CUBRID 운영자 교육

CUBRID 9.2 이상

Date: 2015-04-21

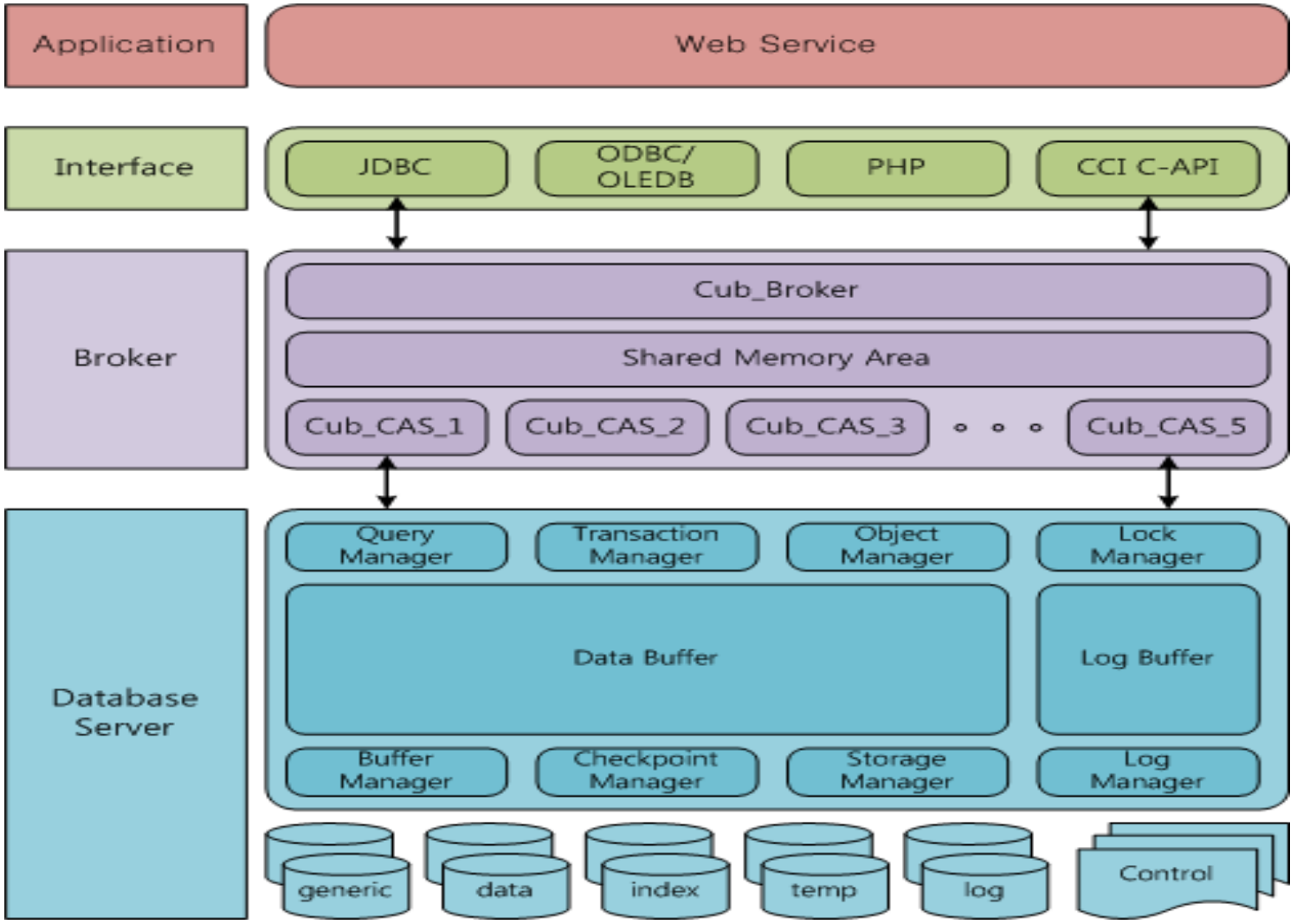
㈜큐브리드 기술본부

목차

CUBRID

1. CUBRID 시스템 구조

CUBRID 시스템 구조도



CUBRID

CUBRID 시스템 구조

- Database Server
  - 데이터를 저장 및 관리하는 기능을 수행 한다.
  - Multi thread기반 client/server 방식으로 동작 한다.
  - 사용자가 입력한 질의를 처리하고 DB 내의 객체 관리 한다.
  - 잠금(LOCK)과 로깅(Logging) 기법을 이용해 다수 사용자의 동시 트랜잭션을 지원 한다.
  - 운영에 필요한 백업 및 복구 기능을 지원 한다.

- BROKER
  - CUBRID 전용 미들웨어로 외부 응용 프로그램 간의 통신을 중계하는 역할을 한다.
  - 브로커는 커넥션 풀 링, 모니터링, 로그 추적 및 분석 기능 제공 한다.

- CUBRID Manager
  - GUI 환경에서 Database Server와 Broker를 관리할 수 있는 관리자 도구이다.
  - CUBRID 설치 시 기본으로 CUBRID Manager Server 모듈는 같이 설치된다.
  - CUBRID Manager 관리자 도구를 다운로드 받아 사용한다.

CUBRID 프로세스

- CUBRID 설치 후 서비스를 위한 구동이 완료되면 프로세스를 확인 할 수 있다.
  - “ps -ef | grep cub_*” 명령어로 CUBRID 전체 구동된 process 확인한다.
  - CUBRID Service가 구동되면 Master, DB server, Broker, CAS, Manager 프로세스가 확인된다.

- Master Process
  - 프로세스 검색 명령어에 의해 “cub_master” 로 확인된다.
  - 큐브리드 서비스를 구동하면 하나의 cub_master 프로세스가 구동된다.
  - Client와 DB Server프로세스 사이의 연결을 담당하는 프로세스이다.

- Database Server Process
  - 프로세스 검색 명령어에 의해 “cub_server <db_name>”의 형태로 확인된다.
  - 데이터베이스가 구동되어 있을 때 검색되는 프로세스이다.
  - 구동되는 데이터베이스의 수 만큼 프로세스가 확인된다.
  - 데이터베이스 파일 및 로그 파일 등에 직접 접근하여 사용자 요청을 처리하는 프로세스이다.

- 실행 모드
  - CUBRID의 프로그램들은 종류에 따라 클라이언트/서버 모드(client/server mode)와 독립 모드(standalone mode)의 두 가지 실행 모드가 있다.
  - 클라이언트/서버 모드는 프로세스가 구동되어 있는 상태이며 독립 모드는 프로세스가 구동되지 않은 상태이다.
  - 클라이언트/서버 모드는 해당 프로그램이 클라이언트 프로세스로서 동작하여 서버 프로세스에 접속하는 방식이다.
  - 독립 모드는 CSQL등의 하나의 프로세스가 독립 모드(standalone mode) 로 데이터베이스에 접속되어 있으면 추가 접근을 할 수 없다.

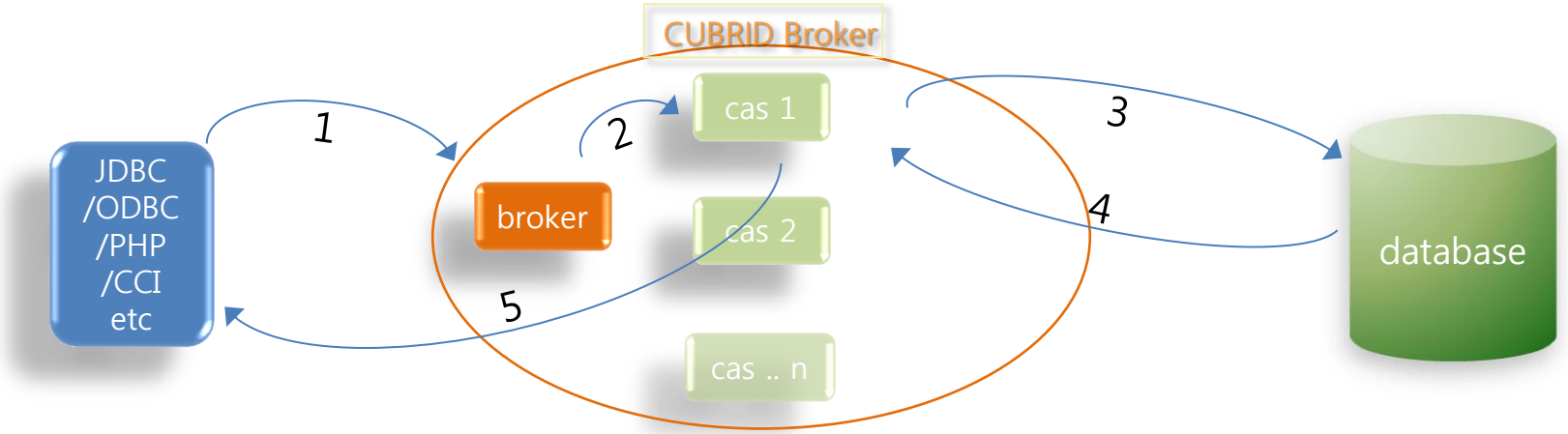
CUBRID 프로세스 - 계속

- BROKER Process
  - 프로세스 검색 명령어에 의해 “cub_broker”로 확인되고 브로커가 구동되어 있을 때 검색되는 프로세스이다.
  - cubrid_broker.conf에 등록되어 Service 상태가 ON인 브로커의 개수 만큼 확인된다.
  - 응용 Client(JDBC, ODBC, PHP 등)과 cub_cas 프로세스 사이의 연결을 중계하는 기능을 수행한다.
  - 브로커는 cub_cas 프로세스의 상태를 관리하고 모니터링한다.

- CAS Process
  - 프로세스 검색 명령어에 의해 “cub_cas”로 확인되고 브로커가 구동 되어 있어야만 확인할 수 있는 프로세스로 cub_broker 프로세스 없이 존재 한다면 비정상 프로세스다.
  - DB에 연결하고자 하는 모든 종류의 응용 Client가 사용하는 공용 응용 서버의 역할을 한다.
  - 환경 설정에 따라 cub_broker 프로세스에 의해 동적으로 조정된다.
  - Query 분석이나 최적화, 실행 계획 생성 등의 작업이 수행된다.

- Manager Process
  - cub_auto Process
    - 프로세스 검색 명령어에 의해 “cub_auto”로 확인되고 8001번 기본포트를 사용한다.
    - CUBRID Manager Client 사용자의 인증 처리를 및 주기적인 자동화 작업, 진단 정보 수집하는 프로세스이다..
  - cub_js Process
    - 프로세스 검색 명령어에 의해 “cub_js”로 확인되고 8002번 기본포트를 사용한다.
    - CUBRID Manager Client에서 전송되는 사용자 요구를 수행하는 프로세스이다.
  - 기타 프로세스
    - CUBRID 버전에 따라 “cub_cmserver”, “cub_cmhttpd”, “cub_cmserver_ext” 구동되는 프로세스의 차이가 있다.
    - CUBRID 9.3.2 버전부터 CUBRID Client(GUI-관리도구) 프로세서가 “cub_cmserver” 하나로 통합 되였다.

CUBRID 처리 프로세서



1. 응용(JAVA,PHP)에서 작업(질의) 처리를 broker로 요청 한다.
2. Broker 는 관리하고 있는 CAS프로세스에게 작업을 할당 한다
   - 질의의 작업처리는 CAS가 수행한다.(프로세스 단위로 작업을 수행 함)
   - 응용(JAVA,PHP)은 CAS에 직접 요청하는 것이 아니고, CAS를 관리하는 broker 에게 작업 요청을 한다.
3. CAS는 데이터베이스에 연결하고 질의처리를 요청한다.
   1. 응용(JAVA,PHP)의 컨넥션 URL 정보로 데이터베이스 지정하며, 지정한 데이터베이스로 CAS가 연결한다.
   2. CAS는 데이터베이스 작업 후 연결유지해 동일한 데이터베이스로 작업요청이 들어오면 기존연결 재 사용 한다.
   3. CAS는 연결유지 상태에서 다른 데이터베이스의 작업요청이 들어오면 현재 연결을 끊고 새로이 연결 한다.
   4. 데이터베이스 별로 broker를 할당해서 사용하는 것이 성능과 유지관리에 유리하다.
4. 데이터베이스는 CAS에서 요청을 처리하고 결과를 해당 CAS에게 전달한다.
5. 데이터베이스의 작업결과(데이터)를 응용에 직접 전달 한다.

