할 설정(CUBRID 9.3 이상)
[standalone]
data_buffer_size=1G
```

  - 서버와 클라이언트 설정값으로 나누어 진다.
    - 설정값 변경시 서버 설정이면 데이터베이스 서버를, 클라이언트 설정이면 broker 나 csql 등을 재구동 해야 적용된다.
    - SQL을 사용하여 클라이언트 설정값을 변경한 경우 해당 클라이언트의 구동중에만 적용하다.
  - 설정값 작성 규칙
    - 대/소문자를 구분하지 않는다. 보편적으로 소문자를 사용한다.
    - 설정 이름과 설정값은 동일한 라인에 입력되어야 한다.
    - 등호 기호(=)를 사용하며, 양 옆에는 공백문자를 사용할 수 있다.
    - 설정값이 문자열인 경우 따옴표 없이 문자열만 입력하며, 문자열에 공백 문자가 포함된 경우만 따옴표를 사용한다.

데이터베이스 환경 설정 - 계속

- 접속 설정
  - cubrid_port_id
    - 클라이언트가 데이터베이스 서버 접속시 사용하는 포트로 마스터 프로세스가 사용하며, 기본값은 1523 이다.
  - max_clients
    - 하나의 데이터베이스 서버로 동시에 연결 가능한 클라이언트의 초대 개수를 지정하며, 기본값은 100 이다.
    - 브로커에서 접속가능한 최대값(MAX_NUM_APPL_SERVER)과 backup, csql 등의 용도로 10개 이상을 합한 값으로 설정할 것을 권장한다.

- 메모리 설정
  - data_buffer_size
    - 데이터베이스 서버가 메모리 내에 캐시하고 있는 데이터 버퍼의 크기이며 기본값은 512M 이다.
    - 실제 데이터베이스의 크기 및 실제 메모리의 크기, 기타 다른 프로세스의 개수 및 크기를 고려해서 설정해야 한다.
    - 이 값을 너무 크게 설정하면 시스템 가용 메모리 부족으로 인해 메모리 스와핑(swapping)이 발생할 수 있다.
  - sort_buffer_size
    - order by, group by, 인덱스 생성 등 정렬이 필요할 경우 사용되는 버퍼의 크기로 클라인트 1개당 설정되는 값이며, 기본값은 2M 이다.
      - 인덱스 생성 시 정렬 버퍼가 많이 필요하므로 늘리는 것이 성능에 유리하지만 너무 과도하게 설정될 경우 불필요하게 많은 메모리를 점유할 수 있다.
    - 서버는 각 클라이언트의 정렬 요청마다 하나의 정렬 버퍼를 할당하며, 정렬을 완료한 후에는 할당되었던 버퍼 메모리를 해제한다.

데이터베이스 환경 설정

- 로깅 설정
  - log_buffer_size
    - 메모리에 캐시 되는 로그 버퍼의 크기를 설정하는 것으로, 기본값은 256M 이다.
    - 데이터베이스 수정 연산이 많고 길고 큰 트랜잭션이 많은 경우, 이 값을 크게 하면 디스크 I/O 감소로 성능이 향상될 수 있다.
  - force_remove_log_archives
    - 보관 로그를 정해진 개수(log_max_archives)만큼만 남기도록 설정하는 것으로, 기본값은 yes 이다.
    - HA 환경에서는 no 로 설정하여야 한다.
      - 로그가 master/slave 간 데이터 복제를 위해서도 사용되므로 미처 복제되지 못한 정보가 지워질 수 있다.
      - 복제를 위한 정보가 전달된 보관 로그는 정해진 개수를 초과하면 삭제된다.
  - log_max_archives
    - 보관 로그를 보관할 최대 개수를 설정하는 것으로, 설치시 설정된 기본값은 0 이다.
      - 트랜잭션이 종료되지 않았거나, 로그의 내용이 데이터베이스 데이터 볼륨이 반영되지 않았다면 이 값을 초과한 보관로그가 남아있을 수 있다.
    - force_remove_log_archives 가 yes 이거나 HA 환경일때 적용된다.

- HA 설정
  - ha_mode
    - HA 사용 여부를 설정하며, 기본값은 off 이다.
    - HA 사용시 on 로 설정하여야하며, 설정값을 변경한 경우 모든 CUBRID 서비스를 재구동하여야 한다.

데이터베이스 환경 설정 - 계속

- 동시성 설정
  - lock_timeout
    - 다른 트랜젝션이 레코드 또는 테이블에 대한 잠금을 가지고 있어 그 잠금을 얻기 위해 기다리는 시간으로 , 기본값은 -1 이다.
    - -1일 경우 무제한으로 기다리며, 일반적으로 권장하는 값은 10s(초)이다.
    - 1min 으로 설정하면 1분, 1h 로 설정하면 1시간이며, 0일 경우 대기하지 않는다.
    - 잠금을 얻지 못하면 질의에 대하여 locktimeout 에러가 발생한다.
  - lock_escalation
    - 레코드에 대한 잠금을 테이블 단위 잠금으로 변경하는 기준이 되는 레코드 잠금의 개수를 설정하며, 기본값은 100,000 이다.
    - 설정 값이 작으면, 메모리 잠금 관리에 의한 오버헤드가 적은 반면 동시성은 줄어든다.
    - 반대로 설정 값이 크면 메모리 잠금 관리에 의한 오버헤드가 큰 반면 동시성이 향상된다.
  - isolation_level
    - 트랜잭션의 고립 수준을 설정한다. 이 값에 따라 데이터베이스에 대한 잠금을 잡고 푸는 방법의 차이가 발생한다. 기본값은 4(TRAN_READ_COMMITTED) 이다.
      - TRAN_READ_COMMITTED : commit 된 데이터만을 읽을 수 있다. 변경이 진행중인 데이터는 MVCC 에 의해 변경전의 데이터를 읽는다.

데이터베이스 환경 설정 - 계속

- 디스크 설정
  - temp_file_max_size_in_pages
    - 질의 수행 시 생성된 permanent temp 볼륨이 부족할 경우 temporary temp 볼륨이 생성되는데, 이 볼륨의 크기를 지정한다.
    - 기본값은 -1로 필요한 만큼 제한없이 생성할 수 있으나, 디스크의 공간 부족이 발생할 수 있다.
      - 0 이면 생성하지 않는다. 공간이 부족하게된 질의는 에러 처리된다.
    - 예상되는 permanent temp 볼륨을 미리 확보하고, 디스크 여유 공간을 감안하여 생성될 수 있는 크기를 적절히 제한하는 것이 좋다.

- 오류 메시지 설정
  - error_log_level
    - 에러 로그 저장 수준을 지정한다. 기본값은 NOTIFICATION 이다. 너무 많은 에러가 남는 경우 SYNTAX 로 조정하여도 된다.
  - error_log_warning
    - 경고 메시지를 로그로 남길 것인지를 지정하며, 기본값은 no 이다.
  - call_stack_dump_on_error
    - 데이터베이스 서버에서 오류가 발생했을 때 에러 내용을 좀더 자세히 알 수 있도록 수행된 함수에 대한 콜-스택을 남길지에 대한 설정이며, 기본값은 no이다.

데이터베이스 환경 설정 - 계속

- 질의 캐시 설정
  - max_plan_cache_entries
    - 질의 실행 계획이 메모리에 캐싱되는 개수를 설정하며, 기본값은 1000 이다.

- 기타 설정
  - service
    - CUBRID 서비스 시작 시 자동으로 구동되는 서비스를 설정한다. 기본값은 server,broker,manager 이다.
    - server 는 기본적으로 마스터 프로세스가 구동되며, 실제 데이터베이스 서버는 다음의 server 항목에 등록된 데이터베이스만 자동으로 구동된다.
    - HA 환경일 경우 server 대신 heartbeat 을 등록하여 HA 가 자동으로 구동되도록 할 수 있다.
  - server
    - CUBRID 서비스 시작 시 자동으로 구동되는 데이터베이스 서버를 설정한다. 여러 개일 경우 쉼표(,)로 구분한다.
  - Java_stored_procedure
    - JAVA 로 작성된 Stored procedure 를 사용하기 위한 설정값이다. 기본값은 no 이다.
    - 이 설정을 사용하기 위해서는 Java 가상 머신을 사용하기 위한 JVM 에 대한 path 도 설정되어 있어야 한다.
      - 예) LD_LIBRARY_PATH=$JAVA_HOME/jre/lib/amd64:$JAVA_HOME/jre/lib/amd64/server
  - block_ddl_statement
    - 데이터 정의 문(Data Definition Language, DDL)을 실행되지 않도록 설정한다. 기본값은 no 이다.
  - block_nowhere_statement
    - UPDATE/DELETE문에 조건(Where)이 없는 SQL 을 실행되지 않도록 설정한다. 기본값은 no 이다.
  - ddl_audit_log
    - DDL 사용 기록을 남기는 설정으로, 기본값은 no 이다.
    - 설정시 $CUBRID/log/ddl_audit 아래에 클라이언트별 DDL 사용 기록이 남는다.
    - 이 설정을 통해 테이블이 삭제거나 했을때 원인 파악이 용이하다.

브로커 환경 설정

- 브로커 환경 설정 파일 ($CUBRID/conf/cubrid_broker.conf)
  - 브로커 운영에 필요한 환경 설정 값들이다.
  - 편집기를 통하여 수정 가능하며, 브로커를 재 구동해야 적용된다.
  - 설정값이 잘못된 경우 구동시 오류가 발생하며, 구동되지 않는다.

```
# 모든 브로커 공통 설정
[broker]
MASTER_SHM_ID           =30001
ADMIN_LOG_FILE          =log/broker/cubrid_broker.log

# query_editor 브로커 만을 위한 설정
[%query_editor]
SERVICE                 =ON
SSL                     =OFF
```

  - 설정값 작성 규칙
    - 대/소문자를 구분하지 않는다. 보편적으로 설정은 대문자, 설정값은 소문자를 사용한다. 단, 디렉토리 명은 대소문자 구분된다.
    - 설정 이름과 설정값은 동일한 라인에 입력되어야 한다.
    - 등호 기호(=)를 사용하며, 양 옆에는 공백문자를 사용할 수 있다.
    - 설정값이 문자열인 경우 따옴표 없이 문자열만 입력하며, 문자열에 공백 문자가 포함된 경우만 따옴표를 사용한다.

브로커 환경 설정

- 브로커 운영중 설정 변경
  - 브로커 운영중 일부 설정은 재구동없이 변경 가능하다.
  - 단 이렇게 변경된 설정은 브로커 구동중에만 유효하며, 재구동시 설정 값을 따른다.

```
# 특정 업무 사용을 위해 추가만 해두었던 브로커를 사용 가능하게 만든다.
$ cubrid broker on broker10

# 사용량의 증가로 10초 이상 걸리는 SQL 에 대한 slow log 를 남기도록 설정한다.
$ broker_chanager broker10 long_query_time 10
OK
```

- 동적 변경이 가능한 설정값 일부는 다음과 같으며, 자세한 내용은 매뉴얼을 참고한다.
  - ACCESS_MODE, SESSION_TIMEOUT, SQL_LOG_MAX_SIZE, TIME_TO_KILL
  - LONG_QUERY_TIME, LONG_TRANSACTION_TIME, MAX_QUERY_TIMEOUT
  - APPL_SERVER_MAX_SIZE, APPL_SERVER_MAX_SIZE_HARD_LIMIT
  - MAX_PREPARED_STMT_COUNT

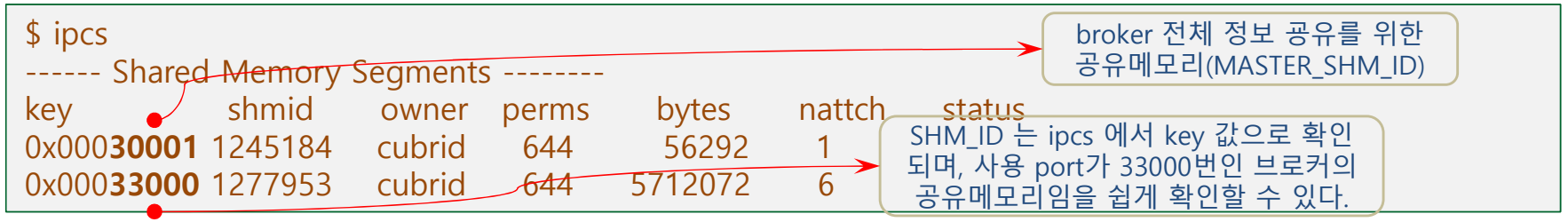
broker 환경 설정 - 계속

- 브로커 설정 정보 확인
  - **cubrid broker info** 명령을 통해 현재 "실행 중"인 브로커의 설정 값을 확인할 수 있다.