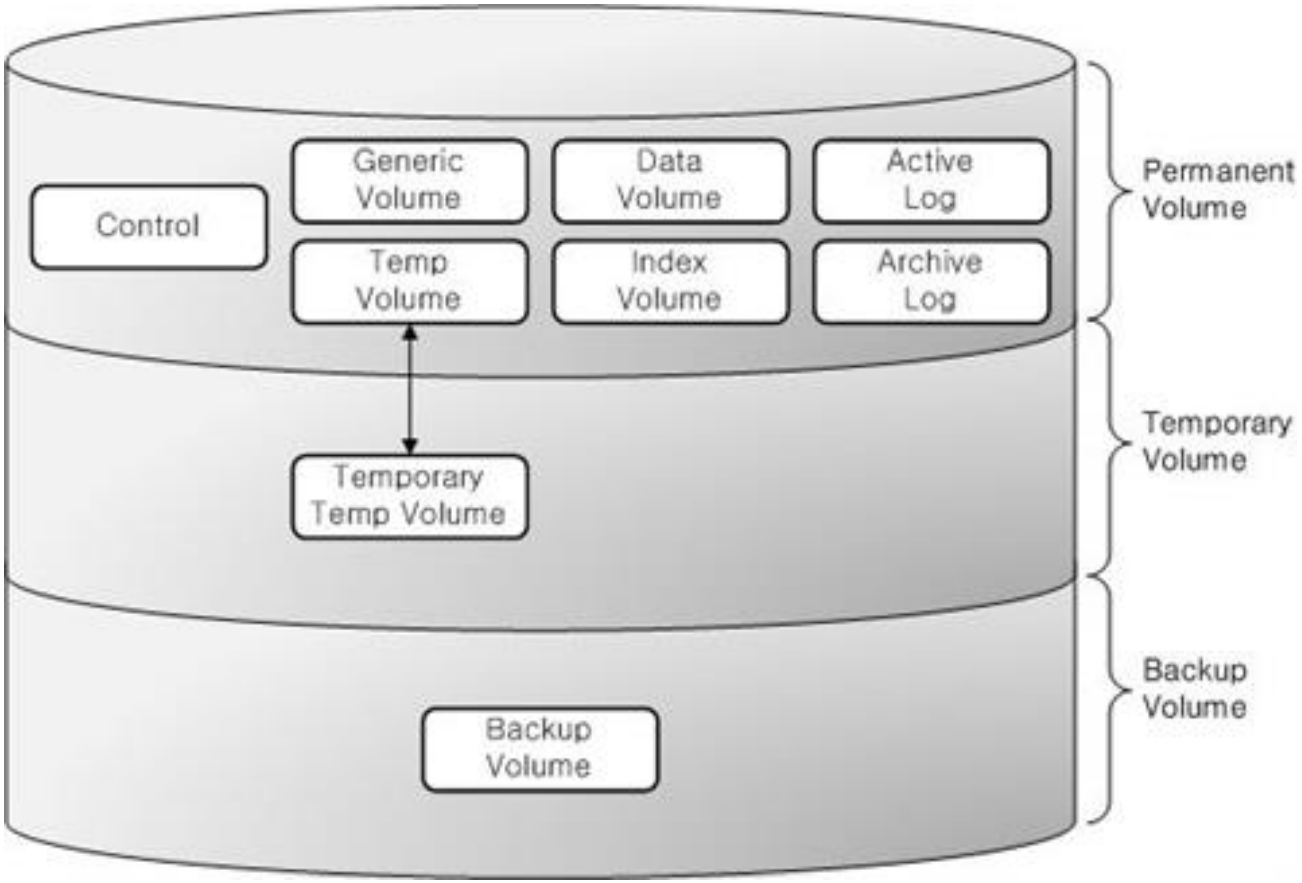
※ CAS와 트랜잭션

트랜잭션 처리 중에는 하나의 CAS가 하나의 응용에 종속되며 트랜잭션 처리 종료 전까지는 다른 요청 받지 않는다.

- ✓ CAS 프로세서 구동 수 만큼에 동시 트랜잭션이 처리된다.
- ✓ 응용 프로그램에서 select 도 commit/rollback을 처리해야 한다.

CUBRID

데이터베이스 구성 파일



CUBRID

데이터베이스 구성 파일 - 계속

- 영구적 볼륨(Permanent Volumes)
  - 범용 볼륨(Generic volume)
    - DB생성 시 초기 볼륨이며 data, index볼륨으로 사용 가능하다.
  - 데이터 볼륨(Data volume)
    - 스키마, 레코드, 멀티미디어 데이터와 같은 응용 프로그램의 데이터를 저장하기 위한 공간이다.
  - 인덱스 볼륨(Index volume)
    - 신속한 질의 처리와 무결성 제약 조건(integrity constraints)의 인덱스 정보를 유지하는 공간이다.
  - 임시 볼륨(Temp volume)
    - 질의 처리(join, group by, order by, sub-query, create index…)나 정렬(sorting)을 위해 사용되는 공간이다.
  - 활성 로그(Active Log)
    - Active Log는 데이터베이스에 가장 최근에 발생된 committed/aborted/active 트랜잭션의 상태를 기록한다.
    - 매체 고장이 발생할 경우 데이터베이스 복구를 위해 사용되고 각 데이터베이스는 하나의 active 로그 볼륨을 가진다.
  - 보관 로그(Archive Log)
    - 활성 로그(Active log) 공간이 모두 사용되면 보관 로그(Archive log)로 복사한다.
    - 서비스 처리량에 따라 일일 여러 개의 보관 로그(Archive log)가 생성된다.

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

CUBRID

데이터베이스 구성 파일 - 계속

- 일시적 볼륨(Temporary Volumes)
  - 일시적으로 사용되는 볼륨파일로써 사용자가 영구 볼륨으로 지정한 공간을 초과하여 데이터가 축적되는 경우에만 생성된다.
  - 서비스 운영 시 DB의 Temporary temp volume을 생성하는 비용은 상당히 클 수 있어 운영상황을 고려해 temp 볼륨 공간을 생성하는 것이 성능상 유리하다.
  - Temporary temp volume은 DB가 재 시작 할 때 삭제된다.

- 백업 볼륨(Backup Volumes)
  - Backup 작업에 의해 생성되는 파일이다.
  - Backup 볼륨은 백업할 시점 데이터베이스의 스냅샷이다.
  - $CUBRID/cubrid.conf의 backup_volume_max_size_bytes로 백업 분할 크기는 조정할 수 있다.

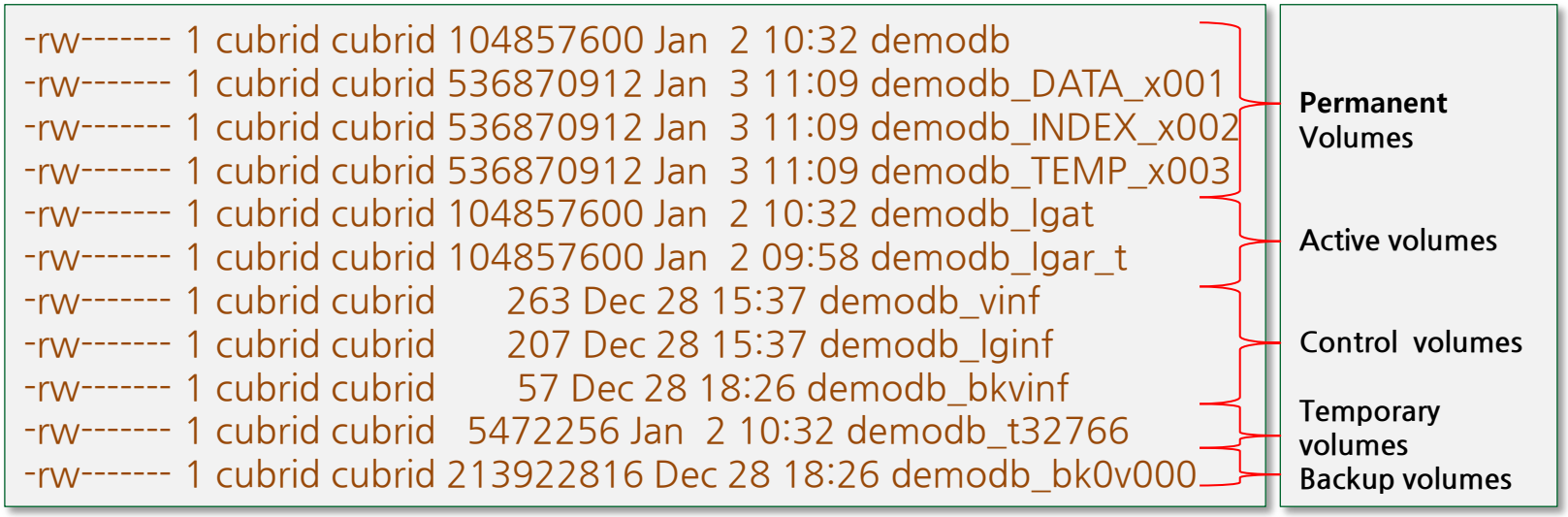
- CUBRID 환경 파일
  - databases.txt 파일
    - 데이터베이스의 위치(디렉터리) 정보를 기록하고 있는 파일이다.
  - cubrid.conf 파일
    - CUBRID server 파라미터의 설정값이 저장되어 있는 파일이다.
  - cubrid_broker.conf 파일
    - CUBRID broker 파라미터의 설정값이 저장되어 있는 파일이다.
  - .cubrid.sh
    - CUBRID 제품설치 시 자동으로 생성되며 CUBRID PATH 정보 파일이다.
    - Windows 제품은 .cubrid.sh 파일이 생성되지 않고 환경변수로 자동 설정된다.

CUBRID

데이터베이스 구성 파일 - 계속

- databases.txt파일
  - 데이터베이스 생성 시 databases.txt 파일에 생성된 데이터베이스에 관한 정보가 기록된다.
  - 생성된 데이터베이스의 정보는 DB이름, 경로, 서버호스트명 정보가 기록된다.
  - $CUBRID_DATABASES 환경 변수가 명시되어 있는 경로에 저장된다.
  - databases.txt 관리 사항
    - DB서버의 호스트명이 변경될 경우 databases.txt 파일에 기록된 호스트명도 변경한다.
    - 데이터베이스 생성 및 삭제 시 databases.txt 파일이 수정되므로 CUBRID 사용자(계정)은 읽기/쓰기의 파일 권한이 설정되어 있었야 한다.
    - 데이터베이스 삭제가 필요할 때는 deletedb 명령어를 이용해 삭제한다.