```
$ cubrid broker info
#
# cubrid_broker.conf
#
# broker parameters were loaded from the files
# /home/cubrid/CUBRID/conf/cubrid_broker.conf
# broker parameters
[broker]
MASTER_SHM_ID            =30001
ADMIN_LOG_FILE           =log/broker/cubrid_broker.log

[%query_editor]
SERVICE                  =ON
SSL                      =OFF
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
CCI_DEFAULT_AUTOCOMMIT   =ON
:
```

broker 환경 설정 - 계속

- 브로커 접속 설정
  - BROKER_PORT
    - 응용에서 접속하기 위한 브로커의 포트 번호를 설정한다. 1024 ~ 49151 의 값을 설정한다.
      - 65535까지 사용 가능하나, 49152 이상은 동적할당 포트로 사용하지 않는 것이 좋다.
  - APPL_SERVER_SHM_ID
    - broker CAS간 정보 공유를 위해 사용하는 공유 메모리의 ID를 설정하며, 시스템 내에서 유일한 값이어야 한다.
    - 운영상 공유 메모리 확인의 편의를 위해서 브로커 포트와 같은 값을 권장한다.



  - ACCESS_MODE
    - 데이터베이스에 대한 접근 모드를 설정하며, 기본값은 RW 이다.
    - 일반적으로는 설정하지 않으며, 부하분산을 위해 Stand-by 서버를 읽기 전용으로 사용할때 설정할 수 있다.
    - 설정 가능한 모드는 RW, RO, SO 가 있으며, 각각은 다음과 같다.
      - Read Write(RW): 읽기와 쓰기가 가능하다.
      - Read Only(RO): 읽기만 가능하다.
      - Standby Only(SO): Stand-by 서버에만 접속 가능하며, 읽기만 가능하다.

broker 환경 설정 - 계속

- 브로커 연결 설정 (구동되는 CAS의 수량과 관련됨)
  - MIN_NUM_APPL_SERVER
    - 브로커 구동시 구동되는 CAS 프로세스의 최소 개수를 설정하는 것으로, 기본값은 5 이다.
  - MAX_NUM_APPL_SERVER
    - 구동되어 있는 CAS 의 개수를 초과하여 요청이 들어오는 경우, CAS 가 추가적으로 구동되어 요청을 받을 수 있으며, 이렇게 구동될 수 있는 CAS의 최대 개수를 설정하는 것로, 기본값은 40 이다.
  - TIME_TO_KILL
    - 자동 추가된 CAS 가 일정 시간이상 요청을 받지 않으면 해당 CAS 를 종료시키는데, 이 시간을 설정하는 값이며, 기본값은 120(초) 이다.
      - IDLE 또는 CLOSE_WAIT 인 상태에 적용된다. was 의 경우 connection pooling 을 사용하고 있는데 pool 의 개수가 CAS 의 개수보다 많은 경우, 사용자 증가시 CAS 가 증가될 수 있다. 이렇게 늘어난 CAS 는 TIME_TO_KILL 에 의해 종료될 수 있으며 이럴 경우 connection pool 에서 제거되어야 한다.

- 브로커 접속 보안 설정
  - SSL
    - 응용에서 브로커로 접속시 TLS 를 이용한 패킷 암호화를 설정하며, 기본값은 OFF 이다.
    - SSL 설정시 SSL 을 이용하지 않는 경우 접속할 수 없다. 반대로 미설정시 SSL 을 이용한 접속도 제한된다.
  - ACCESS_CONTROL
    - 브로커에 접속하는 응용 클라이언트를 제한하기 위한 설정으로 기본값은 OFF 이다.
    - 접속이 가능한 데이터베이스, 데이터베이스 사용자, IP 등을 지정할 수 있다.
  - ACCESS_CONTROL_FILE
    - 브로커에 접속이 가능한 데이터베이스 이름/데이터베이스 사용자/IP 목록을 저장한 파일을 설정한다.
    - 자세한 내용은 매뉴얼을 참고한다.

CUBRID

broker 환경 설정 - 계속

- 트랜잭션 및 질의 설정
  - LONG_QUERY_TIME
    - 브로커 slow log 에 기록될 수행이 오래 걸리는 SQL 의 기준 시간을 설정하며, 기본 값은 60 (초)이다.
    - 이 값보다 오래 걸리는 SQL 이 slow log 에 기록된다.
  - LONG_TRANSACTION_TIME
    - LONG_QUERY_TIME 과 유사하며, SQL 수행 시간이 아닌 트랜잭션 수행 시간을 기준으로 한다. 기본값도 동일하다.
  - MAX_QUERY_TIMEOUT
    - 질의 수행시 너무 오래 걸리는 경우 타임아웃을 설정하여 설정 시간을 초과하면 질의를 멈추고 롤백시킨다.
    - 기본값은 0 으로, 질의 수행이 끝날때까지 무한 대기한다.
  - SESSION_TIMEOUT
    - 트랜잭션이 시작된 후 응용에서 일정시간동안 아무런 요청이 없는 경우 트랜잭션을 롤백시키고 연결을 종료한다. 이를 위한 기준시간을 설정하는 것으로 기본값은 300 (초)이다
      - 트랜잭션의 시작은 별도로 지정하지 않으며, auto commit off 상태에서 select 도 트랜잭션의 시작으로 간주된다.
  - 시간 설정 방법
    - 값 뒤에 ms, s, min, h을 붙여 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다.
    - 지정하지 않은 경우 초 단위로 설정된다.

broker 환경 설정 - 계속

- 브로커 로그 설정
  - SQL_LOG
    - 브로커를 통해 실행되는 모든 SQL 을 기록하며, 기본값은 ON이다.
  - LOG_DIR
    - SQL 로그가 저장되는 디렉터리로, 기본값은 log/broker/sql_log 이다.
    - 이하 설정되는 모든 디렉토리는 $CUBRID 아래를 기준으로 하며, 절대 path 지정도 가능하다.
  - SLOW_LOG
    - SLOW SQL/TRANSACTION 에 대한 로그를 남길지를 설정하며, 기본값은 ON 이다.
    - LONG_QUERY_TIME, LONG_TRANSACTION_TIME 을 초과하거나 에러가 발생한 SQL/트랜잭션이 기록된다.
  - SLOW_LOG_DIR
    - SLOW SQL 로그 파일이 생성되는 디렉터리를 지정한다. 기본값은 log/broker/sql_log이다.
  - SQL_LOG_MAX_SIZE
    - SQL 로그 및 SLOW SQL 로그 파일의 최대 크기를 설정하며, 기본값은 10000(10M) 이다.
      - 1G, 100M 로도 설정이 가능하다.
    - SQL 수행후 로그 화일의 크기가 위 설정 값을 초과하는 경우 .bak 로 보관하며, .bak 는 overwrite 된다.
  - ERROR_LOG_DIR
    - cas 에서 요청 수행중 발생한 에러를 기록할 디렉토리를 지정하며, 기본값은 log/broker/error_log 이다.

broker 추가

- 브로커(BROKER) 추가
  - 특정 업무를 위해 별도의 브로커를 추가하거나, 데이터베이스 추가시 브로커를 추가한다.
  - 브로커는 관리하는 CAS 가 데이터베이스와 연결후 그 응용에서 연결 해제를 하더라도, connection 을 유지하고 있다.
    - 새로운 데이터베이스와 작업을 하게 되는 경우 기존 연결을 끊고 새로 연결을 맺게된다.
    - 이러한 connection overhead 를 줄이기 위해 데이터베이스별로 브로커를 추가하여 관리하는 것이 좋다.
    - 또한 브로커 별로 cas의 sql log 가 구분되므로 관리측면에서도 하나의 데이터베이스에 관련된 SQL과 트랜잭션을 관리하기 용이하다.
    - 추가한 브로커는 재구동 후 사용할 수 있다.
  - 브로커 추가는 cubrid_broker.conf 의 기존 브로커 하나의 내용을 전부 복사하여 추가한다.
    - 추가한 후, 브로커 이름과, BROKER_PORT, APPL_SERVER_SHM_ID 를 중복되지 않게 변경한다.
    - BROKER_PORT 는 CAS들이 사용하는 포트 번호를 확보해야 하므로, 이전 BROKER 포트 값에 MAX_NUM_APPL_SERVER 를 합한 값보다 크게 하여야 하며, 보통 1000 간격으로 한다.

```
# 기존 내용
[%BROKER1]
SERVICE                 =ON
SSL                     =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
```

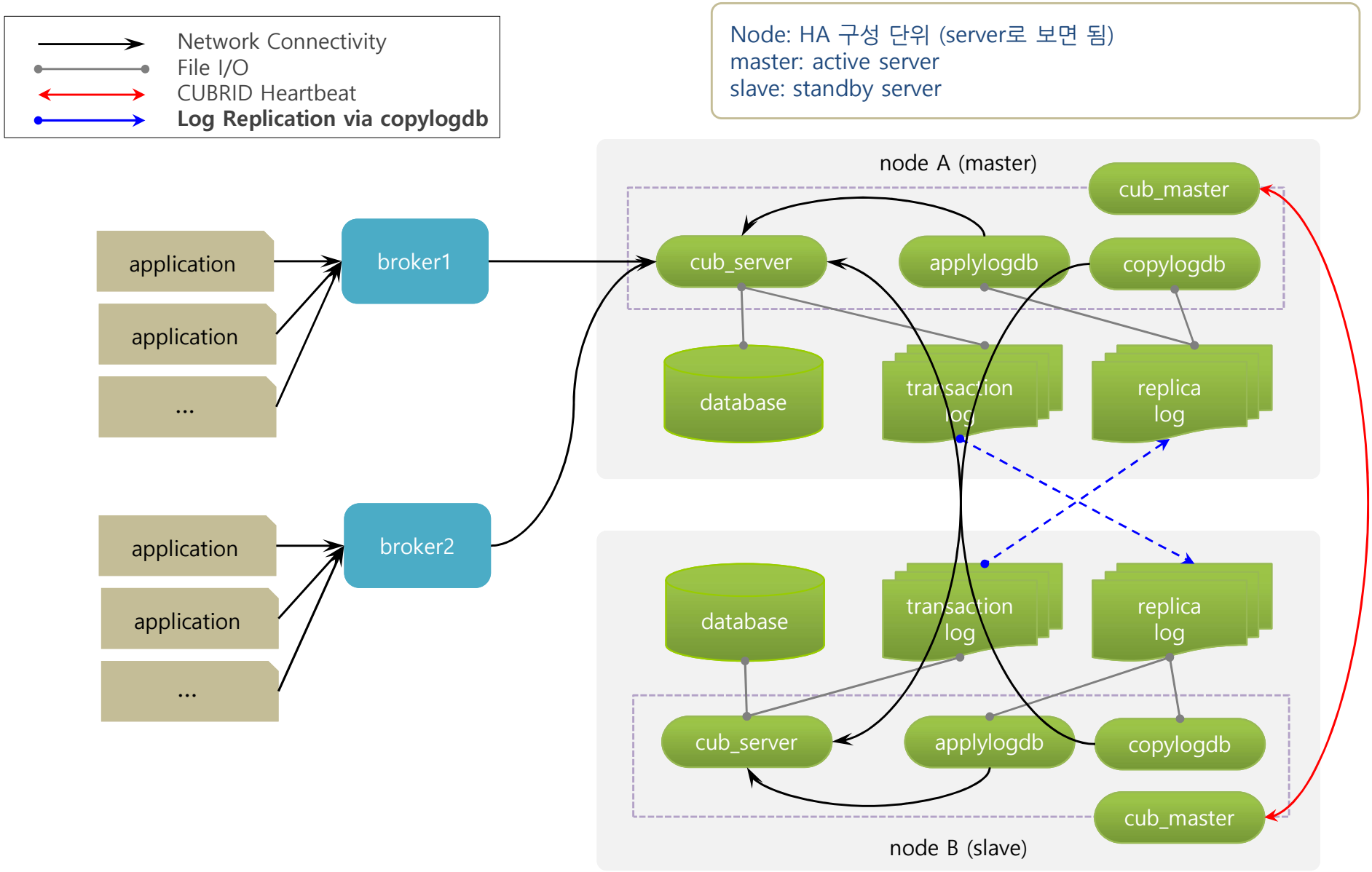
```
# BROKER1에 대한 전체 내용 복사후 브로커이름 등 수정
[%BROKER2]
SERVICE                 =ON
SSL                     =ON
BROKER_PORT             =34000
MIN_NUM_APPL_SERVER     =5
MAX_NUM_APPL_SERVER     =40
APPL_SERVER_SHM_ID      =34000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR           =log/broker/error_log
SQL_LOG                 =ON
TIME_TO_KILL            =120
SESSION_TIMEOUT         =300
KEEP_CONNECTION         =AUTO
CCI_DEFAULT_AUTOCOMMIT  =ON
```

15. HA

CUBRID HA 구조

- 개요
  - High Availability(고가용성) 을 위해 Active 서버와 Stand-by 서버로 구성
    - Active 서버: 읽기, 쓰기가 가능한 서버로 마스터 노드가 속해 있다.
    - Stand-by 서버: 주로 대기상태로 있으며, 슬레이브 노드가 속해 있다. 읽기만 가능하다.
      - 읽기 요청이 많은 서비스의 경우 Stand-by 서버를 읽기 전용으로 서비스에 활용할 수 있다.
      - 읽기 전용으로 서비스할 경우, 응용에서 읽기만을 가진 트랜잭션에 대하여 적절하게 Active 와 Stand-by 간 연결(connection) 배분을 해주어야 한다.
  - Active 서버 장애로 서비스 불가시 Stand-by 서버로 자동 절체(fail-over)되어 중단시간을 최소화하여 서비스가 유지되도록 한다.
    - Active 서버와 Stand-by 서버의 데이터베이스는 항상 동기화된 상태를 유지
    - Active 서버와 Stand-by 서버의 상태를 heartbeat 메시지를 이용하여 실시간으로 확인
  - Active 서버와 Stand-by 서버는 완전히 독립된 별개의 서버로, 저장공간을 공유하지 않는 shared-nothing 구조를 가진다.
    - Active 서버와 Stand-by 서버의 데이터베이스간 데이터 동기화가 필요하며, 이를 위해 다음의 과정이 수행된다.
      - 트랜잭션 로그 다중화: Active 서버에서 생성되는 트랜잭션 로그를 실시간으로 다른 노드에 복제한다.
      - 트랜잭션 로그 반영: 실시간으로 복제된 트랜잭션 로그를 분석하여 자기 노드의 데이터베이스 서버에 데이터를 반영한다.
  - Active 와 Stand-by 는 별도 설정되지 않으며, 먼저 구동된 쪽이 Active 가 된다.
    - 구동 및 종료시 주의하여야 한다. Active 를 먼저 구동하여야 하며, Stand-by 를 먼저 종료시켜야 한다.
    - Active 를 먼저 종료시 Stand-by 로 절체가 발생하게 된다. 이후 사용자가 Active 와 Stand-by 였던 서버를 혼돈하여 구동시 동기화에 문제가 발생할 수도 있다.

CUBRID HA 구조도



CUBRID

CUBRID HA 기본 개념

- 노드
  - HA를 구성하는 논리적인 단위
    - 마스트 노드: Active 서버를 사용한 읽기, 쓰기 모두 가능하며, 복제의 대상이 되는 노드다.
    - 슬레이브 노드: 마스터 노드와 동일한 내용을 가지는 노드로, 마스터 노드의 변경 내용이 자동으로 반영된다. Stand-by 서버를 사용한 읽기 전용 서비스가 가능하며, 마스터 노드 장애 시 자동 절체에 의해 마스터 노드가 될 수 있다.
    - 레플리카 노드: 슬레이브 노드와 동일하나, 마스터 노드 장애시 이 노드로의 절체가 일어나지 않는다.

- 프로세스
  - HA 동작을 위해 필요한 프로세스는 다음과 같으며, HA 구동시 구동된다.
    - cub_master: heartbeat 을 통해 서로의 상태를 확인하며, 마스터 노드 장애시 절체를 시킨다.
    - cub_server: 데이터베이스 서버
    - copylogdb: 상대노드의 데이터베이스의 로그(트랜잭션 로그)를 복사해와서 복제 로그(replica log)를 만든다.
    - applylogdb: 복제 로그(replica log)의 내용을 자기 노드의 데이터베이스에 반영한다.

CUBRID HA 기본 개념 - 계속

- Heartbeat 메시지
  - HA 를 구성하는 여러 노드(마스터/슬레이브/레플리카) 간 상태를 확인하기 위해 주고 받는 메시지이다.
  - UDP 59901 포트를 이용하여 heartbeat check 를 수행한다.

- Fail-over와 Fail-back
  - Fail-over: 마스터 노드에 장애로 인해 서비스를 제공할 수 없는 상태가 되면 슬레이브 노드가 마스터 노드가 되는 것으로, 자동으로 fail-over 가 가능하다.
  - Fail-back: 장애가 발생했던 기존 마스터 노드의 장애가 복구되면 원래 마스터 노드를 다시 마스터 노드로 되돌리는 것이며, 수동으로 fail-back 을 처리하여야 한다.

- 복제 방식
  - 동기(sync) 방식: 트랜잭션 커밋 시점에 트랜잭션 로그가 상대편 노드에 복사되는데, 복사가 완료된 것이 확인되면 커밋을 완료한다. 스탠바이 노드에 트랜잭션 로그가 완전히 복사되는 것을 보장한다.
  - 비동기(async) 방식: 트랜잭션 로그가 상대편 노드에 복사 완료되었는지 확인하지 않고 마스터 노드의 트랜잭션 커밋을 완료된다, sync 방식보다 성능적으로 유리하지만 마스터 노드가 비정상 종료되면 노드 간의 데이터 불일치가 발생할 수 있다.

CUBRID HA 주의사항

- LINUX 환경만 지원

- 구축시 고려사항
  - 테이블에 기본키 필요
    - 데이터 동기화를 위해 복제하는 형태로, 수정/삭제를 위해 기본키가 존재하여야 함.
  - LOB 지원하지 않음
    - CLOB → varchar(1G), BLOB → bit varying(1G) 로 변경하여야 한다.
      - file 의 경우 128M-1byte 까지만 저장된다.
    - JDBC LOB method 는 변경하지 않아도 됨(CUBRID 10.2.1 이상)
  - trigger의 경우 CUBRID 9.2 이상 지원된다.
  - java SP 생성/변경 등은 Stand-by server 에 반영되나, class/jar 파일은 개별 설치해야 한다.
    - Master 에서 SP 생성후, class/jar 파일을 Slave 의 <database dir>/java 에 복사한다.
    - 또는 class/jar 파일을 Slave 의 임의의 위치에 복사한후, loadjava <database name> 을 수행한다.
- 사용시 고려사항
  - incr(), decr() 은 Active server 에서만 사용
  - 통계정보 갱신(update statistics)은 10.0 이후 Stand-by server 에 반영됨.
  - stand-alone mode 작업은 해당 서버에만 반영된다.
  - 데이터베이스 백업시 -r 옵션 사용 금지

- 다음의 경우 각 서버별 개별 작업 권장 (HA 중지)
  - 분할 작업이 어려운 대량 데이터 작업
    - 특히 UPDATE로 특정 컬럼을 모두 변경하는 경우
  - 레코드가 많은 테이블에 대하여 스키마(기본키 변경 제외)/ 인덱스 /권한 변경

CUBRID HA 구성 전 점검

- hosts 파일에 Master/Slave 서버 이름 등록 여부 확인
  - HA 구동 및 운영중 상호간 정보 확인을 위해 hostname 을 이용하므로, 반드시 /etc/hosts 에 hostname 이 등록되어 있어야 한다.
  - IP adress 가 여러 개일 경우 hostname 에 설정된 IP address 로 /etc/hosts 에 등록하여야 한다.

```
# hostname 확인 및 hostname 에 설정된 기본 IP address 확인
$ hostname; hostname -I
# hostname 과 IP address 가 올바른지 확인. (` 는 shift~)
$ ping -4 `hostname`

# Master 의 hostname/IP (db1/192.168.2.10), Slave 의 hostname/IP (db2/192.168.2.11) 를 Master/Slave 모두에 등록한다. (root 권한 필요)
root # vi /etc/hosts
192.168.2.10    db1
192.168.2.11    db2
```

- 방화벽 설정
  - Master/Slave 간 데이터 동기화에 필요한 port 오픈: 1523/tcp

```
# 1523/tcp port 방화벽에 등록 (root 권한 필요)
root # firewall-cmd --permanent --zone=public --add-port=1523/tcp
root # firewall-cmd --reload
root # firewall-cmd --list-ports
1523/tcp

# 또는 방화벽 내림
root # systemctl stop firewalld
# 재구동시 방화벽 사용하지 않도록 설정
root # systemctl disable firewalld
```

CUBRID HA 구성 전 점검 - 계속

- HA 제약 조건 검사
  - HA 구성전 HA 제약조건을 먼저 검사하여 문제가 있으면 조치를 취한후 HA 구성을 진행한다.

```
# 반드시 dba 계정으로 접속할 필요는 없다. 업무에서 사용되는 사용자 계정으로 접속한다. 단, 사용자 계정이 여럿일 경우 dba 로 접속하는 것이 편리하다.
$ csql -u dba demodb
```

앞서 재구성과정에서 데이터베이스를 demodb_bak 으로 이름 변경해 놓은 것을 이름을 다시 demodb 로 변경하여 사용한다.
$cubrid renamed demodb_bak demodb

  - primary key 확인

```
-- 다음 SQL 의 결과가 나오면 Primary Key 가 없는 테이블이다.
select owner_name, class_name
from   db_class
where class_type = 'CLASS' and is_system_class = 'NO'
and class_name not in
  ( select class_name from db_index  where is_primary_key ='YES'  )
order by owner_name, class_name
```

  - LOB 사용 확인

```
-- 다음 SQL 의 결과가 나오면 LOB 를 사용하는 테이블이다.
select class_name, attr_name, data_type
from   db_attribute
where data_type like '%LOB'
order by class_name, attr_name
```


CUBRID

CUBRID HA 구성 방법

- HA 기본 구성
  - Master/Slave 1:1 구성으로 CUBRID HA 고유기능인 장애 발생시 무 중단 서비스에 초점을 맞춘 구성이다.

- 데이터베이스 생성시 HA 구성
  - 데이터베이스 생성시 바로 HA 를 구성하는 방법이다.
  - Master 와 Slave 에 동일한 이름의 데이터베이스를 각각 생성하고 설정 조정으로 간단하게 구성할 수 있다.
    1. cubrid.conf 와 cubrid_ha.conf, databasese.txt 를 수정한다.
  - 터베이스를 생성하여 작업 완료후 HA 구성을 권장한다.
    - 특히 primary key 부재나 LOB 사용으로 HA 제약조건이 충족되지 않은 경우 동기화가 이루어지지 않기때문에 기본키 추가나 컬럼 타입 변경을 해주어야 하나, 이미 등록된 데이터는 동기화가 되어있지 않기에 재구성이 필요하다.
  - 테스트 운영중 Slave 를 구동하지 않아 데이터 동기화가 깨지는 경우가 발생한다.
    - Slave 구동시 동기화가 이루어지지만, 오랜 시간동안 Slave가 구동되어 있지 않았다면 동기화에 시간이 오랜 시간이 걸려 재구성하는 것이 유리하다.
  - HA 구성 방법
    1. Master와 Slave 모두 동일한 이름의 데이터베이스를 생성한다.
    2. HA 사용을 위한 설정(cubrid.conf, cubrid_ha.conf, databases.txt) 을 조정한다. 설정 조정 방법은 사용중인 데이터베이스의 HA 구성시와 동일하므로 그때 설명한다.
    3. Master 를 먼저 구동하여 구동 확인 후, Slave 를 구동한다.
    4. Primay key 를 가지고 LOB type 을 가지지 않는 테이블을 생성하여 사용한다.

CUBRID HA 구성 방법 - 계속

- 싱글로 사용중인 데이터베이스의 HA 구성
  - 다음 순서에 따라 Slave 로 데이터베이스 복제를 진행한다.