- 데이터베이스 볼륨파일 정리



2. CUBRID 설치

LINUX 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.
    - 확인방법 : $> uname -a & rpm -q glibc
    - 결과 : Linux host_name **2.6.18-53.1.14.el5xen** #1 SMP Wed Mar 5 12:08:17 EST x86_64 x86_64 x86_64 GNU/Linux → Kernel 2.6
  - 기본 라이브러리: Linux 버전에 상관없이 기본 라이브러리 확인이 필요하다.
    - 확인방법 : $> rpm -q ncurses & libgcrypt & libstdc++
  - 32/64bit 여부 : Linux bit를 확인하고 설치할 CUBRID 버전과 bit를 선택한다.
    - 확인 방법 : $> uname -a
    - 결과 : Linux host_name 2.6.18-53.1.14.el5xen #1 SMP Wed Mar 5 12:08:17 EST **x86_64 x86_64 x86_64** GNU/Linux → 64bit OS
  - 방화벽 설정
    - Linux 서버 환경에서 CUBRID 사용하기 위해서 다음 포트는 기본적으로 오픈 한다.
    - CUBRID Manager : 8001, 8002
    - 브로커: 30000, 33000 (질의편집기 : 30000, 응용개발 : 33000)

LINUX 설치

- Linux 버전
  - CUBRID 설치 및 관리할 사용계정을 생성해서 제품설치를 권장한다.
  - CUBRID DBMS Server와 Client(CUBRID 명령어, Broker)는 동일 버전에서만 완전한 호환을 지원한다.

```
root#> useradd cubrid

root#> su - cubrid

cubrid$> sh CUBRID-9.3.2.0016-linux.x86_64.sh
Copyright (C) -2014 Search Solution Corporation. All rights reserved.
...
Do you agree to the above license terms? (yes or no) : yes
Do you want to install this software(CUBRID) to the default(/home/cubrid/CUBRID) directory? (yes
or no) [Default: yes] : yes
Install CUBRID to '/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. For installation of CUBRID tools like Query
Browser, Manager and Web Manager, please refer to http://www.cubrid.org/wiki_tools.
Do you want to continue? (yes or no) [Default: yes] : yes

If you want to use CUBRID, run the following commands
  % . /home/cubrid/.cubrid.sh
  % cubrid service start

cubrid$> cubrid_rel
CUBRID 9.3 (9.3.2.0016) (64bit release build for linux_gnu)
```

WINDOWS 설치 전 확인 사항

- Windows에서 설치
  - 공식 지원 윈도우 버전
    - 32bit: Windows XP 32bit, Windows Vista 32bit, Windows 2003 server 32bit
    - 64bit: Windows Vista 64bit, (Windows 7)
  - 호환 CPU: Intel x86, Intel EM64T, AMD64
  - Windows 32bit & 64bit 지원
    - OS bit 확인 방법 : 내컴퓨터 → 속성 → 시스템에서 확인 가능.
  - 설치 유형 선택
    - 전체설치 : CUBRID서버와 명령행 도구 및 인터페이스 드라이버(JDBC, ODBC, OLEDB 등)가 설치된다.
    - 인터페이스 드라이버 설치 : 드라이버(JDBC, ODBC, OLEDB)만 설치된다.
  - 방화벽 설정
    - Windows 서버 환경에서 CUBRID를 사용하기 위해서 다음 포트는 기본적으로 오픈 한다.
    - CUBRID Manager : 8001, 8002
    - 질의편집기 : 30000 ~ 30040
    - 응용개발 : 33000 ~ 33040
- JAVA 설치
  - CUBRID Manager client, CUBRID Query Browser, JAVA Stored Procedure 사용이 필요할 경우 JRE 1.6이상 버전이 설치되어 있어야 한다.

WINDOWS 설치

- Windows 버전
  - CUBRID 설치는 administrator 계정을 권장한다.
  - Default로 C:₩CUBRID 위치에 CUBRID 제품이 설치된다.
  - CUBRID Manager Client 사용을 위해 JRE 1.6 이상 버전을 설치한다.
  - 설치유형 선택 시 전체설치 선택한다.
  - 설치과정

1단계: 설치 디렉터리 지정

2단계: 설치 유형 선택
• 전체 설치 : CUBRID 서버와 명령행 도구 및 인터페이스 드라이버(JDBC, C API)가 모두 설치된다.
• 인터페이스 드라이버 설치 : 인터페이스 드라이버(JDBC, C API)만 설치된다. CUBRID 데이터베이스 서버가 설치된 컴퓨터에 원격 접근하여 개발하는 경우, 이 설치 유형을 선택할 수 있다.

3단계: 샘플 데이터베이스 생성
→ 샘플 데이터베이스를 생성하려면 약 300MB의 디스크 공간이 필요하다.

4단계: 설치 완료

→ 우측 하단 [ ]모양에 CUBRID Service Tray가 나타난다.

CUBRID Engine 구조

CUBRID/	CUBRID 의 홈 디렉터리
bin/	실행 파일이 위치한 디렉터리
conf/	환경 설정 파일이 위치한 디렉터리
databases/	databases.txt 와 sample 데이터베이스가 위치한 디렉터리
demo/(LINUX)	demodb생성 script가 저장된 디렉터리
include/	include file 이 위치한 디렉터리
java/	JAVA SP 관련 파일이 위치한 디렉터리
jdbc/	CUBRID jdbc driver가 위치한 디렉터리
lib/	library 파일이 위치한 디렉터리
locales/	CUBRID 로켈 디렉토리
log/	각종 log 가 저장되는 디렉터리(server, broker)
msg/	CUBRID 에서 사용되는 각종 메시지 파일이 저장된 디렉터리
share/	HA관련 script가 저장된 디렉터리
tmp/	CUBRID 임시 디렉터리
var/	CUBRID 프로세서의 소켓 디렉터리

3. CUBRID GUI 도구

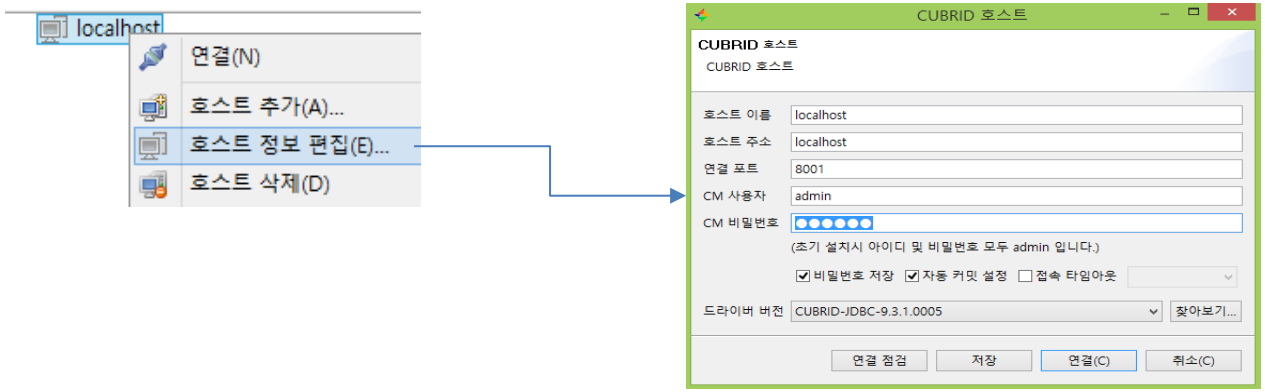
CUBRID Manager 설치와 접속

- CUBRID Manager OS 환경
  - CUBRID Manager는 Java 실행 환경에서만 실행이 가능하기 때문에 우선 JRE를 설치한다.
  - CUBRID Manager는 Linux, Windows, Mac OS와 32bit, 64bit를 모두 지원한다.

- CUBRID Manager 설치
  - Windows: 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.

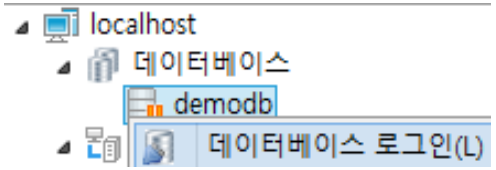
- CUBRID Manager 실행
  - Windows: C:₩CUBRID₩cubridmanager₩cubridmanager.exe 클릭해 실행한다.
  - Linux: $CUBRID/cubridmanager 위치로 이동해 ./cubridmanager를 실행한다.

- CUBRID Manager 접속
  - CUBRID Manager 접속 포트는 8001, 8002번이며 CUBRID Server에서 포트 오픈이 필요하다.
  - 로그인(default 사용자인 admin의 비밀번호는 admin이다. 처음 로그인 후 비밀번호를 변경해야 한다)

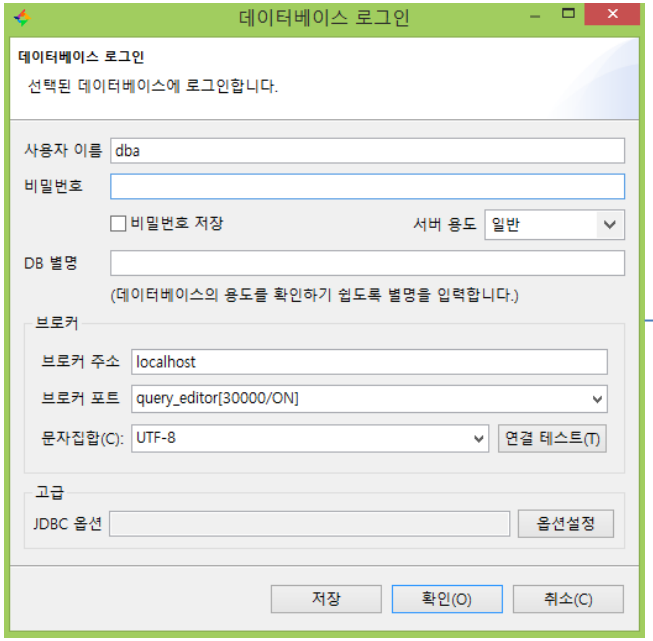


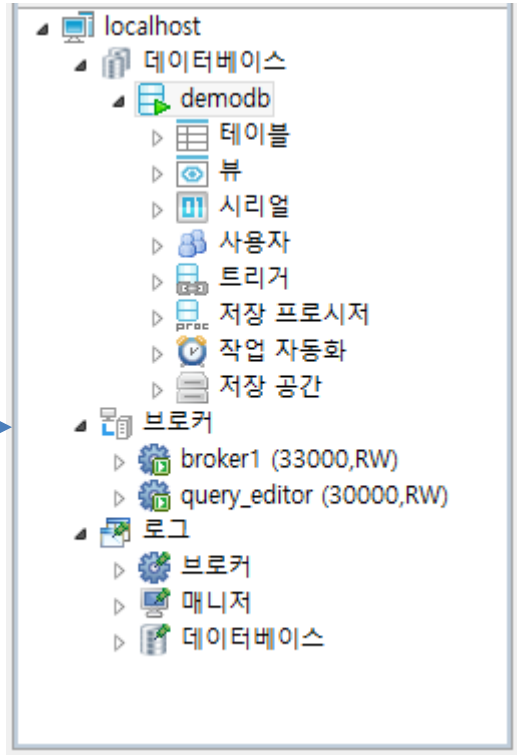
CUBRID Manager 설치와 접속 - 계속

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



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

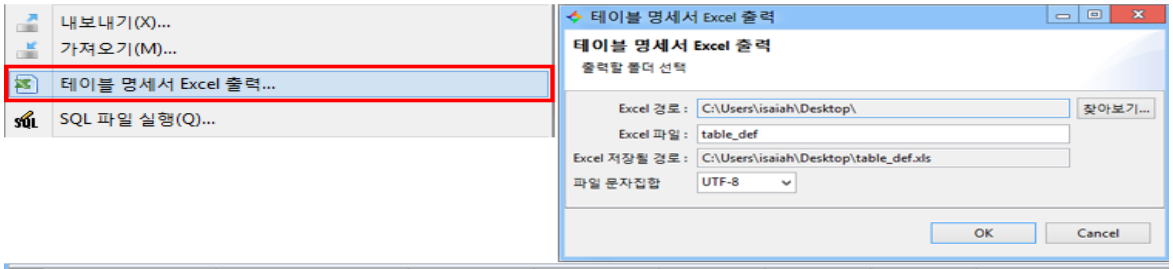




CUBRID Manager 설치와 접속 - 계속

- 데이터베이스 관리
  - CUBRID 매니저에서 데이터베이스를 구동 및 정지 기능을 제공한다, 데이터베이스를 선택한 후, 툴바에서 |시작/정지|을 클릭해 시작과 정지를 수행할 수 있다.
  - 데이터베이스 생성 마법사로 사용할 데이터베이스를 생성할 수 있다.
  - 데이터베이스 백업과 복구, Unload/load(export/import) 기능을 제공한다.
  - 성능과 관리의 필요한 데이터베이스 최적화, 공간정리, 검사 기능을 제공한다.
  - 데이터베이스 이름변경 및 복사 기능을 제공되며 이외 관리에 필요한 다양한 기능일 제공한다.