```
# demodb 를 HA 로 구성한다.
# backup 은 /data/backup 아래에, Master 는 db1, Slave 는 db2 로, OS 계정은 cubrid 로 가정한다.
# Master/Slave 를 구분하기 위해 shell prompt 부분을 M$ 와 S$ 로 구분하였으며, Master/Slave 모두 동일하게 진행되는 부분은 MS$ 로 하였다.

# 1. HA 제약 조건 검사.

# 2. CUBRID 서비스 종료. 드물게 종료가 완료되지 못하는 경우가 있어 csql Stand-alone 으로 접속후 종료한다.
M$ cubrid service stop
M$ csql -u dba -S demodb
csql> ;exit

# 3. HA 구성 대상 데이터베이스를 백업한다. 백업 파일의 위치는 여유공간이 충분해야 한다.
M$ mkdir /data/backup # root 작업이 필요할 수 있음. mkdir /data; chown cubrid. /data
M$ cubrid backupdb -S -z -D /data/backup demodb

# 4. 백업화일을 Slave 로 보내준다. scp/ftp/sftp/rsync 등을 이용하여 보낸다. 보내는 위치는 Master 와 동일하여야 한다. scp 설정하여 scp 를 이용한다.
S$ mkdir /data/backup # root 작업이 필요할 수 있음. mkdir /data; chown cubrid. /data
M$ scp /data/backup/demodb_bk* cubrid@db2:/data/backup
```

scp 는 연결을 허용하는 것으로 설정하므로 db2 에서 설정한다.
root 권한이 필요하며, /etc/hosts.allow 파일에 다음의 내용을 추가한다. 이미 sshd 가 있으면, 그 줄의 IP 만 추가하면 된다.
sshd: 192.168.2.

설정후 처음 scp 를 사용하는 경우 연결 계속 여부를 물어오는데 yes 를 입력하면 된다.

CUBRID

CUBRID HA 구성 방법 - 계속

- 싱글로 사용중인 데이터베이스의 HA 구성
  - 다음 순서에 따라 Slave 로 데이터베이스 복제를 진행한다.

```
# 5. Slave 에 데이터베이스가 만들어질 디렉토리를 생성한다. 백업을 이용한 복구 과정을 거치므로 Master
와 동일한 위치로 하여야 한다. demodb 의 위치는 databases.txt 를 통해 확인 가능하다.
M$ more $CUBRID_DATABASES/databases.txt
demodb          /home/cubrid/CUBRID/databases/demodb    localhost       /home/cubrid/CUBRID/databases/demodb
file:/home/cubrid/CUBRID/databases/demodb/lob
S$ mkdir /home/cubrid/CUBRID/databases/demodb

# 6. demodb 의 기본 정보를 Slave 에 등록한다. demodb 가 표시된 한 줄을 copy 해서 Slave 에 paste 한다.
M$ more $CUBRID_DATABASES/databases.txt
S$ cat >> $CUBRID_DATABASES/databases.txt
demodb          /home/cubrid/CUBRID/databases/demodb    localhost       /home/cubrid/CUBRID/databases/demodb
file:/home/cubrid/CUBRID/databases/demodb/lob <CR>^D

# 7. 백업 정보화일을 Slave 에 복사한다. 내용이 간단하므로 copy & paste 한다.
# 백업 정보화일(demodb_bkvinf) 은 demodb 가 위치한 곳에 있다.
M$ more /home/cubrid/CUBRID/databases/demodb/demodb_bkvinf
  0 0 /data/backup/demodb_bk0v000
S$ cat > /home/cubrid/CUBRID/databases/demodb/demodb_bkvinf
 0 0 /data/backup/demodb_bk0v000 <CR>^D

# 8. Slave 에서 백업 파일을 이용하여 demodb 를 복구한다.
S$ cubrid restoredb demodb
```

CUBRID HA 구성 방법 - 계속

- 싱글로 사용중인 데이터베이스의 HA 구성
  - 복제가 완료되었으므로 설정을 조정하고, HA 로 demodb 를 구동한다.

```
# 9. databases.txt 에 Master/Slave 서버 정보를 동록한다. 서버 정보인 localhost 를 수정한다.
MS$ vi $CUBRID_DATABASES/databases.txt
demodb          /home/cubrid/CUBRID/databases/demodb    db1:db2         /home/cubrid/CUBRID/databases/demodb
file:/home/cubrid/CUBRID/databases/demodb/lob

# 10. cubrid_ha.conf 에 Master/Slave 서버 정보와 데이터베이스 정보 등을 등록한다.
MS$ vi $CUBRID/conf/cubrid_ha.conf
[common]
ha_port_id=59901
ha_node_list=cubrid@db1:db2
ha_db_list=demodb
ha_apply_max_mem_size=300
ha_copy_sync_mode=sync:sync
ha_copy_log_max_archives=1

# 11. cubrid.conf 끝 부분의 HA 관련 설정을 수정/추가한다.
MS$ vi $CUBRID/conf/cubrid.conf
log_max_archives=20
ha_mode=on
force_remove_log_archives=no

# 12. Master 를 먼저 구동하여 구동 완료 확인후, Slave 를 구동한다.
M$ cubrid hb start
S$ cubrid hb start
```

CUBRID HA 구성 방법 - 계속

- JDBC 등 연결 설정
  - Master 서버의 장애로 인해 Slave 로 절체되었을때, 응용은 새로운 Master 로 연결한다.
  - 새로운 Master 로의 연결은 브로커에서도 해주지만, 기존 Master 서버 자체가 접속 불가한 상황이면 응용은 새로운 Master 의 브로커로 연결을 해야 한다.
  - 대체 서버(althost)를 지정하여 기본 서버로의 연결이 불가할 때 대체 서버로 연결되도록 설정한다.

```
# connection URL 에 althost 를 추가한다. (예, JDBC)
jdbc:cubrid:db1:33000:demodb:::?charset=utf-8&althosts=db2:33000
```

- Slave 를 부하분산을 위해 read-only 로 사용하는 경우
  - 절체되었을 때를 위해 별도의 브로커를 추가를 하는 것이 좋다. 절체된 경우 기존 Master 로 접속되던 부분이 새로운 Master 로 접속이 되면서 기본 브로커의 접속 허용치를 넘어설 수 있기 때문이다.
    - 접속 부하가 증가하게 되므로, 최대한 빨리 Master 를 복구하여 fail-back 을 시켜주어야 한다.
  - 추가적으로 부하분산 용임을 알기 쉽게 하기 위해 ReadOnly 설정을 추가한다.

```
# broker_ro 를 40000 포트로 추가하고, ReadOnly 설정을 한다.
# vi $CUBRID/conf/cubrid_broker.conf
[%broker_ro]
ACCESS_MODE                =RO
BROKER_PORT                =40000
  :
```

CUBRID HA 구동 및 종료

- 구동
  - heartbeat 명령에 의해 구동되며, 데이터 복제를 위한 프로세스들도 같이 구동된다.
  - Master/Slave 는 설정되지 않으며, 구동 순서에 의해 먼저 구동된 쪽이 Master 가 된다.
  - 구동 방법

```
# HA 를 구동한다.
cubrid hb start
```

  - CUBRID 서비스 구동시 같이 구동되지 않는다. 브로커 등 CUBRID 서비스와 함께 구동하기 위해서는 설정을 변경하여야 한다.

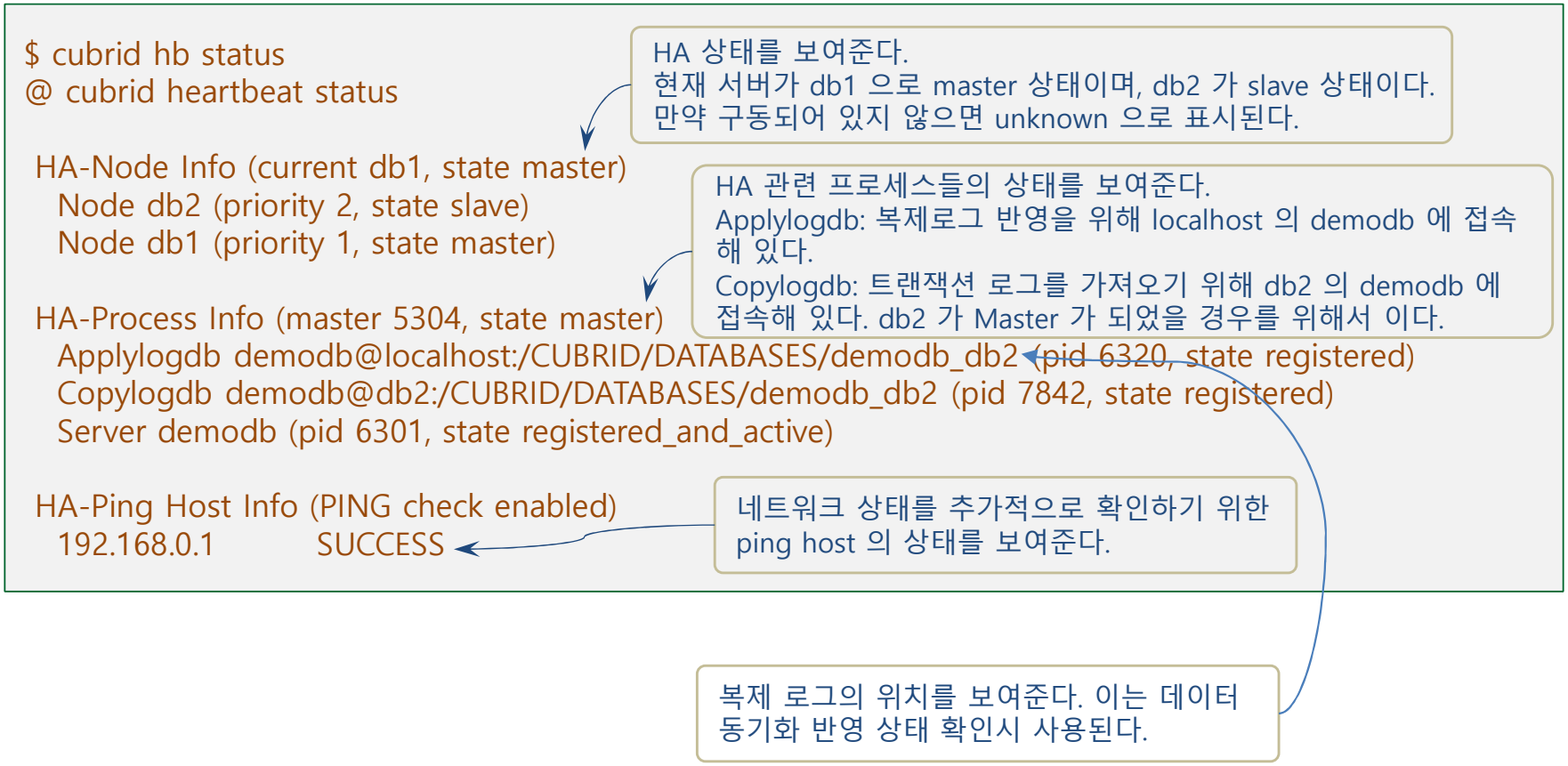
```
# heartbeat 을 같이 구동하기 위해 설정을 수정한다.
$ vi $CUBRID/conf/cubrid.conf
service=broker.manager,heartbeat
```

- 종료
  - 위 설정(cubrid.conf 의 service 부분) 과 상관없이 CUBRID 서비스를 종료하면 HA 까지 같이 종료된다.
  - 필요시 HA 만 종료할 수 있다.

```
# CUBRID 관련 서비스를 모두 종료한다.
$ vi $CUBRID/conf/cubrid.conf
service=broker.manager,heartbeat
```

HA 상태 확인

- HA 서비스 상태 확인
  - 관련 프로세스들의 구동 상태를 확인한다.
  - 서비스 상태 확인시 (cubrid service status) 에서도 동일한 정보를 볼 수 있다.



HA 데이터 동기화 상태 확인

- Master 와 Slave 간 데이터 동기화 상태 확인
  - Slave 서버에서 수행하며, 복제 로그를 통해 동기화 상태를 확인한다.
  - 복제 로그의 기본 위치는 다음과 같은 형태이다.
    - $CUBRID_DATABASES/<데이터베이스 이름>_<Master 서버 이름>

```
# demodb 의 복제 로그 위치를 확인한다. 대부분 기본 위치를 이용하고 있어 확인하지 않아도 된다.
$ cubrid paramdump demodb | grep ha_copy_log_base
ha_copy_log_base=""
```

HA 데이터 동기화 상태 확인

```
# demodb 의 데이터 동기화 상태를 5초 간격으로 확인한다.
# Master 는 db1, 복제 로그는 기본 위치로 $CUBRID_DATABASES/demodb_db1 이다.
$ cubrid applyinfo -a -L $CUBRID_DATABASES/demodb_db1 -r db1 -i 5 demodb

 *** 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                     : demodb
DB creation time            : 04:29:00.000 PM 15/08/2019 (1352014140)
EOF LSA                     : 27722 | 10088
Append LSA                  : 27722 | 10088
HA server state             : active
 ***  Active Info. ***
DB name                     : demodb
DB creation time            : 04:29:00.000 PM 15/08/2019 (1352014140)
EOF LSA                     : 27726 | 2512
Append LSA                  : 27726 | 2512
HA server state             : active

 *** Delay in Copying Active Log ***
Delayed log page count      : 0
Estimated Delay             : 0 second(s)
 *** Delay in Applying Copied Log ***
Delayed log page count      : 0
Estimated Delay             : 0 second(s)
```

Slave 데이터베이스로 데이터 반영 상태
Fail count: 복제 로그 반영 실패 회수

Slave 로 복제 로그 복사 상태

Master 서버의 트랜잭션 로그 기록 상태

Slave 로 트랜잭션 로그 복제 지연 상태
지연된 페이지수와 예상 완료시간

Slave 로 데이터 반영 지연 상태
지연된 페이지수와 예상 완료시간

CUBRID

실습

1. 앞의 HA 구성 전 점검 부분을 따라 점검한다.
    1. hostname 과 IP address 설정이 올바른지 확인하고, 누락되어 있으면 추가한다.

```
# Master/Slave 동일하게 확인 및 적용
[cubrid@db1 new_db]$ hostname; hostname -I
db1
192.168.2.10
[cubrid@db1 new_db]$ ping -4 db1
PING db1 (192.168.2.10) 56(84) bytes of data.
64 bytes from db1 (192.168.2.10): icmp_seq=1 ttl=64 time=0.279 ms
[cubrid@db1 new_db]$ more /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

[root@db1 ~]# cat >> /etc/hosts
192.168.2.10    db1
192.168.2.11    db2
^D
[root@db1 ~]# more /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.2.10    db1
192.168.2.11    db2

[root@db1 ~]# ping -4 db2
PING db2 (192.168.2.11) 56(84) bytes of data.
64 bytes from db2 (192.168.2.11): icmp_seq=1 ttl=64 time=0.516 ms
```

실습

1. …
   2. 방화벽 설정을 확인하고, port 를 허용하도록 추가한다.

```
# Master/Slave 동일하게 확인 및 적용
[root@db1 ~]# systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2022-05-24 09:23:40 KST; 1 day 11h ago

[root@db1 ~]# firewall-cmd --permanent --zone=public --add-port=1523/tcp
success
[root@db1 ~]# firewall-cmd --reload
success
[root@db1 ~]# firewall-cmd --list-ports
1523/tcp
[root@db1 ~]#
```

실습

1. ...
    3. Primary key 부분을 검사해본다.
        1. 실습 과정에서 추가된 일부 테이블에 Primary key 가 없으므로, 해당 테이블은 삭제한다. dba 계정으로 로그인.

```
[cubrid@db1 new_db]$ csql -u dba demodb -S
        CUBRID SQL Interpreter
Type `;help' for help messages.
csql> select owner_name, class_name
csql> from  db_class
csql> where class_type = 'CLASS' and is_system_class = 'NO'
csql> and class_name not in
csql>   ( select class_name from db_index  where is_primary_key ='YES'  )
csql> order by owner_name, class_name
csql> ;x

=== <Result of SELECT Command in Line 1> ===
  owner_name            class_name
============================================
  'DBA'                 'ai_tbl'
  'DBA'                 'ai_tbl2'
  'DBA'                 'employee'
  'DBA'                 'tbl'
  'PUBLIC'              'code'
5 rows selected. (0.032323 sec) Committed.
1 command(s) successfully processed.
csql> drop table ai_tbl, ai_tbl2, employee, tbl, code;
Execute OK. (0.020702 sec) Committed.
1 command(s) successfully processed.
csql>
```

실습

1. ...
    4. LOB 부분을 검사해본다.

```
csql> select class_name, attr_name, data_type
csql> from  db_attribute
csql> where data_type like '%LOB'
csql> order by class_name, attr_name
csql> ;x

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

There are no results.
0 row selected. (0.043253 sec) Committed.

1 command(s) successfully processed.
csql> ;ex
```

실습 - 계속

2. db2 에 CUBRID 를 db1 과 동일한 방법으로 설치(/home/cubrid/CUBRID)하고, HA 구성 진행
   1. HA 실습을 위해 db2 의 demodb 는 삭제한다. 실습중 demodb 를 위한 디렉토리는 만들지 않아도 된다.
   2. 앞부분의 "싱글로 사용중인 데이터베이스의 HA 구성" 부분을 따라하면 된다.

3. HA 구성이 완료되었으면, Master 에 테이블을 생성하고 데이터를 입력한다.

```
[cubrid@db1 ~]$ csql -u dba demodb@localhost

        CUBRID SQL Interpreter

Type `;help' for help messages.

csql> create table ha_tbl (id int primary key);
Execute OK. (0.066194 sec) Committed.

1 command(s) successfully processed.
csql> insert into ha_tbl values(10);
1 row affected. (0.006213 sec) Committed.

1 command(s) successfully processed.
```

실습 - 계속

4. Slave 에서 생성된 테이블을 검색해본다.

```
[cubrid@db2 ~]$ csql -u dba demodb@localhost
        CUBRID SQL Interpreter
Type `;help' for help messages.
csql> select * from ha_tbl;

=== <Result of SELECT Command in Line 1> ===
            id
==============
            10
1 row selected. (0.009260 sec) Committed.
1 command(s) successfully processed.
csql>
```

5. Master 에서 테이블을 삭제한 후, Slave 에서 테이블을 검색한다.

```
csql> drop table ha_tbl;
Execute OK. (0.017223 sec) Committed.

1 command(s) successfully processed.
csql>
```

```
csql> select * from ha_tbl;

In line 2, column 1,

ERROR: before ' ; '
Unknown class "ha_tbl".

0 command(s) successfully processed.
csql>
```

실습 - 계속

5. Slave 에서 HA 상태 및 데이터 동기화 상태를 확인한다.

```
[cubrid@db2 ~]$ cubrid applyinfo -a -L $CUBRID_DATABASES/demodb_db1 -r db1 demodb
 *** Applied Info. ***
Insert count                : 1
Update count                : 0
Delete count                : 0
Schema count                : 2
Commit count                : 29
Fail count                  : 0
 *** Copied Active Info. ***
DB name                     : demodb
DB creation time            : 01:39:05.000 PM 05/23/2022 (1653280745)
EOF LSA                     : 12 | 2072
Append LSA                  : 12 | 2072
HA server state             : active
HA promotion time           : 09:36:42.000 AM 05/24/2022 (1653352602)
DB restore time             : 05:53:46.000 PM 05/23/2022 (1653296026)
Mark will be deleted        : false
 ***  Active Info. ***
DB name                     : demodb
DB creation time            : 01:39:05.000 PM 05/23/2022 (1653280745)
EOF LSA                     : 12 | 2072
Append LSA                  : 12 | 2072
HA server state             : active
HA promotion time           : 09:36:42.000 AM 05/24/2022 (1653352602)
DB restore time             : 05:53:46.000 PM 05/23/2022 (1653296026)
Mark will be deleted        : false
 *** Delay in Copying Active Log ***
Delayed log page count      : 0
Estimated Delay             : - second(s)
 *** Delay in Applying Copied Log ***
Delayed log page count      : 0
Estimated Delay             : - second(s)
[cubrid@db2 ~]$
```

16. 응용 연결 설정

JDBC

- 사용환경
  - JDK1.5 이상
- JDBC driver
  - $CUBRID/jdbc/cubrid_jdbc.jar
- 연결설정

```
import java.sql.*;
import cubrid.jdbc.driver.*;

// HA 구성이 아닌 경우
String url_c = "jdbc:cubrid:192.168.2.10:33000:demodb:::?charset=utf8";
// HA 구성일 경우, althost 를 이용하여 Master 장애시 연결할 Slave 정보를 설정한다.
String url_ha = "jdbc:cubrid:192.168.2.10:33000:demodb:::?althosts=192.168.2.111:33000&charset=utf8";

String db_user = "db_user";
String db_passwd = "passWd";

Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
Connection conn = DriverManager.getConnection(url, db_user, db_passwd);
...
```

ODBC

- 설치
  - Windows 용 CUBRID 설치시 같이 설치된다.
  - 개별 설치시 ftp.cubrid.org 에서 내려받아 사용한다.
  - Unicode(UTF8), ANSI 문자셋 용 설치 파일 별도 제공단다.

- 연결설정

```
// ANSI 문자셋
sConn = "driver={CUBRID Driver};server=localhost;port=33000;uid=db_user;pwd=passWd;db_name=demodb;"

// UNICODE ( UTF8 ) 문자셋
sConn = "driver={CUBRID Driver Unicode};server=localhost;CHARSET=utf-8;
port=33000;uid=db_user;pwd=passWd;db_name=demodb;"

// HA 접속
sConn = "driver={CUBRID Driver Unicode};server=localhost;CHARSET=utf-8;
port=33000;uid=db_user;pwd=passWd;db_name=demodb;althosts=192.168.0.218:33000;loginTimeout=600;"
```

ADO.NET

- 설치
  - ftp.cubrid.org 에서 내려 받아 사용한다.

- 연결설정

```
using CUBRID.Data.CUBRIDClient;

namespace CUBRID.ADO.NET.DataProvider.MyClass
{
    …
    sb = new CUBRIDConnectionStringBuilder();
    sb.User = “db_user" ;
    sb.Password = “passWd”;
    sb.Database = "demodb";
    sb.Port = "33000";
    sb.Server = "localhost";
    using (CUBRIDConnection conn = new CUBRIDConnection(sb.GetConnectionString()))
    {
    conn.Open();
    }
    …
}
```

PHP

- 설치
  - PECL, apt_get 을 이용하여 설치
  - ftp.cubrid.org 에서 소스를 받아 빌드

- 연결설정

```
$conn = cubrid_connect("localhost", 33000, "demodb", "db_user", "passWd");

$con = cubrid_connect_with_url("cci:CUBRID:localhost:33000:demodb:db_user:passWd");

$con = cubrid_connect_with_url("cci:CUBRID:localhost:33000:demodb:db_user:passWd:?autocommit=flase");

$con = cubrid_connect_with_url(
                "cci:CUBRID:localhost:33000:demodb:db_user:passWd:?althosts=192.168.0.101:33000&
login_timeout=5000&disconnect_on_query_timeout=true", "db_user", "passWd");
```

CCI C-API

- 빌드 환경
  - header file : $CUBRID/include/cas_cci.h
  - library file : $CUBRID/lib/libcascci.so