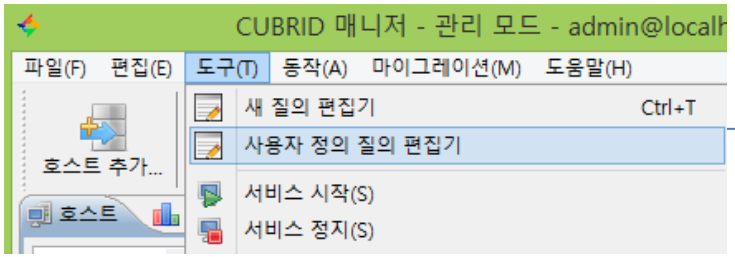
- 테이블 명세서 출력
  - 데이터베이스의 모든 테이블에 대해 테이블 명세서를 Excel로 출력하는 기능을 제공한다, 툴바 매뉴에서 [동작]→[테이블 명세서 Excel 출력] 클릭해 수행한다.

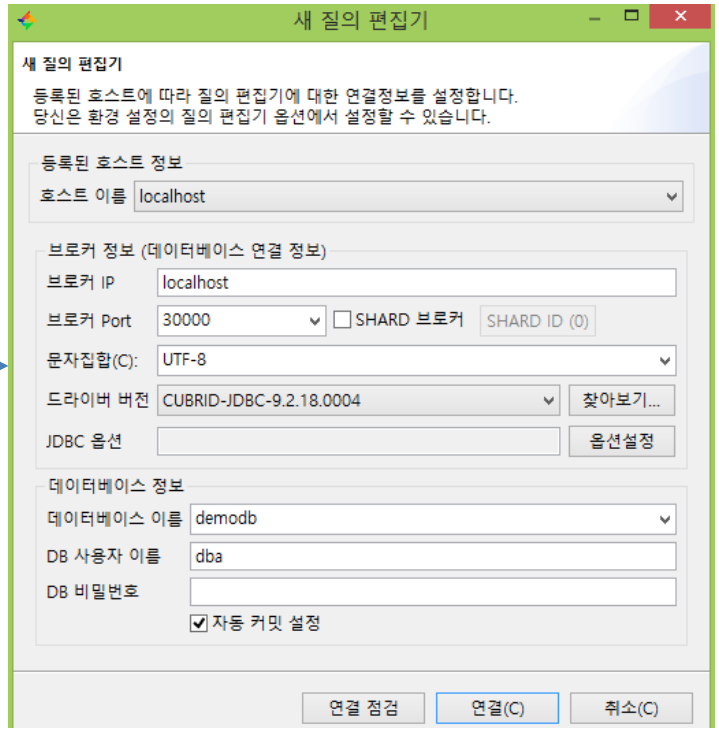


	A	B	C	D	E	F	G	H
1	테이블 명세							
2	시스템 이름		날짜	2012.12.24		작성자		
3	테이블명	athlete						
4	테이블 ID	athlete						
5	컬럼명	컬럼 ID	자료형	크기	NULL	PK	FK	비고
6	code	code	INTEGER		Y			
7	name	name	VARCHAR	40				
8	gender	gender	CHAR	1	Y			
9	nation_code	nation_code	CHAR	3	Y			
10	event	event	VARCHAR	30	Y			
11								
12	인덱스 정의							
13	NO	인덱스명		컬럼 ID		순서	비고	
14								
15	DDL							

CUBRID Manager 질의 편집기

- 질의 편집기
  - CUBRID 매니저 질의 편집기는 모든 DML, DDL, DCL 문의 수행을 지원하는 질의 도구로서, 보다 쉽고 편리하게 질의를 편집하고 실행할 수 있는 기능이다.
  - 질의 편집기를 실행하려면 메뉴에서 |도구| → |새 질의 편집기|를 선택하거나, 툴바에서 |새 질의 편집기|를 클릭한다.





  - 호스트 이름: 좌측 화면에 등록한 호스트
  - 브로커 IP: 브로커 서버의 IP
  - 브로커 Port: 접속 대상 브로커 포트(default: 30000)
  - 문자집합: 데이터베이스 문자셋을 선택
  - 드라이버 버전: CUBRID 엔진버전의 JDBC를 선택
  - 데이터베이스 이름: 접속대상 DB명
  - DB 사용자 이름/비밀번호: DB 계정/비번

  - Manager 매뉴얼: http://www.cubrid.org/wiki_tools/entry/cubrid-manager-reference_kr

4. CUBRID 구동 및 종료

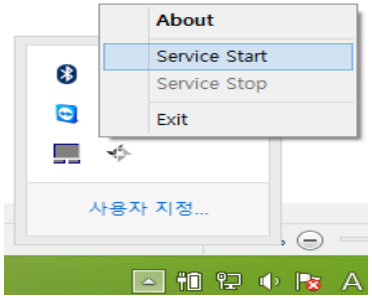
CUBRID 서비스 구동

- 서비스 구동
  - CUBRID 운영에 필요한 기본 프로세스 구동
  - CUBRID 사용자 계정으로 로그인 필요
  - master, broker, manager server 구동
  - database server 는 별도 구동, 또는 설정을 통하여 서비스 구동 시 같이 구동 가능

- 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
```

- Windows
  - service 에 등록되어 부팅 시 구동(default)



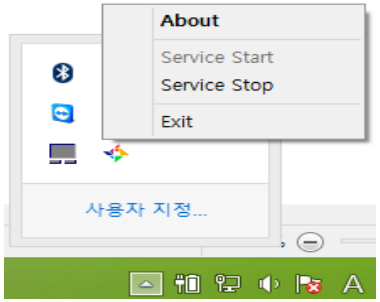
CUBRID 서비스 종료

- 서비스 종료
  - CUBRID 관련 모든 프로세스 종료
  - CUBRID 사용자 계정으로 로그인 필요
  - master, broker, manager server 및 database server 종료

- 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
```

- Windows
  - Exit 를 선택하면, 서비스 종료 후 tray 응용까지 종료됨.



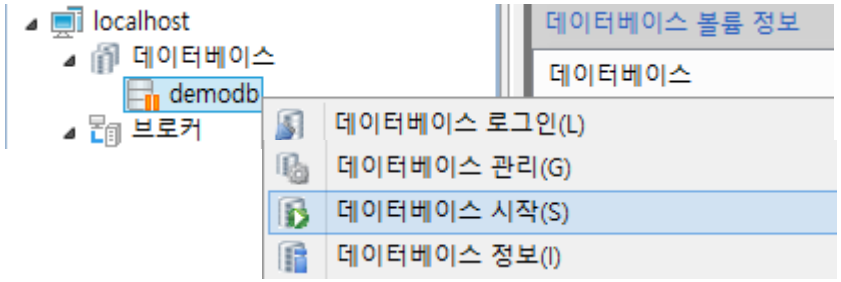
CUBRID Server 구동

- 데이터베이스 구동
  - 사용하는 데이터베이스 별 서버 구동

- 명령어
  - CUBRID 사용자 계정으로 로그인

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

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

CUBRID  R9.2

++ cubrid server start: success
```

- Windows
  - CUBRID Manager Client에서 database dba 계정으로 로그인해야만 구동이 가능.



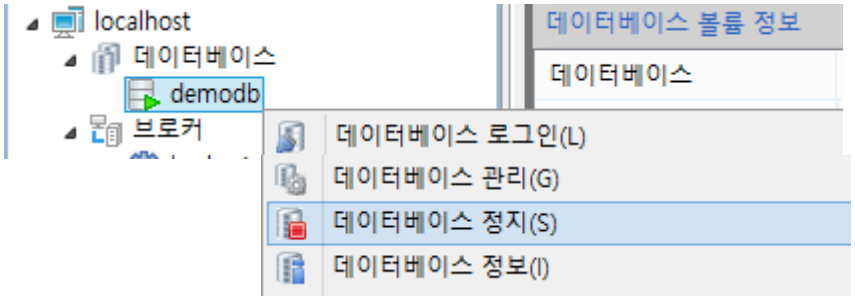
CUBRID Server 종료

- 데이터베이스 종료
  - 사용하는 데이터베이스 별 서버 종료

- 명령어
  - CUBRID 사용자 계정으로 로그인

```
$ cubrid server stop demodb
@ cubrid server stop: demodb

Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
```

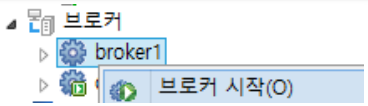
- Windows
  - CUBRID Manager에서 database dba 계정으로 로그인



CUBRID Broker 구동/종료

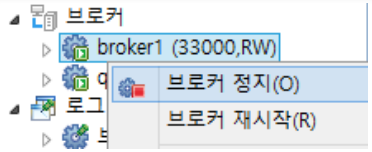
- 구동
  - CUBRID가 설치되어 있는 호스트의 브로커 구동.
  - CUBRID service 구동 시 자동으로 구동된다.

```
$ cubrid broker start
@ cubrid broker start
++ cubrid broker start: success
→ 이미 구동되어 있을 경우  아래와 같이 출력
++ cubrid broker is already running.
```



- 종료
  - CUBRID가 설치되어 있는 호스트의 브로커 종료.
  - CUBRID service종료 시 자동으로 종료.

```
$ cubrid broker stop
@ cubrid broker stop
++ cubrid broker stop: success
→ 이미 종료되어 있을 경우
++ cubrid broker is not running.
```



CUBRID Manager Server 구동/종료

- 구동
  - CUBRID가 설치되어 있는 호스트의 Manager Server 구동.
  - CUBRID service 구동 시 자동으로 구동된다.

```
$ cubrid manager start
@ cubrid manager server start
++ cubrid manager server start: success
→ 이미 구동되어 있을 경우  아래와 같이 출력
++ cubrid manager server is already running.
```

- 종료
  - CUBRID가 설치되어 있는 호스트의 Manager Server 종료.
  - CUBRID service종료 시 자동으로 종료.

```
$ cubrid manager stop
@ cubrid manager server stop
++ cubrid manager server stop: success
→ 이미 종료되어 있을 경우
++ cubrid manager server is not running.
```

5. 데이터베이스 생성

데이터베이스 생성

- CUBRID engine을 설치한 후 사용할 데이터베이스를 생성한다.
  - CUBRID 설치 시 demodb를 생성할 수 있으며 demodb는 사용자의 편의를 위해 sample Data를 가지고 있는 데이터베이스로 CUBRID 테스트 용도로 사용이 가능하다.
  - 프로젝트에 사용할 데이터베이스를 생성해 서비스할 것을 권장한다.

- 데이터베이스 생성
  - 데이터 보존 연한간 전체 데이터의 크기를 예측해 생성한다.
  - 데이터베이스의 생성된 크기는 줄일 수는 없으므로 주의해야 한다.

- 기본 정보
  - 페이지 크기 : 16384 bytes(디스크 I/O 단위이며, 성능상 무난한 크기임)
  - 볼륨 : 스키마나 데이터정보 등이 저장되는 공간
    - 32bit 사용시 한 개 볼륨의 최대 크기 : 2G
    - 범용 볼륨 : 스키마정보, 카다로그 정보 저장됨.
    - 로그 볼륨 : 데이터 변경에 관련된 정보 저장. 백업 복구 시 사용됨.
    - 데이터, 인덱스 볼륨 : 각 데이터, 인덱스 저장됨.
    - 템프 볼륨 : 질의 처리 및 정렬(sorting)을 수행할 때 중간, 최종 결과를 임시로 저장하는 공간

- 디스크 구성 Tip
  - 일반적으로 디스크 RAID 구성은 1+0 또는 0+1이 안성정과 I/O 성능이 가장 좋다.
  - RAID 1+0 , 0+1 구성이 불가능한 환경이라면 RAID-5 구성을 권장한다.

데이터베이스 생성 - 계속

- 명령어 : createdb
  - 특수문자로 시작하는 데이터베이스 명은 사용할 수 없다.
  - 데이터베이스 명은 최대 17자까지만 사용할 수 있다.
  - DB 생성 명령어