```
T_CCI_ERROR cci_error;
con = cci_connect_with_url("cci:CUBRID:localhost:33000:demodb:db_user:passWd:", "db_user", "passWd");
if (con < 0) {
                printf("ERR: cci connect failure₩n");
                return -1;
}
...
req = cci_prepare (con, query, 0, &cci_error);
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);
}
do {
                error = cci_fetch (req, &cci_error);
                ...
} while ((res = cci_cursor(req, 1, CCI_CURSOR_CURRENT, &error)) == 0);
error = cci_close_req_handle (req);
cci_end_tran(con, CCI_TRAN_COMMIT, &error);
cci_disconnect(con);
```

17. 성능 개선

질의 스캔 방식

- Sequential scan
  - 테이블의 모든 페이지를 순차적으로 읽는 방식(full scan)
- Temp(List File) scan
  - Derived Table(inline view) 처리
  - 임시 결과 (distinct, union, order by, group by ...)
- Index scan
  - Index page read + 데이터파일(heap file) access
  - 인덱스를 탈 수 있는 조건이 있어야 함
  - Where 절, select list에 사용되는 컬럼이 모두 index로 설정되어 있는 경우 index page만 read
- 질의 수행 비용
  - 예상되는 CPU 비용 + IO 비용 및 cardinality
- 조인 순서
  - nl-join의 특별한 형태로 인덱스가 제대로 걸리는 지 확인이 필요
  - outer 테이블과 inner 테이블이 맞게 배치되었는지 확인 필요
    - 선행(outer) 테이블은 레코드 수가 적은 것, 범위 줄이기(일의 양을 결정)
    - 후행(inner) 테이블은 레코드 수가 많고, 인덱스가 있는 것

CUBRID TRACE 정보

- CUBRID TRACE 출력방법
  - csql>;trace on
- Query Plan
  - SORT : distinct & order by & group by
  - JOIN : NESTED LOOPS & MERGE JOIN (inner join & outer join)
- Trace Statistics
  - SELECT 정보 (time : 전체 질의시간, **fetch** : 페이지 접근한 회수, ioread : 디스크 접근회수)
  - TALBE FULL SCAN (table : 테이블 정보, heap time : 소요시간, **fetch** : 페이지 접근한 회수, ioread : 디스크 접근회수, readrows : 스캔 시 읽은 행의 개수, rows : 결과 행의 개수)
  - **참조: fetch 정보는 개수가 아닌 페이지를 접근 회수임, 같은 페이지를 다시 fetch하더라도 회수가 증가함**
  - 인덱스 Trace : (index : 인덱스 정보, btree time : 소요시간, **fetch** : 페이지 접근한 회수, ioread : 디스크 접근회수, readkeys : 읽은 키의 개수, filteredkeys : 키 필터가 적용된 키의 개수, rows : 스캔한 행 개수)
  - 테이블 Trace : (lookup time : 인덱스 스캔 후 데이터에 접근하는데 소요된 시간 rows : 데이터 필터까지 적용한 이후의 행 개수)
- GROUP-BY & ORDER-BY
  - GROUPBY (time : 소요시간, sort: true :정렬이 적용여부, page : 정렬에 사용된 임시 페이지 개수, ioread : 디스크 접근 소요시간, rows : group by에 대한 결과 행의 개수)
  - ORDERBY (time: 소용시간, sort : 정렬 적용여부, page: 정렬에 사용된 임시 페이지 객수, ioread: 디스크 접근 소요시간)

CUBRID TRACE 출력 - 1

- **CSQL FULL SCAN TRACE 정보**

```
• SQL
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year
WHERE o.host_year > 1950
group by h.host_year
order by h.host_year

• TRACE
Query Plan:
  SORT (distinct)
    SORT (group by)
      NESTED LOOPS (inner join)
        TABLE SCAN (h)
        INDEX SCAN (o.pk_olympic_host_year) (key range: h.host_year=o.host_year, key filter:
(o.host_year> ?:0 ))

Trace Statistics:
  SELECT (time: 1, fetch: 443, ioread: 0)
    SCAN (table: history), (heap time: 0, fetch: 148, ioread: 0, readrows: 147, rows: 147)
      SCAN (index: olympic.pk_olympic_host_year), (btree time: 0, fetch: 294, ioread: 0, readkeys:
147, filteredkeys: 0, rows: 147) (lookup time: 0, rows: 147)
    GROUPBY (time: 0, hash: true, sort: true, page: 0, ioread: 0, rows: 8)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```


CUBRID

CUBRID TRACE 출력 - 2

- **CSQL INDEX SCAN TRACE 정보**

• SQL (인덱스 추가: CREATE INDEX idx01 ON history (host_year, athlete);)

```
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year
WHERE o.host_year > 1950
group by h.host_year
order by h.host_year
```

• TRACE

```
Query Plan:
  SORT (distinct)
    SORT (group by)
      NESTED LOOPS (inner join)
        INDEX SCAN (o.pk_olympic_host_year) (key range: (o.host_year> ?:0 ))
        INDEX SCAN (h.idx01) (key range: h.host_year=o.host_year, covered: true)

Trace Statistics:
  SELECT (time: 0, fetch: 45, ioread: 0)
    SCAN (index: olympic.pk_olympic_host_year), (btree time: 0, fetch: 15, ioread: 0, readkeys: 14,
filteredkeys: 0, rows: 14) (lookup time: 0, rows: 14)
      SCAN (index: history.idx01), (btree time: 0, fetch: 28, ioread: 0, readkeys: 131, filteredkeys: 0,
rows: 147, covered: true)
    GROUPBY (time: 0, hash: true, sort: true, page: 0, ioread: 0, rows: 8)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```


CUBRID

CUBRID PLAN 정보

- CUBRID 플랜 출력방법
  - csql>;set level 257 : simple plan
  - csql>;set level 513 : detail plan

- 조인 방식: 질의 계획에서 출력되는 조인 방식
  - 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 인덱스 사용한 조인

- 조인 종류: 위에서 (inner join) 부분으로, 질의 계획에서 출력되는 조인 종류
  - inner join, left outer join, right outer join, cross join

- 조인 테이블의 종류: 중첩 루프 조인에서 루프의 어느 쪽에 위치하는가를 기준
  - outer 테이블: 조인할 때 가장 처음에 읽을 기준 테이블
  - inner 테이블: 조인할 때 나중에 읽을 대상 테이블

- 스캔 방식: 질의가 인덱스를 사용하는지 여부를 판단
  - sscan: 순차 스캔(sequential scan). full table scan을 의미
  - iscan: 인덱스 스캔(index scan). 인덱스를 사용하여 스캔 의미

- COST: CPU, IO 등 주로 리소스의 사용과 관련하여 비용을 내부적으로 산정

CUBRID PLAN 출력 - 1

- **CSQL FULL SCAN PLAN 정보**

```
•    SQL
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year
WHERE o.host_year > 1950
group by h.host_year
order by h.host_year

•    PLAN
temp(distinct)
    subplan: temp(group by)
                subplan: idx-join (inner join)
                            outer: sscan
                                        class: h node[0]
                                        cost:  1 card 147
                            inner: iscan
                                        class: o node[1]
                                        index: pk_olympic_host_year term[0]
                                        filtr: term[1]
                                        cost:  1 card 2
                            cost:  3 card 15
                sort:  1 asc
                cost:  9 card 15
    sort:  1 asc
    cost:  15 card 15
```


CUBRID

CUBRID PLAN 출력 - 2

- **CSQL INDEX SCAN PLAN 정보**

```
•    SQL (인덱스 추가: CREATE INDEX idx01 ON history (host_year, athlete);)
SELECT /*+ RECOMPILE */ DISTINCT h.host_year, o.host_nation
FROM history h INNER JOIN olympic o
ON h.host_year = o.host_year
WHERE o.host_year > 1950
group by h.host_year
order by h.host_year

•    PLAN
temp(distinct)
    subplan: temp(group by)
                subplan: idx-join (inner join)
                            outer: iscan
                                        class: o node[1]
                                        index: pk_olympic_host_year term[1]
                                        cost:  1 card 2
                            inner: iscan
                                        class: h node[0]
                                        index: idx01 term[0] (covers)
                                        cost:  1 card 147
                            cost:  2 card 15
                sort:  1 asc
                cost:  8 card 15
    sort:  1 asc
    cost:  14 card 15
```


CUBRID

질의 힌트

- 인덱스 지정
  - Where절 다음에 using index 구문을 통해 index 힌트 지정
  - 지정한 인덱스와 full scan 중 평가
  - 비용 기반의 인덱스 선택
    - 비용 강제 0 설정 → (+) 지정
  - 인덱스 명 중복시 <테이블명>.<인덱스명>
  - join 시 필요한 인덱스 모두 지정

```
create index idx1 on athlete(name)
create index idx1 on record(athlete_code)

select * from athlete where name = 'Lee Eun-Chul' using index none
select * from athlete where name = 'Lee Eun-Chul' using index idx1
select * from athlete where name = 'Lee Eun-Chul' using index idx1(+)
select * from athlete using index idx1(+) → index 사용을 위한 조건이 없어 idx1 index 를 사용하지 못함. (변경: where name > ''using index idx1(+);)

select host_year, medal from athlete a, record b where a.name = 'Lee Eun-Chul' and a.code = b.athlete_code using index a.idx1, b.idx1
```

- 인덱스 사용 팁
  - 인덱스 키의 사이즈는 되도록 작게 설계하여야 성능에 유리하다.
  - 분포도가 좋은 컬럼(좁은 범위), 기본 키, 조인의 연결 고리가 되는 컬럼에 대해 인덱스를 구성한다.
  - 업데이트가 빈번하지 않은 컬럼으로 인덱스를 구성한다.

인덱스 정렬

- 인덱스를 이용한 정렬 성능 개선
  - 검색 조건과 정렬 대상을 이용한 인덱스 생성

```
csql> SELECT * FROM nation WHERE continent = 'Asia'
csql> ORDER BY name;

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

  code                  name                  continent             capital
============================================================================
  'AFG'                 'Afghanistan'         'Asia'                'Kabul'
  'BRN'                 'Bahrain'             'Asia'                'Manama'
  'BAN'                 'Bangladesh'          'Asia'                'Dhaka'
  'BHU'                 'Bhutan'              'Asia'                'Thimphu'
  'BRU'                 'Brunei Darussalam'   'Asia'                'Bandar Seri Begawan'
  'BIR'                 'Burma'               'Asia'                ''
  'CAM'                 'Cambodia'            'Asia'                'Phnom Penh'
Trace Statistics:
  SELECT (time: 0, fetch: 1, ioread: 0)
    SCAN (table: nation), (heap time: 0, fetch: 1, ioread: 0, readrows: 220, rows: 47)
    ORDERBY (time: 0, sort: true, page: 0, ioread: 0)
```

```
csql> CREATE INDEX idx1 ON nation(continent, name);
Execute OK. (0.009801 sec) Committed.

1 command(s) successfully processed.
csql> SELECT * FROM nation WHERE continent = 'Asia'
csql> ORDER BY name;

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

  code                  name                  continent             capital
============================================================================
  'AFG'                 'Afghanistan'         'Asia'                'Kabul'
  'BRN'                 'Bahrain'             'Asia'                'Manama'
  'BAN'                 'Bangladesh'          'Asia'                'Dhaka'
  'BHU'                 'Bhutan'              'Asia'                'Thimphu'
  'BRU'                 'Brunei Darussalam'   'Asia'                'Bandar Seri Begawan'
  'BIR'                 'Burma'               'Asia'                ''
  'CAM'                 'Cambodia'            'Asia'                'Phnom Penh'
Trace Statistics:
  SELECT (time: 0, fetch: 50, ioread: 0)
    SCAN (index: nation.idx1), (btree time: 0, fetch: 49, ioread: 0, readkeys: 47,
 filteredkeys: 0, rows: 47) (lookup time: 0, rows: 47)
```

인덱스 힌트

- 인덱스 힌트
  - WHERE절 힌트
    - USING INDEX : 질의 수행 시 사용할 인덱스 지정
  - FROM절 힌트
    - USE INDEX : 지정한 인덱스들만 선택 시 고려
    - FORCE INDEX : 해당 인덱스 선택이 우선
    - IGNORE INDEX : 지정한 인덱스들은 선택에서 제외

```
csql> SELECT * FROM participant
csql> WHERE host_year >= 2000 AND nation_code LIKE 'C%';

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

    host_year  nation_code                              gold        silver        bronze
============================================================================================
         2000  'CAF'                                       0             0             0
         2000  'CAM'                                       0             0             0
         2000  'CAN'                                       3             3             8
         2000  'CAY'                                       0             0             0
         2000  'CGO'                                       0             0             0
Trace Statistics:
  SELECT (time: 0, fetch: 43, ioread: 0)
    SCAN (index: participant.pk_participant_host_year_nation_code), (btree time: 0
, fetch: 42, ioread: 0, readkeys: 40, filteredkeys: 0, rows: 40) (lookup time: 0,
rows: 40)
```


CUBRID

인덱스 힌트