```
$ cubrid createdb [options] database_name locale_name.charset
```

  - cubrid: CUBRID 서비스 및 데이터베이스 관리를 위한 통합 유틸리티이다.
  - createdb: 새로운 데이터베이스를 생성하기 위한 명령이다.
  - database_name: 데이터베이스가 생성될 디렉터리 경로명을 포함하지 않고, 생성하고자 하는 데이터베이스의 이름을 고유하게 부여한다. 이 때, 지정한 데이터베이스 이름이 이미 존재하는 데이터베이스 이름과 중복되는 경우, CUBRID는 기존 파일을 보호하기 위하여 데이터베이스 생성을 더 이상 진행하지 않는다.
  - locale_name: 데이터베이스에서 사용할 로캘 이름을 입력한다.
  - charset: 데이터베이스에서 사용할 문자셋을 입력한다. CUBRID에서 사용 가능한 문자셋은 iso88591, euckr, utf8이다.
    - locale_name이 en_US이고 charset을 생략하면 문자셋은 iso88591이 된다.
    - locale_name이 ko_KR이고 charset을 생략하면 문자셋은 utf8이 된다.
    - 나머지 locale_name은 charset을 생략할 수 없으며, utf8만 지정 가능하다.

데이터베이스 생성 - 계속

- 명령어 : createdb [options]

옵션	옵션 천체 이름	설 명	초기값
-F	--file-path	초기 볼륨이 위치할 경로 지정	현재
-L	--log-path	로그 볼륨이 위치할 경로 지정	현재
-B	--lob-base-path	LOB파일이 저장될 위치 경로 지정	<File-path>/lob
-r	--replace	DB가 이미 존재하는 경우 기존 데이터베이스를 삭제하고 재생성함.	존재 시 에러 발생
	--db-volume-size	생성되는 데이터베이스 볼륨의 크기를 바이트 단위로 지정한다.	512M
	--db-page-size	데이터베이스 페이지의 크기를 바이트 단위로 지정한다.	16K
	--log-volume-size	로그 볼륨의 크기를 지정한다.	db_volume_size와 동일
	--log-page-size	로그 볼륨의 페이지 크기를 바이트 단위로 지정한다	db_page_size와 동일

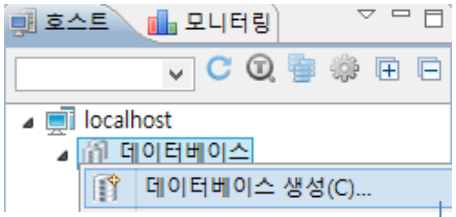
- 사용 예제

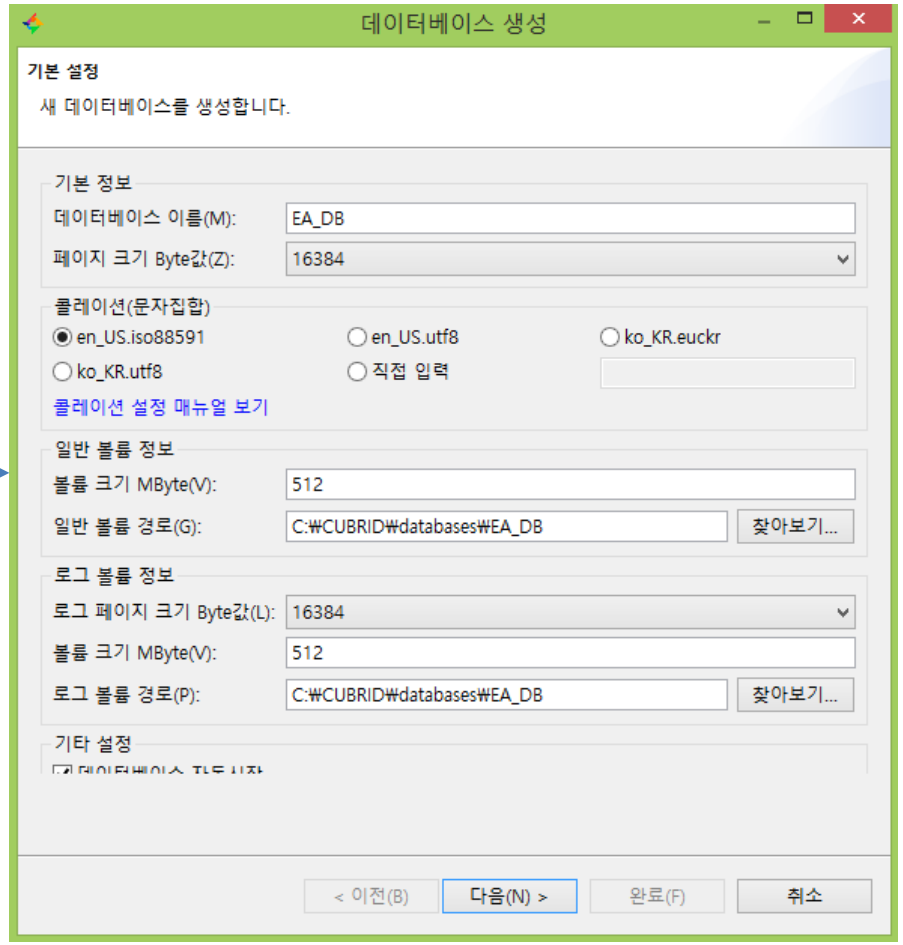
```
cubrid createdb --db-volume-size=512M -F /data --log-volume-size=200M -L /log edudb ko_KR.utf8
```


CUBRID

데이터베이스 생성 - 계속

- CUBRID Manager를 이용한 DB생성





1. 기본정보
    1. 생성할 데이터베이스 이름
    2. Page크기 (default 16K)
2. 콜레이션(문자집합)
    1. 데이터 문자 셋 선택
3. 일반 볼륨 정보
    1. 초기 생성 볼륨의 크기
    2. Page 수(생성 볼륨 크기 환산 값)
    3. 초기 볼륨 생성 위치
4. 로그 볼륨 정보
    1. page 크기(default는 DB page값과 동일
    2. 로그 볼륨 크기
    3. 로그 볼륨 page 수
    4. 로그 볼륨 생성 위치

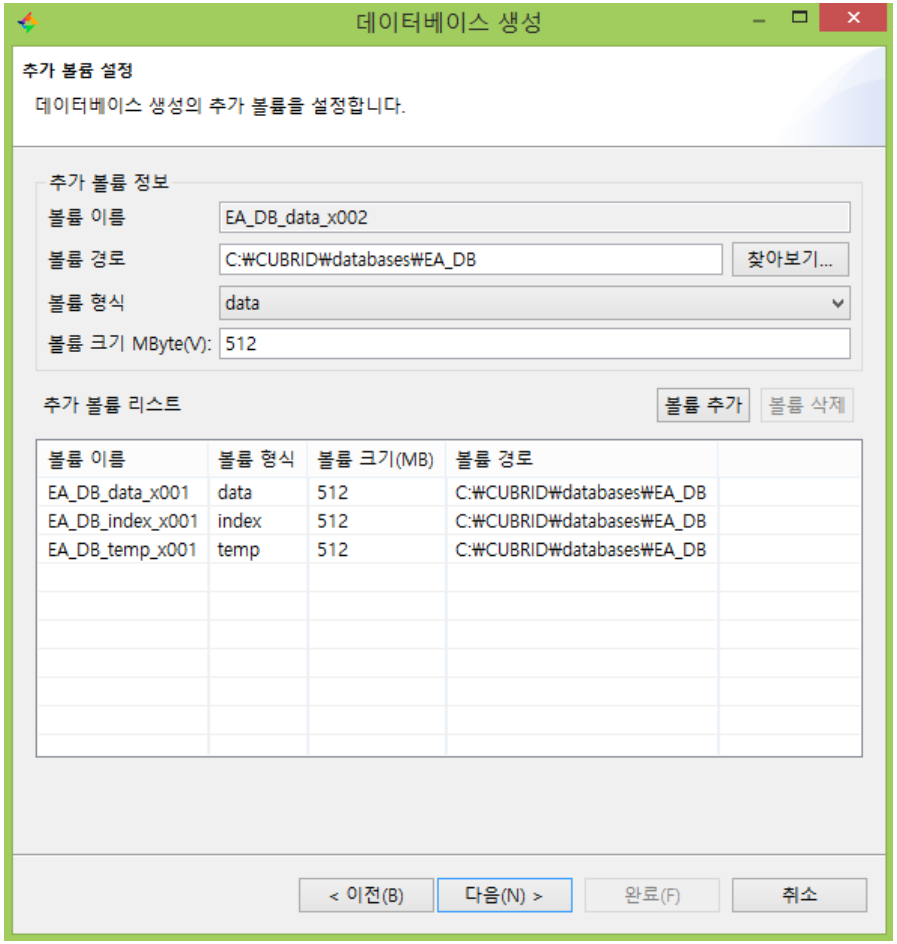
데이터베이스 생성 - 계속

- 볼륨 추가
  - 용도별 볼륨을 분산하는 것이 성능상 효율적이다.
  - 운영상 적합한 크기를 예측해 사용 용도별 볼륨 확장 필요하다.

1. 추가 볼륨 정보
    1. 추가할 볼륨의 이름을 기록
    2. 생성될 볼륨 위치
    3. 볼륨의 용도
    4. 볼륨 크기
    5. 볼륨 Page 수(볼륨 크기의 page환산 값)

2. 추가 볼륨 리스트

- Default로 추가되도록 설정된 볼륨으로 삭제하고 다시 추가하는 것이 가능. 리스트를 클릭 후 볼륨 삭제 버튼을 이용하여 제거 가능하다.

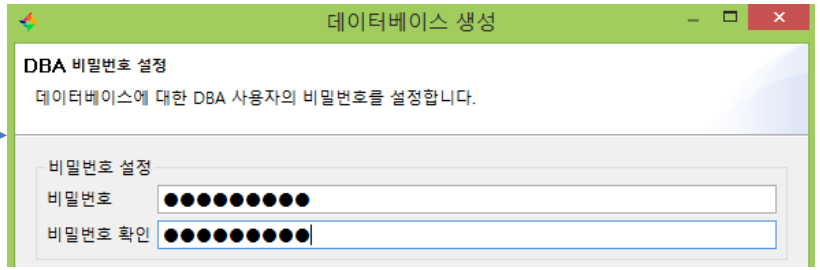


데이터베이스 생성 - 계속

- 볼륨 자동 증가 설정(CUBRID Manager에서만 지원)
  - 각각의 볼륨의 여유 공간 부족 시 볼륨 자동 생성한다.
  - 확장 볼륨 기준(여유공간 비율) 및 자동 확장 볼륨 크기 지정한다.

1. 볼륨 형식

- 데이터, 인덱스 볼륨에 대하여 여유 공간 비율 설정 및 생성될 볼륨 크기를 기록한다. 확장 페이지 수는 볼륨 크기에 따라 자동 변환 된다.
- 데이터와 인덱스 볼륨에 대해서만 자동 볼륨 추가 설정이 가능하다.



- DBA 비밀번호 설정
  - DBA의 비밀번호를 설정한다.
  - Console작업 시 dba의 default 비밀번호는 없다.

- 데이터베이스 설치 완료
  - 지금까지 설정한 내용들을 보여주고 DB설치 종료.

볼륨 추가

- 명령어 사용 : addvoldb
  - 볼륨 추가 명령어

```
$ cubrid addvoldb   [options]   database_name
```

옵션	옵션 전체 이름	설　　명	초기값
-F	--file-path	추가되는 데이터베이스 볼륨이 생성될 디렉터리 경로를 지정한다.	DB생성경로
-p	--purpose	추가되는 데이터베이스 볼륨의 용도를 지정한다.	Generic볼륨
-S	--SA-mode	Off-line 상태에서 데이터베이스 볼륨 추가 작업을 실행한다.	On-line
-C	--CS-mode	On-line 상태에서 데이터베이스 볼륨 추가 작업을 실행한다.	On-line
-n	--volume-name	추가되는 데이터베이스 볼륨의 이름을 지정한다.	DBname_x001
	--db-volume-size	추가되는 데이터베이스 볼륨의 크기를 지정한다.	512M

* -F 옵션의 경우 초기 값이 시스템 parameter의 volume_extension_path의 설정 값을 따른다. 기본값은 DB생성 경로이다.

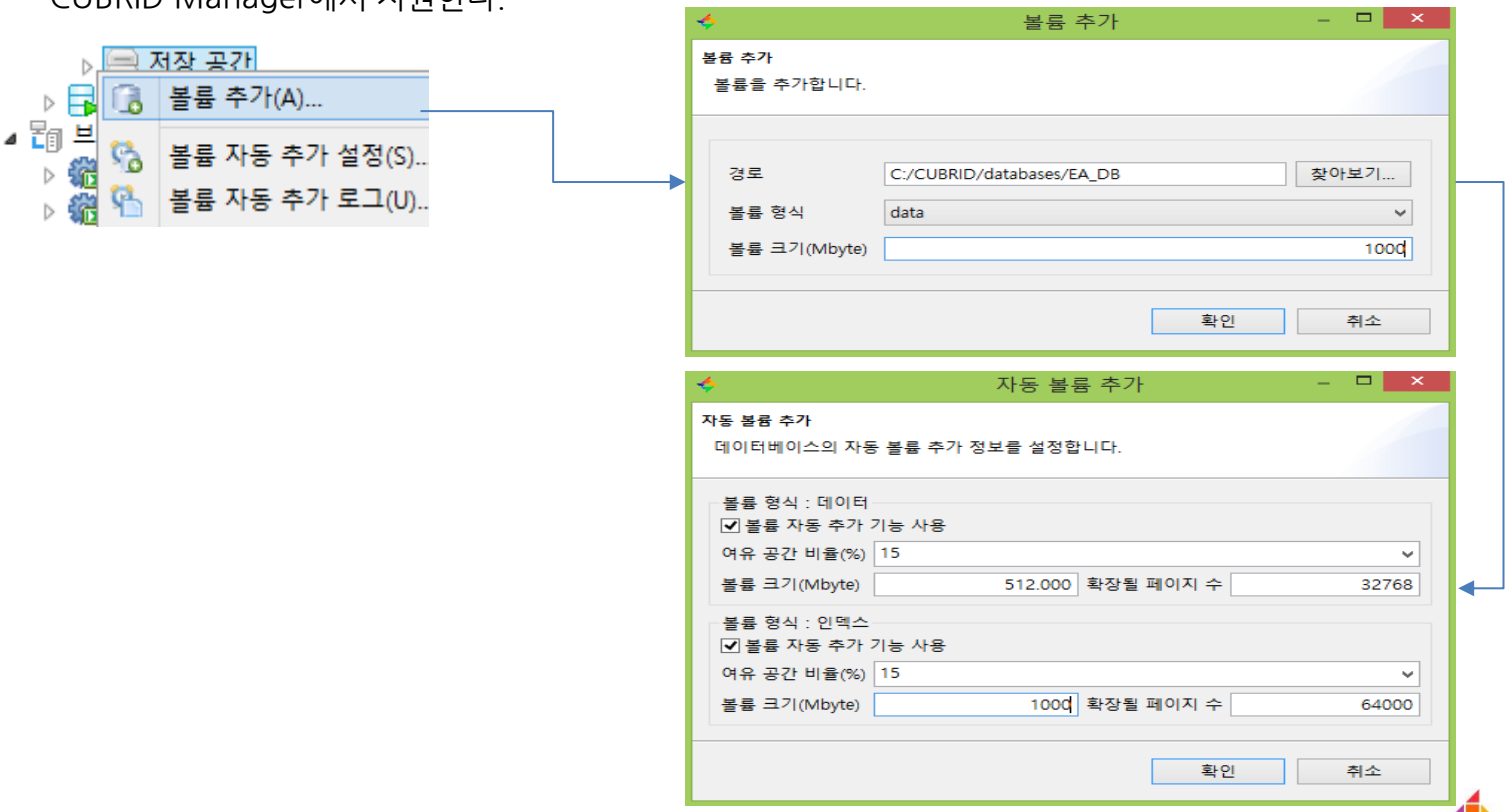
- 사용 예제
  - 볼륨
    - 일반위치: /data, 로그위치: /log, 데이터 2G, 인덱스 1G, 템프 1G 생성

```
$ cubrid addvoldb -S -p data -F /data -n edudb_data_x001 --db-volume-size=2G edudb
$ cubrid addvoldb -S -p index -F /data -n edudb_index_x002 --db-volume-size=1G edudb
$ cubrid addvoldb -S -p temp -F /data -n edudb_temp_x003 --db-volume-size=1G edudb
```

볼륨 추가 - 계속

- 볼륨 추가
  - Data, Index 볼륨의 여유 공간이 부족할 경우 볼륨을 추가한다.

- 확장 자동화
  - Data, Index 볼륨의 여유 공간이 부족할 경우 볼륨 자동 생성 설정한다.
  - 확장될 볼륨의 기준(여유공간 비율) 및 자동 확장이 되는 볼륨의 크기를 지정한다.
  - CUBRID Manager에서 지원한다.



6. 백업과 복구

데이터베이스 백업 시 참고사항

- 데이터베이스 백업 시 참고사항
  - 데이터베이스 백업은 데이터베이스의 이미지를 파일 등으로 덤프 하는 것이다 .
  - OS 유틸리티(cp, sftp/ftp, scp등)를 사용한 데이터베이스 볼륨(파일) 백업은 권장하지 않는다.
  - 데이터베이스 백업 시점에 불필요한 로그 아카이브 볼륨들을 정리 할 수 있다.
  - 백업은 매일 받는 것을 권장하며, DBA가 수작업으로 하는 것보다는 자동 백업을 하는 것이 편리하다.
  - 백업 볼륨은 외부 저장장치에 백업하는 것이 안전하다.
  - 데이터베이스를 새로운 버전으로 migration했다면 새로 생성한 데이터베이스를 즉시 백업해야 한다. 이전 버전의 백업 볼륨은 새로운 버전 복구에 사용 불가능하기 때문이다.
  - 백업 볼륨을 이동하거나 이름을 변경하지 않으면 이후의 백업 시에 이전 볼륨을 덮어쓴다.
  - 백업 볼륨은 데이터베이스를 가장 최근의 상태로 복구하기 위해서 로그 볼륨과 함께 사용된다.

- 데이터베이스 백업 정책
  - 백업할 데이터의 선택
    - 데이터베이스의 전체 또는 일부만 백업 할 것인가?
    - 데이터 보존 기간은 얼마로 할 것인가?
    - 데이터베이스와 함께 백업되어야 할 다른 파일은 있는가?
  - 백업 방법(backupdb)
    - 사용 가능한 백업 툴 및 백업 장비
    - FULL/INCREMENTAL백업(0,1,2 level)
    - On-line/Off-line 백업
  - 백업 시기
    - 데이터베이스의 활동이 가장 적은 시기

백업

- 명령어 이용한 백업: backupdb

```
$ cubrid  backupdb  [options]  database_name
```

  - Disk와 tape등의 미디어 지정이 가능하다.
  - 백업이 완료되면 백업볼륨 및 백업정보 파일을 생성한다. (예, demodb_bk0v000, demodb_bkvinf)

옵션	인자	설명	기본값
-S | -C		stand-alone, client-server mode 지정(On/Off-line)	-C
-D	filepath/device	볼륨이 저장될 경로 지정	log file 경로와 같음
-l	0,1,2	백업 레벨	0
-r		백업 후 불필요한 archive log 삭제	수행 않음
--no-check		백업 전에 데이터베이스 일관성 점검을 수행하지 않는다.	수행
-z		데이터베이스를 압축하여 백업 볼륨에 저장함	수행 않음
-t	integer	백업을 수행하는 thread 수	0<auto>
-e		백업 시 활성 로그 볼륨을 포함하지 않도록 설정 함	수행 않음

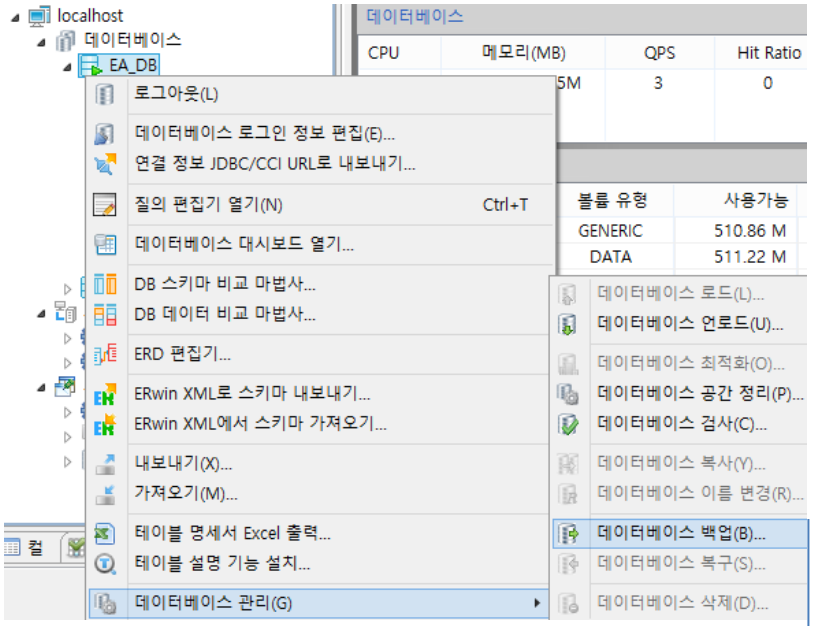
- 사용 예

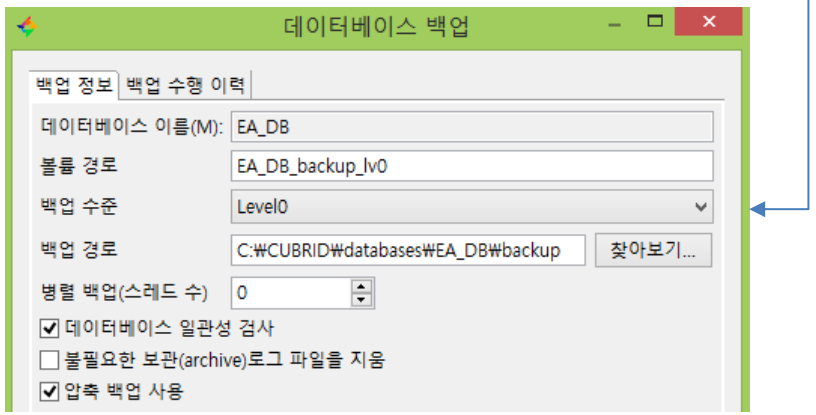
```
$ cubrid  backupdb  -C -z -r -D /CUBRID/databases/demodb -l 0 demodb
→ windows% cubrid backupdb -C -z -r -D C:₩CUBRID₩databases₩demodb -l 0 demodb
```

백업 - 계속

- CUBRID Manager를 이용한 백업 방법

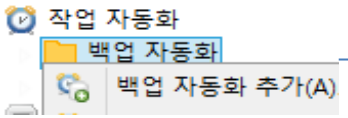
1. 데이터베이스를 선택
2. 백업 정보
   - 백업 볼륨의 이름을 명시한다
   - 백업 레벨을 선택한다. Full backup을 받은 적이 없는 경우 level 0만 표시된다.
   - 백업 경로를 선택한다.
   - 백업 작업을 수행할 thread개수를 설정한다.
   - 데이터베이스 일관성 검사(DB이상 유무 확인)
   - 보관로그 파일 지움(archive log 정리)
   - 압축 백업 사용(백업 파일을 압축하여 저장한다.



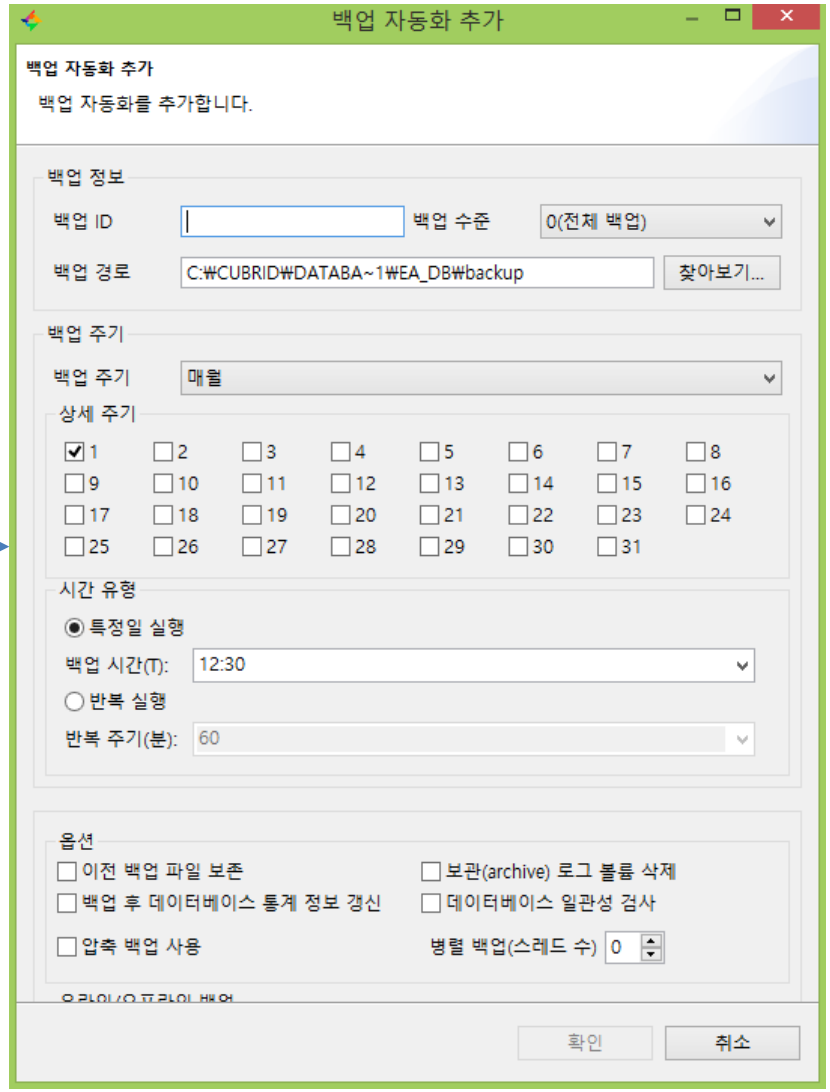


백업 - 계속

- CUBRID Manager를 이용한 백업 자동화 설정



1. 백업 정보
   - 백업 ID : 임의 값 설정
   - 백업 level 선택 : 처음 백업 시 level 0만 가능
   - 백업 경로 : 별도의 Disk 또는 media 권장
2. 백업 주기
   - 백업 주기 : 디스크/백업파일 사이즈에 따라 선택
   - 상세 주기 : 백업주기에 따라 날짜, 요일 등
   - 백업 시간 : DB활동이 가장 적은 시기(시, 분)
3. 옵션
   - 이전 백업 파일 보존 : 이전 단계 복구가 필요할 경우, - 보관 로그 볼륨 로그 삭제: archive log 정리 여부(권장)
   - 백업 후 데이터베이스 통계 정보 갱신 : DB에 갱신 작업이 빈번하게 발생하는 경우 주기적인 작업이 필요
   - 데이터베이스 일관성 검사 : 백업 시 DB 이상 유무를 확인, - 압축 백업 사용 : 사용할 것을 권장
   - 병렬 백업 : 백업 작업에 이용될 thread 수
4. 온/오프라인 백업 : 오프라인 백업 시 DB가 정지 된 후 백업을 진행하고 DB가 구동된다.



CUBRID

복구

- 명령어를 이용한 복구: restoredb