```
csql> SELECT * FROM participant
csql> WHERE host_year >= 2000 AND nation_code LIKE 'C%'
csql> USING INDEX fk_participant_nation_code;

=== <Result of SELECT Command in Line 3> ===

    host_year  nation_code                    gold        silver        bronze
============================================================================
          2004  'CAF'                           0             0             0
          2000  'CAF'                           0             0             0
          2004  'CAM'                           0             0             0
          2000  'CAM'                           0             0             0
          2004  'CAN'                           3             6             3
Trace Statistics:
  SELECT (time: 0, fetch: 88, ioread: 0)
    SCAN (index: participant.fk_participant_nation_code), (btree time: 0, fetch: 87,
 ioread: 0, readkeys: 20, filteredkeys: 0, rows: 86) (lookup time: 0, rows: 40)
```

* 인덱스 생성 후 플랜
CREATE UNIQUE INDEX athlete_idx1 ON athlete(code, nation_code);
CREATE INDEX athlete_idx2 ON athlete(gender, nation_code);

```
csql> SELECT /*+ RECOMPILE */ *
csql> FROM athlete USE INDEX (athlete_idx1, athlete_idx2)
csql> WHERE gender = 'M' AND nation_code = 'USA';

=== <Result of SELECT Command in Line 4> ===

         code  name                      gender                 nation_code           event
============================================================================================
        10956  'Ewing Patrick'           'M'                    'USA'                 'Basketball'
        10953  'Everett Adam'            'M'                    'USA'                 'Baseball'
        10928  'Ervin Anthony'           'M'                    'USA'                 'Swimming'
        10861  'Drunnond Jonathan'       'M'                    'USA'                 'Athletics'
        10859  'Driscoll Jean'           'M'                    'USA'                 'Athletics'
Trace Statistics:
  SELECT (time: 1, fetch: 442, ioread: 0)
    SCAN (index: athlete.athlete_idx2), (btree time: 0, fetch: 441, ioread: 0, rea
dkeys: 1, filteredkeys: 0, rows: 438) (lookup time: 0, rows: 438)
```

조인 힌트

- 조인 힌트
  - 조인 방법 지정
    - USE_NL : Nested Loop 조인 지정
    - USE_IDX : Index 조인 지정
    - USE_MERGE : Sort Merge 조인 지정
  - 조인 순서 지정
    - ORDERED : **FROM** 절에 명시된 테이블의 순서대로 조인
  - 조인 한정
    - 힌트 다음 () 안에 테이블 명. 예) ORDERED(tbl_a)

```
SELECT /*+ USE_IDX */ h.host_year, h.unit,
h.score, o.host_nation
FROM history h, olympic o
WHERE h.host_year >= 2000 AND
o.host_year >= 2000
AND h.host_year = o.host_year;
```

```
SELECT /*+ ORDERED */ h.host_year, h.unit,
h.score, o.host_nation
FROM olympic o, history h
WHERE h.host_year >= 2000 AND
o.host_year >= 2000
AND h.host_year = o.host_year;
```

```
idx-join (inner join)
    outer: sscan
                class: h node[0]
                sargs: term[2]
                cost:  1 card 15
    inner: iscan
                class: o node[1]
                index: pk_olympic_host_year term[0]
                filtr: term[1]
                cost:  1 card 2
    cost:  2 card 1
```

```
nl-join (inner join)
    edge:  term[0]
    outer: iscan
                class: o node[0]
                index: pk_olympic_host_year term[1]
                cost:  1 card 2
    inner: sscan
                class: h node[1]
                sargs: term[0] AND term[2]
                cost:  1 card 15
    cost:  3 card 1
```

질의 튜닝 참고 – 인덱스 활용

- 컬럼에 인덱스가 있음에도 활용하지 못하는 SQL을 인덱스를 활용 할 수 있도록 튜닝
- 인덱스가 있어도 활용하지 못하는 경우 : 인덱스 컬럼에 함수 사용, NOT 연산, 다중컬럼 인덱스의 경우 첫번째 컬럼이 아닌 컬럼으로 조회, 조회조건 없이 조회
- 발생빈도 : 상

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT code, NAME FROM athlete WHERE NAME IS NOT null;	SELECT code, NAME FROM athlete WHERE NAME >= '';	• NOT 연산은 인덱스 사용이 불가함으로 >= 연산으로 변경한다. • CUBRID는 empty string(공백 & '')과 NULL이 다르므로 NULL과 공백 값을 모두 제외할 경우 > 연산으로 처리한다.
CREATE INDEX i_game_game_date ON game(game_date); SELECT athlete_code FROM game WHERE TO_CHAR(game_date,'yyyy')='2000';	SELECT athlete_code FROM game WHERE game_date BETWEEN to_char('20000101','yyyymmdd') and to_char('20001231','yyyymmdd');	• TO_CHAR 함수를 사용하여 인덱스 사용 불가. 2000년 1월부터 12월까지 조회하여도 동일한 결과. TO_CHAR 함수를 제거하여 인덱스 활용하도록 변경한다.
CREATE INDEX idx01_athlete ON athlete(nation_code, name); SELECT * FROM athlete WHERE NAME LIKE 'Fernandez%'	CREATE INDEX idx01_athlete ON athlete(name, nation_code); SELECT * FROM athlete WHERE NAME LIKE 'Fernandez%'	• Index가 nation_code,name로 있고, 조회조건을 name으로 검색 할 경우에 인덱스 사용이 불가함으로, 인덱스를 name, nation_code로 생성한다.
SELECT game_date FROM game ORDER BY game_date DESC LIMIT 1	CREATE INDEX i_game_game_date_medal ON game (game_date); SELECT game_date FROM game where game_date >=('1000-01-01') ORDER BY game_date DESC LIMIT 1	• 특정 단일 컬럼에 대한 MIN, MAX 구하는 경우 해당 컬럼에 인덱스 생성한다. • 인덱스를 스캔할 수 있도록 where절에 전체 검색할 수 있는 조건을 추가한다. • order by와 limit 사용해 1건을 출력한다.

select * from game where stadium_code='30115' and nation_code='KOR'
create index i_game_1 on game(stadium_code);
create index i_game_2 on game(nation_code);
	select * from game where stadium_code='30115' and nation_code='KOR'
	create index i_game_1 on game(stadium_code, nation_code);
		• 하나의 테이블에 하나의 인덱스만 선택. 단일 인덱스 여러 개 보다는 multi-column index 생성한다.

질의 튜닝 참고 – Skip order by 처리

- 인덱스를 생성하면, 생성 한 인덱스는 지정한 컬럼으로 정렬되어 있고, 각각의 인덱스 정보는 레코드의 위치정보가 있다. 정렬하고자 하는 컬럼이 인덱스로 생성 되어 있을 경우, 인덱스는 정렬되어 있음으로 별도의 정렬 처리를 생략함으로 성능이 향상 된다.
- 발생빈도 : 상

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT * FROM athlete WHERE NAME > 'D' ORDER by NAME limit 10;	CREATE INDEX i_athlete_name ON athlete(NAME); SELECT * FROM athlete WHERE NAME > 'D' ORDER by NAME limit 10;	• 전체 데이터에서 'D'보다 큰 데이터를 찾아서 name으로 정렬하고 10건을 처리하는 것을, name 인덱스를 활용하여 D 보다 큰 데이터를 10건만 찾아서 처리하도록 튜닝 할 수 있다.(skip ORDER BY)
SELECT tab.* FROM (SELECT a.code, a.nation_code, b.name name1, a.[name], a.AREA, a.seats, a.address FROM stadium a, nation b WHERE a.nation_code = b.code AND a.seats >10000 ORDER BY a.seats, a.name) tab WHERE ROWNUM > 0 AND ROWNUM <= 10	CREATE INDEX idx01_stadium ON stadium (seats, name); SELECT a.code, a.nation_code, b.name name1, a.[name], a.AREA, a.seats, a.address FROM stadium a, nation b WHERE a.nation_code = b.code AND a.seats >10000 ORDER BY a.seats, a.name LIMIT 0, 10	• 불필요한 subquery 제거 후 limit 함수를 사용해 페이징 처리한다. • 조회 결과 전체 데이터를 정렬한 후 최종 10건의 데이터를 조회된다. • 추가한 인덱스를 이용하면 SKIP ORDER BY 처리로 최종 결과 10건만 스캔하여 성능을 개선할 수 있다.(skip ORDER BY)
select rn, * from (select rownum rn, * from (select * from tbl where ... order by ...) where rownum <= 20) where rn >= 11	select orderby_num() rn, * from tbl where ... order by ... limit 11, 20	• 정렬한 순서로 넘버 값을 출력하는 경우는 ORDERBY_NUM() 함수를 사용한다. • MySQL 형태의 limit 를 사용하면 질의 성능이 향상되며 SQL이 단순해 진다.

CUBRID

질의 튜닝 참고 – 불 필요한 처리내용 제거(1)

- 불필요한 처리내용을 제거 함으로서 쿼리 성능을 높인다.
- count 데이터만을 조회 할 경우의 불필요한 처리내용으로는 order by절, 불필요한 outer조인, 불필요한 컬럼, 인라인뷰 등을 제거한다. 이때, 테이블의 데이터를 참조하는 것이 아닌 인덱스 만으로 처리가 가능 함 (이를 Covering index라고 함)
- 발생빈도 : 상

튜닝 전	튜닝 후	튜닝 내용 설명
create index athlete_idx2 on athlete (gender, nation_code); Select * from athlete where gender='M' and nation_code='ESP';	Select gender, nation_code from athlete where gender='M' and nation_code='ESP';	• Covering index(index page read만으로 질의 처리) 가능하도록 Select list 및 where 절에 불필요한 컬럼을 명시하지 않는다.
SELECT COUNT(*) FROM (SELECT a.host_year, a.game_date FROM game a LEFT JOIN event b ON a.event_code = b.code LEFT JOIN athlete c ON a.athlete_code = c.code LEFT JOIN olympic d ON a.host_year = d.host_year WHERE A.nation_code = 'USA' ORDER BY a.host_year, a.game_date DESC) result	SELECT COUNT(*) FROM (SELECT a.host_year, a.game_date FROM game a WHERE A.nation_code = 'USA') result	• 최종 결과가 count를 리턴하는 것으로 불필요한 left outer join, order by 제거하면 성능이 매우 향상된다.
SELECT COUNT(tab.code) totcnt FROM (SELECT a.code, a.nation_code, b.[name] AS nation_name, a.[name], a.AREA, a.seats, a.address FROM stadium a, nation b WHERE a.nation_code = b.code AND a.seats >10000 ORDER BY a.name) tab	SELECT COUNT(a.code) totcnt FROM stadium a, nation b WHERE a.nation_code = b.code AND a.seats >10000	• 불필요 한 인라인뷰, 컬럼, ORDER BY를 제거하면 성능이 향상된다.

CUBRID

질의 튜닝 참고 – 부분범위 처리

- 스칼라쿼리의 수행 회수는 추출 row수 만큼 수행 됨. 이때, 추출 row수를 부분범위 처리하여 스칼라쿼리의 수행 횟수를 줄여서 성능 향상
- 발생빈도 : 상

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT * FROM (SELECT ROWNUM num, A.* FROM (SELECT t.host_year, t.event_code, t.athlete_code, (select name from athlete b where b.code = t.athlete_code) athlete_name, t.stadium_code, t.nation_code, t.medal, t.game_date FROM game t where 1 = 1 AND t.game_date BETWEEN TO_DATE('1988-01-01', 'YYYY-MM-DD') AND TO_DATE('2004-12-31', 'YYYY-MM-DD') ORDER BY t.game_date DESC) A) B WHERE B.NUM BETWEEN 0 AND 10;	SELECT t.host_year, t.event_code, t.athlete_code, (select name from athlete b where b.code = t.athlete_code) athlete_name, t.stadium_code, t.nation_code, t.medal, t.game_date FROM (select * from game a where 1 = 1 AND a.game_date BETWEEN TO_DATE('1988-01-01', 'YYYY-MM-DD') AND TO_DATE('2004-12-31', 'YYYY-MM-DD') ORDER BY a.game_date DESC LIMIT 0, 10) t	• SELECT 절의 스칼라쿼리는 row 개수만큼 수행됨. 최종 10건의 데이터가 반환되는 쿼리인데 전체 데이터에 대해 subquery를 적용한 후 최종 10건을 반환하고 있음. 이를 먼저 최종 10건의 데이터를 추출한 후 subquery를 적용하면 subquery 수행 횟수가 10번으로 줄어든다. • 참조: "FOR ORDERBY_NUM() BETWEEN 0 AND 10" 구문은 "LIMIT 0,10" 변경해도 동일한 결과를 출력된다.

질의 튜닝 참고 – Join 컬럼은 인덱스 생성

- NL-Join은 기준이 되는 Driving 테이블과 Join 대상이 되는 Inner 테이블이 있다. 이때, Inner 테이블의 JOIN-KEY가 되는 컬럼은 인덱스가 생성 되어 있으면 성능이 향상 된다.
- 발생빈도 : 상

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT 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;	CREATE INDEX idx01_history on history (host_year); SELECT 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;	• olympic 테이블은 선행 즉, Driving(outer) 테이블이고, history 테이블이 후행 Inner 테이블이다. 후행 Inner 테이블의 host_year 컬럼이 데이터를 filter하는 Join key이다. • NL-JOIN 시 대부분 데이터를 filter할 수 있는 Join key에 인덱스가 없어서 성능이 안 좋은 경우가 많은데 인덱스를 생성 후 전/후 처리 비용을 확인해 인덱스 추가 여부를 결정한다.

질의 튜닝 참고 – 스칼라 쿼리 ←→ Outer join

- SELECT절에서 사용한 스칼라 서브쿼리를 Outer join으로 변경하여 성능을 개선
- FROM절의 Outer join을 스칼라 서브쿼리로 변경하여 성능을 개선
- 발생빈도 : 중

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT t.host_year, t.event_code, t.athlete_code, (select name from athlete b where b.code = t.athlete_code) athlete_name, t.stadium_code, t.nation_code, t.medal, t.game_date FROM game t where 1 = 1 AND t.game_date BETWEEN TO_DATE('1988-01-01', 'YYYY-MM-DD' AND TO_DATE('2004-12-31', 'YYYY-MM-DD')	SELECT t.host_year, t.event_code, t.athlete_code, b.name, t.stadium_code, t.nation_code, t.medal, t.game_date FROM game t LEFT OUTER JOIN athlete b ON b.code = t.athlete_code where 1 = 1 AND t.game_date BETWEEN TO_DATE('1988-01-01', 'YYYY-MM-DD') AND TO_DATE('2004-12-31', 'YYYY-MM-DD')	• SELECT 절의 스칼라 쿼리는 반환되는 레코드 개수만큼 수행되므로 결과 건수가 많은 경우는 호출 횟수를 줄이도록 스칼라 쿼리보다 outer join을 사용하여 성능을 개선할 수 있다.
SELECT a.code, a.name, a.continent, a.capital, b.cnt FROM nation a LEFT OUTER JOIN (SELECT nation_code, COUNT(*) cnt FROM game GROUP BY nation_code) b ON (a.code = b.nation_code) ORDER BY a.name	SELECT a.code, a.name, a.continent, a.capital, (SELECT COUNT(*) FROM game b where a.code = b.nation_code) cnt FROM nation a ORDER BY a.name	• 결과 건수가 적은 경우에 FROM절의 outer join을 스칼라 쿼리로 변경하여 성능을 개선할 수 있다.

CUBRID

질의 튜닝 참고 – 다중컬럼 조건은 인라인뷰로 처리

- 오라클에 다중컬럼 조건처리는 CUBRID에서 { } 괄호를 사용하며 OUTER 테이블 조인건수에 따라 인라인뷰로 성능이 개선될 수 있음
- 발생빈도 : 중

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT a.code, a.nation_code FROM athlete a WHERE (a.code, a.nation_code) IN (SELECT b.athlete_code, b.nation_code FROM game b WHERE b.athlete_code=11280 AND b.nation_code='USA') → CUBRID 질의 오류 발생 함	1, { } 괄호 처리 SELECT a.code, a.nation_code FROM athlete a WHERE (a.code, a.nation_code) IN (SELECT { b.athlete_code, b.nation_code } FROM game b WHERE b.athlete_code=11280 AND b.nation_code='USA') 2, 인라인뷰 처리 SELECT a.code, a.nation_code FROM athlete a, (SELECT DISTINCT b.athlete_code, b.nation_code FROM game b WHERE b.athlete_code=11280 AND b.nation_code='USA') c WHERE a.code=c.athlete_code AND a.nation_code=c.nation_code	• 다중컬럼 조건은 { } 괄호로 사용이 가능하다. • 대상 질의는 FROM 절에 있는 결과를 서비쿼리와 비교하므로 인라인뷰 질의 형식으로 기준 테이블에 조건을 먼저 필터 하는 방식이 대부분 성능에 유리하다.

질의 튜닝 참고 – 뷰테이블 내용을 쿼리문에 반영

- View 테이블에 대한 조회조건은 인덱스 스캔을 못함으로, 뷰 테이블의 내용을 수행하는 SQL문에 직접 반영하여 성능 개선
- 발생빈도 : 중

튜닝 전	튜닝 후	튜닝 내용 설명

CREATE VIEW VIEW_TBL_LIST AS
SELECT
  DISTINCT
  ROWNUM AS rn ,
  a.host_year,
  a.event_code,
  b.code,
  b.name as event_name,
  a.athlete_code,
  c.name as althlete_name,
  a.stadium_code,
  a.nation_code,
  a.medal,
  a.game_date
FROM game a, event b, athlete c
WHERE a.event_code = b.code
  AND a.athlete_code = c.code ;
 	
SELECT
  *
FROM VIEW_TBL_LIST
WHERE code > 20000 ;
	SELECT
	  DISTINCT
	  ROWNUM AS rn,
	  a.host_year,
	  a.event_code,
	  b.code,
	  b.name as event_name,
	  a.athlete_code,
	  c.name as althlete_name,
	  a.stadium_code,
	  a.nation_code,
	  a.medal,
	  a.game_date
	FROM game a, event b, athlete c
	WHERE a.event_code = b.code
	  AND a.athlete_code = c.code
	  AND b.code > 20000 ;
		• View 테이블의 내용을 SQL문에 직접 반영하여 temp로 처리되는 단계를 제거하여 성능 개선

질의 튜닝 참고 – 다중 row를 한row로 처리

- 여러 row로 관리 되는 데이터를 한 개 row로 표시하고자 할 때 connect by 절 대신 group_concat 그룹 함수를 사용하여 성능 개선
- 발생빈도 : 중

튜닝 전	튜닝 후	튜닝 내용 설명
SELECT continent, SUBSTR(MAX(SYS_CONNECT_BY_PATH(name, ',')) , 2) name_list FROM (SELECT name , continent, row_number() over (PARTITION BY continent) rnum FROM nation WHERE continent = 'Oceania') START WITH rnum=1 CONNECT BY PRIOR rnum = rnum-1 AND PRIOR continent= continent GROUP BY continent	SELECT continent, group_concat(name) name_list FROM nation WHERE continent = 'Oceania' GROUP BY continent	• 다중 row의 데이타를 CONNECT BY절을 사용하여 한 row로 처리하는 것을 group_concat 그룹함수를 사용하여 성능 개선

CUBRID

질의 튜닝 참고 – 갱신이 자주 일어나는 컬럼 분리

- 컬럼 개수가 많거나, 레코드 사이즈가 큰 테이블에 대해 특정 컬럼 값만 갱신이 자주 일어나는 경우, 레코드 전체에 대한 갱신이 발생되지 않도록 테이블을 분리 한다.
- Update 수행될 경우 트랜잭션 로그는 변경되는 컬럼의 값만 저장되는 것이 아니라 해당 레코드 전체의 이미지가 저장됨. 트랜잭션 로그 생성의 부하로 인해 성능 저하 발생.
- 테이블의 컬럼 개수가 많고, 이런 테이블에서 update 가 자주 발생하는 컬럼에 개수가 적다면 테이블을 분리
- 발생빈도 : 중

튜닝 전	튜닝 후	튜닝 내용 설명
CREATE TABLE board(id int primary key, title varchar(250), contents string, ... read_count int, ...); UPDATE board SET read_count=read_count+1 WHERE id=1;	CREATE TABLE board(id int primary key, title varchar(250), ... read_count int, ...); CREATE TABLE board_contents(id int primary key, contents string); UPDATE board SET read_count=read_count+1 WHERE id=1;	• 게시판을 관리하는 테이블에서 내용과 조회수를 함께 관리하여 트랜잭션 로그 생성 시 성능저하 발생 • 게시판 내용을 별도 테이블(board_contents)로 분리하여 조회수가 변경 시에 로그 생성 부하를 최소화 함

CUBRID

질의 작성 시 참고

- 선행(outer) 테이블의 데이터 개수 만큼 후행(inner) 테이블에 접근하므로 선행 케이블에서 추출된 데이터 건수가 일의 양을 결정
  - NL-JOIN 조건은 테이블 건수를 확인하고 FROM절 건수가 적은 테이블 순서로 배치해 SELECT 절에 /*+ ORDERED */ 힌트를 사용해서 FROM절에 명시된 테이블의 순서대로 조인하는 실행계획으로 유도해 본다.
  - NL-JOIN 조건에 필터링한 데이터량이 큰 경우 SELECT 절의 /*+ USE_MERGE */ 힌트를 이용해 Merge join 실행계획으로 유도해 본다.
  - 선행 테이블의 데이터 건수를 줄이기 위해 조건절을 추가하거나 INLINE VIEW를 사용해 LIMIT 또는 BETWEEN 함수로 선행 테이블의 데이터 건수를 최소화할 수 있는 방법을 고려해 본다.
  - WHERE절 SUB QUERY를 IN 또는 Exists로 변경이 가능하면 선행(outer) 테이블의 건수를 최소화할 수 있어 질의성능이 향상될 가능성이 높다.
- Where절에서 자주 사용되는 컬럼에는 인덱스 추가를 고려 한다.
  - SKIP_ODERBY는 "정렬 비용 없으므로" 인덱스 사용 조건을 확인해 정렬 컬럼에 결합 인덱스 추가 고려
  - ORDER BY, GROUP BY 컬럼은 결합 인덱스 맨 뒤에 배치될 수 있는지 추가 시 고려
  - ORDER BY와 같이 LIMIT 함수를 사용하면 LIMIT 개수만큼만 Key Range를 하고 인덱스 스캔을 중단해 성능에 유리함
  - 결합 인덱스는 첫번째 컬럼이 WHERE절에 반드시 배치되어야 사용 가능
  - NL-JOIN 시 후행(inner) 테이블의 JOIN KEY가 되는 컬럼에 인덱스가 없다면 추가를 고려
- 인덱스를 많이 생성하는 것은 INSERT/UPDATE/DELETE 성능 저하의 원인이 될 수 있다.
- 불필요한 컬럼을 SELECT에서 검색하면 covering index를 활용할 수 없다.
- 가능하면 동등(=) 비교를 사용하고 인덱스를 사용할 수 없는 부정형(<>, !=) 비교를 지향 한다.
- 데이터 분포도가 좋은 컬럼과 분포도는 나쁘지만 특정 값이 중복도가 낮으면서 자주 호출될 경우 인덱스 생성 고려 한다.
- 필요에 의해서 데이터 분포도가 매우 나쁜 컬럼에 인덱스가 생성되어 있고 insert/delete 자주 발생할 경우 데이터 분포도가 가장 좋은 컬럼과 결합 인덱스를 구성해 사용한다.
- JOIN KEY는 서로 데이터 타입이 동일 않으면 자동 형 변환으로 인한 성능 저하 발생될 수 있다.
- 인덱스 scan이 Full Table scan보다 항상 빠르지는 않다.
- 질의 수행계획 확인을 통한 성능 개선(최적의 인덱스와 최적의 조인 방식 선택)

수고많으셨습니다. 감사합니다.

CUBRID™