```
$ cubrid restoredb   [options]   database_name
```

옵션	인자	설명	기본값
-l	복구 레벨	복구 레벨을 지정한다(0, 1, 2)	0
-d	복구 시점	복구할 시점을 지정 형식) dd-mm-yyyy:hh:mm:ss 예) 21-12-2002:17:00:10. 또는 backuptime : 마지막 백업 시점으로 복구시 사용	가장 최근 시점
-B	파일 패스	복구할 백업 볼륨이 존재하는 디렉터리나 장치 명을 지정	dbname_bkvinf 경로
-p		archive 로그가 없을 경우 무시하고 수행하여 사용자 인터페이스를 받지 않을 경우	
-u		데이터베이스 위치 파일 내에 지정된 경로로 데이터베이스와 로그 볼륨을 복구 (databases.txt)	
--list		백업을 수행하지 않고 백업 볼륨 목록을 출력할 경우	

복구 - 계속

- 데이터베이스 복구
  - 백업 시점으로 복구 또는 백업 이후 원하는 시점을 복구 가능하다.
  - restoredb 명령어 수행 시 -d 옵션을 사용하지 않으면 복구가능한 가장 최근 시점으로 자동 복구한다.

- 데이터베이스 복구방법
  - 최근 시간/시점으로 복구

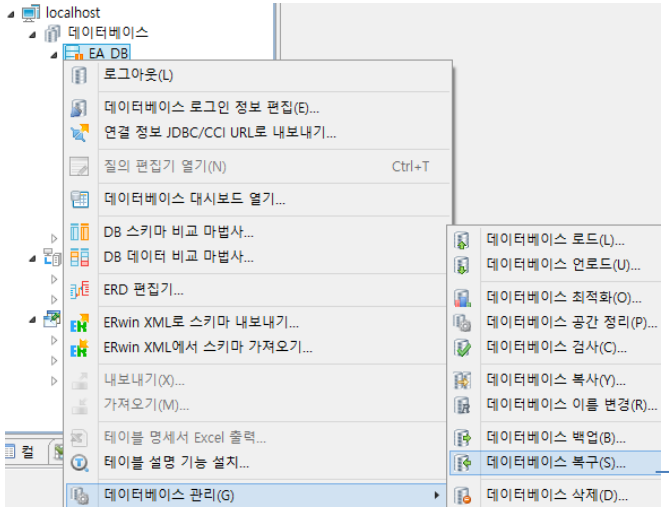
```
$ cubrid restoredb -d 19-02-2015:17:39:00 demodb
```

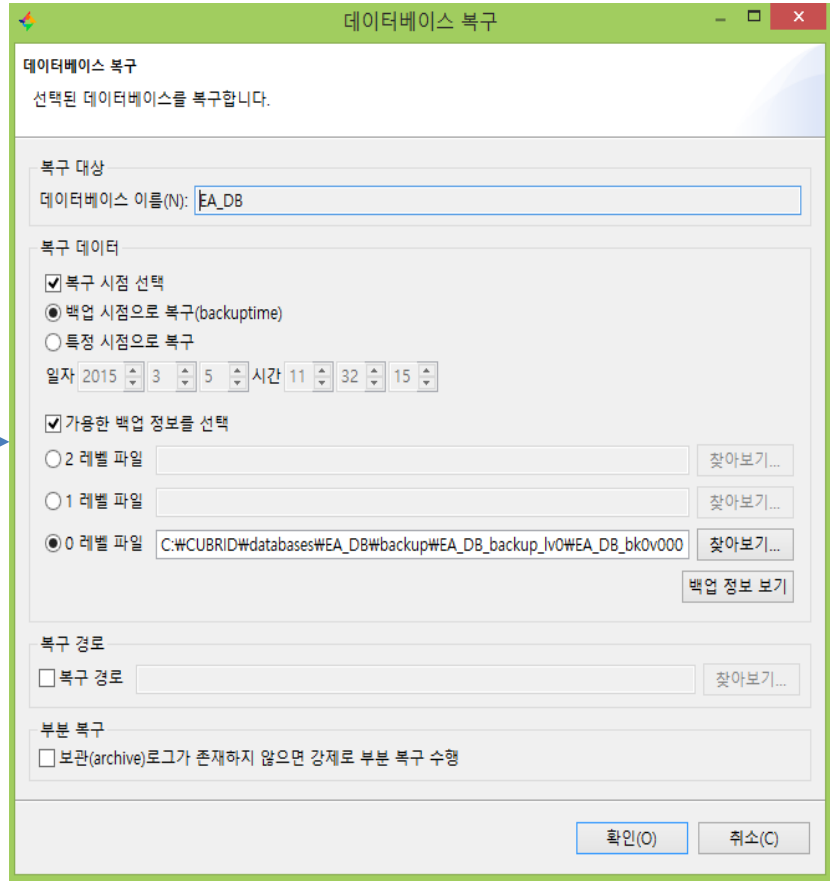
  - 백업 시간/시점으로 복구

```
$ cubrid -d backuptime restoredb demodb
```

복구 - 계속

- CUBRID Manager를 이용한 데이터 복구
  - DB Server 종료
  - 백업 시점으로의 복구 및 백업 이후 원하는 시점을 복구 가능하다.





1. 복구 대상 데이터베이스 이름
2. 복구 데이터
   - 복구 시점 선택 : 백업 시점 또는 특정 시점
   - 가용 가능한 백업 정보 선택 : <db_name>_bkvinf 파일 에 백업의 경로가 등록되어 있을 경우 자동으로 검색 된다.
3. 복구 경로 : 백업을 복구할 위치를 지정할 수 있다.
4. 부분 복구 : archive log 부재 시 백업 볼륨만 가지고 복구

데이터베이스 복구 사례

- 데이터베이스 복구
  - 디스크 장애(media failure) 발생
    - H/W(물리적) 디스크 복구 후 데이터베이스 볼륨(파일) 복구에 실패하거나 손상이 있을경우 backupdb로 받은 백업볼륨을 이용해서 restoredb 명령으로 복구한다.
  - 데이터베이스 특정시점 복원
    - Where 조건이 없는 delete/update로 인한 특정 시점 복원이 필요할 때 백업된 데이터베이스와 로그를 이용해서 복구한다. 이때 백업 볼륨과 로그볼륨(archive log, active log)은 반드시 필요한 위치에 있었야 한다.

- 로그 복구
  - 로그복구는 CUBRID Manager에서는 지원하지 않는 기능이다.
  - 데이터베이스가 비정상 종료 시 손상된 로그 복구한다.
  - 복구 시 시간이 많이 걸릴 수 있으며, 강제 종료 시 손상이 더 심해질 수 있다.

```
$ csql -S -u dba demodb
```

  - 위의 명령 수행 중 log 관련 에러 발생으로 실패 시 다음 명령 수행한다.

```
$ cubrid emergency_patchlog demodb
```

  - 위 명령어 수행 후 csql 접속 시 이상 없으면 복구 완료, 여전히 에러 발생하면 다음 명령 수행한다.

```
$ cubrid emergency_patchlog -r demodb
```

  - 위 명령어 수행 후 csql 접속 시 이상 없으면 복구 완료, 여전히 에러 발생하면 “백업“ 을 이용한 복구 수행한다.

7. 데이터베이스 재구성

데이터베이스 재구성

- 데이터베이스 재구성의 이유
  - 운영 중 데이터베이스 장애복구 또는 최적화를 위해 재구성을 진행할 경우가 있다.
  - CUBRID 버전을 업그레이드할 경우 볼륨 호화성 가능 여부에 따라 재구성이 필요할 수 있다.
  - DB서버 이관 시 CUBRID 버전이 다르면 재구성이 필요할 수 있다.

- 데이터베이스 재구성 방법
  - 언로드(export) : 원본 데이터베이스를 파일로 내려 받기
  - 백업(backup) : 작업 오류를 대비해 backupdb/copydb/renamedb를 사용한 백업
  - Rebuilding : 기존의 데이터베이스 삭제 및 새로운 데이터베이스 생성
  - 로드(import) : 언로드 받은 파일을 새로운 데이터베이스에 적재

데이터베이스 재구성 - 계속

- 데이터베이스 재구성을 위한 유틸리티 : 언로드, 로드, 이름변경

- 데이터베이스 언 로드
  - 현재의 데이터베이스를 파일로 내려 받는다.
  - 특정 테이블에 대하여 백업을 하고 싶은 경우 사용한다.
  - 데이터베이스의 전체 스키마를 확인할 경우 사용한다.
  - 생성파일
    - 스키마 파일 : 데이터베이스의 스키마 정의 포함하는 파일(demodb_schema)
    - 객체 파일 : 데이터베이스의 데이터를 포함한 파일 (demodb_objects)
    - 인덱스 파일 : 데이터베이스에 정의된 인덱스를 포함하는 파일 (demodb_indexes)
    - 트리거 파일 : 데이터베이스에 정의된 트리거를 포함하는 파일 (demodb_triger)

- 데이터베이스 로드
  - 언 로드 받은 파일들을 데이터베이스로 등록한다.
    - 언 로드 형식으로 작성된 파일도 수행 가능하며 해당 테이블에 대하여 권한이 있는 계정으로 수행한다.
  - stand-alone 모드에서 DBA 권한으로 수행한다.
    - 언 로드 한 데이터를 로딩하는 것이므로 각종 시스템 테이블이 포함되어 있어 dba 권한으로 수행을 권장한다.
  - 언 로드 된 데이터 량을 확인 하고 적정 크기의 데이터베이스가 존재해야 한다.
  - 기존 데이터가 존재하는 데이터베이스에 로드작업을 할 경우 로드 될 데이터가 적합한지 확인 한다.

- 데이터베이스 이름변경
  - 오류 발생 시 앞선 DB를 바로 사용하기 위해 DB를 삭제하지 않고 이름 변경해서 백업할 수 있다.
  - 백업/복구 과정보다 빨라 이전 DB를 백업하는 방법으로 많이 사용된다.

데이터베이스 재구성 - 계속

- 데이터베이스 재구성 유틸리티 언로드
- 명령어 사용 : unloaddb

```
$ cubrid unloaddb [options] database_name
```

옵션	옵션전체	설명	기본값
-i	--input-class-file	인수로 지정된 입력 파일에 지정된 클래스를 대상으로 데이터베이스를 언 로드 한다.	
-o	--output-path	스키마와 객체 파일이 생성될 디렉터리를 지정한다. 옵션이 지정되지 않으면 현재 디렉터리에 생성된다	현재 위치
-s	--schema-only	데이터 파일은 생성하지 않고, 스키마 파일만 생성한다.	
-d	--data-only	스키마 파일은 생성하지 않고, 데이터 파일만 생성한다.	
-v	--verbose	언 로드 되는 데이터베이스의 상세 정보를 화면에 출력한다.	수행 않음
-S	--SA-mode	독립 모드에서 데이터베이스를 언 로드 한다.	
-C	--CS-mode	클라이언트/서버 모드에서 데이터베이스를 언로드한다.	
	--include-reference	-i 옵션과 함께 사용되며, 객체 참조도 함께 생성한다.	
	--input-class-only	-i 옵션과 함께 사용되며, 입력 파일에 포함된 테이블에 관한 스키마 파일만 생성한다.	
	--output-prefix	스키마와 객체 파일명 앞에 붙이는 prefix를 지정한다.	

데이터베이스 재구성 - 계속

- 데이터베이스 재구성 유틸리티 로드
- 명령어 사용 : loaddb

```
$ cubrid loaddb [options] database_name
```

옵션	옵션전체	설명	기본값
-u	--user	데이터베이스 사용자의 계정을 입력한다.	public
-p	--password	데이터베이스 사용자의 암호를 입력한다.	
-l	--load-only	객체 파일에 포함된 구문과 데이터 타입 검사를 생략하고 레코드를 로드 한다.	구문 및 타입 검사 진행
-v	--verbose	데이터 로딩 상태에 관한 상세 정보를 화면에 출력한다.	
-c	--periodic-commit	지정된 개수의 레코드가 입력될 때마다 트랜잭션을 커밋 한다.	수행 않음
-s	--schema-file	언 로드 작업에 의해 생성된 스키마 파일을 지정하여, 스키마 로딩을 수행한다.	
-i	--index-file	언 로드 작업에 의해 생성된 인덱스 파일을 지정하여, 인덱스 로딩을 수행한다.	
-d	--data-file	언 로드 작업에 의해 생성된 데이터 파일을 지정하여, 레코드 로딩을 수행한다.	
	--no-statistics	데이터베이스에 관한 통계 정보를 갱신하지 않는다.	
	--ignore-class-file	지정된 파일에 포함된 클래스를 제외하고 로딩 작업을 수행한다.	

데이터베이스 재구성 - 계속

- 데이터베이스 재구성 유틸리티 이름변경
- 명령어 사용 : renamedb

$ cubrid renamedb [OPTION] src-database-name dest-database-name

$ cubrid renamedb demodb demodb_rename

→ 원본 demodb를 경로변경 없이 demodb_rename으로 이름 변경

옵션	인자	설명	기본값
-E	volext_path	target database 확장 볼륨 경로 지정	DB home directory
-I	vol_tofrom_path	각 볼륨에 대해 복사할 경로 파일로 지정	
-d	없음	백업 볼륨과 백업 정보 파일 제거	제거 안 함

데이터베이스 재구성 - 계속

- 재구성 절차
  - 서비스 종료
  - 데이터베이스 백업
    - 데이터베이스 백업, 데이터베이스 복사 중 데이터베이스 이름 변경이 가장 빠르다.

```
$ cubrid renamedb demodb demodb.bak
```

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

```
$ cubrid unloaddb -S -O /data/unload --output-prefix=demodb demodb.bak
```

  - 데이터베이스 생성
    - 데이터베이스 용량을 재 산정해 데이터베이스 생성한다.
  - 스키마 / 데이터 올리기
    - 앞서 내려 받은 스키마/데이터 등을 새로 생성한 데이터베이스에 올린다.

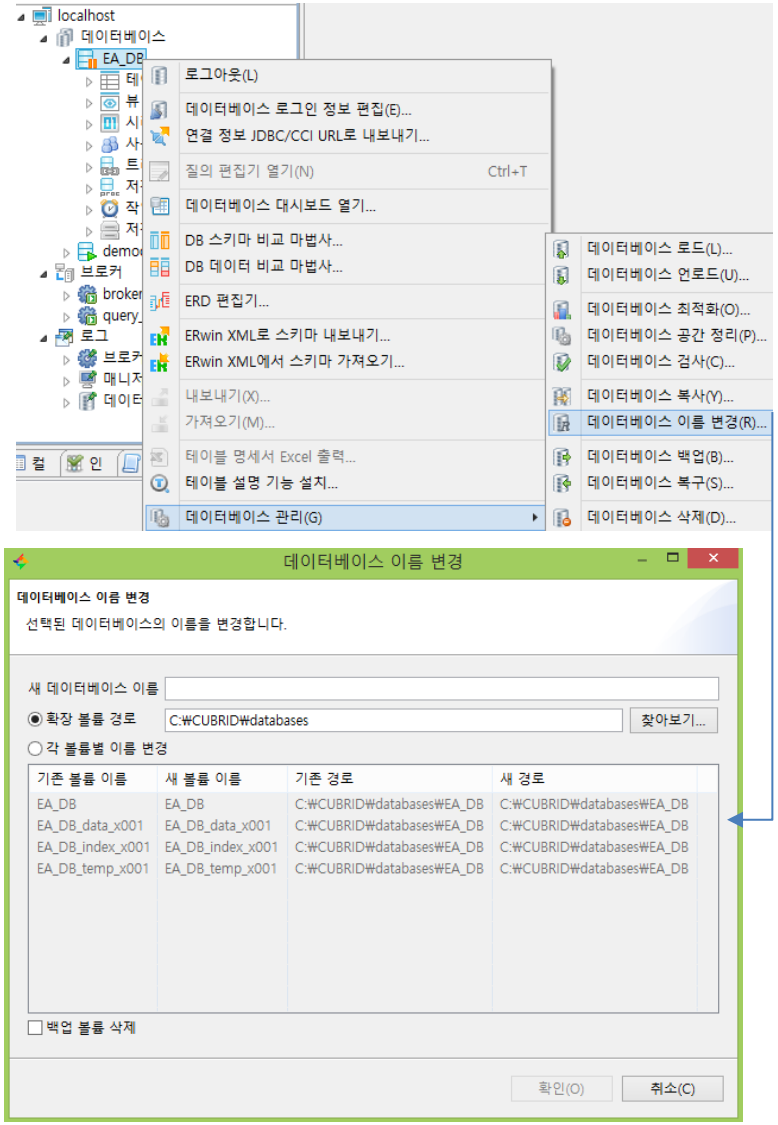
```
$ cubrid loaddb -s /data/unload/demodb_schema demodb
→ --no-oid : 객체개념을 사용하지 않았거나, foreign key option 에 on cache object 미사용 시만 사용
$ cubrid loaddb --no-oid --no-statistics –v -l -c 10000 -d /data/unload/demodb_objects demodb
$ cubrid loaddb -v -i /data/unload/demodb_indexes demodb
$ cubrid loaddb –v -s /data/unload/demodb_trigger demodb
$ cubrid optimized demodb
```

데이터베이스 재구성 - 계속

- CUBRID Manager에서 재구성 하기(이름변경)
  - DB server 정지 상태에서만 수행된다.

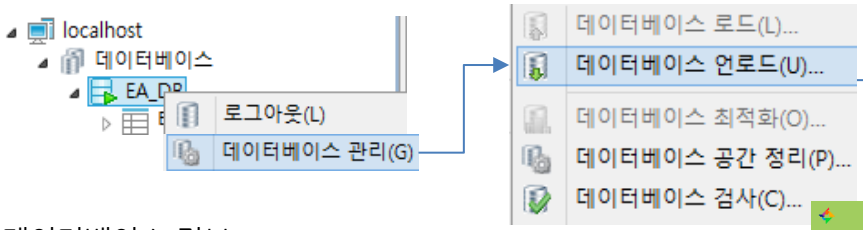
1. 새 데이터베이스 명
   - 변경할 데이터베이스 이름을 명시
2. 옵션
   - 백업볼륨 삭제 : 기존의 백업 삭제 유무
   - 확장볼륨 경로
     추가된 볼륨의 경로를 변경할 경우 명시
   - 각 볼륨 별 이름변경
     볼륨들에 대하여 볼륨의 이름 및 경로를 변경

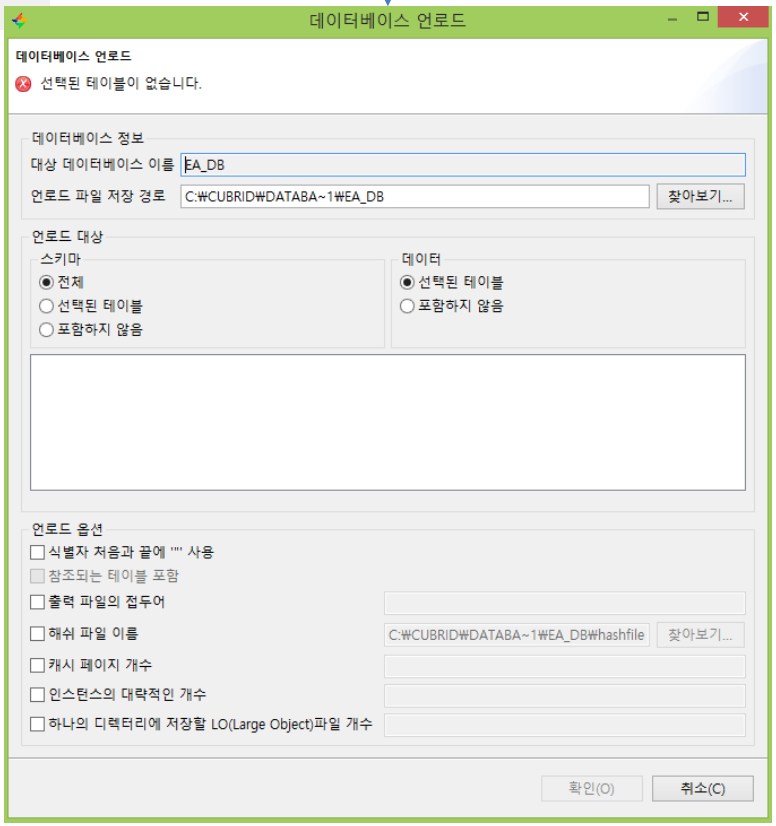
❖ 데이터베이스 이동 및 복사와 다르게 Log볼륨에 대한 경로를 지정 할 수 없다.



데이터베이스 재구성 - 계속

- CUBRID Manager에서 재구성 하기(언로드)

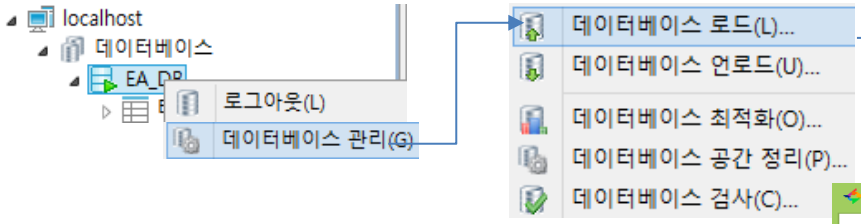


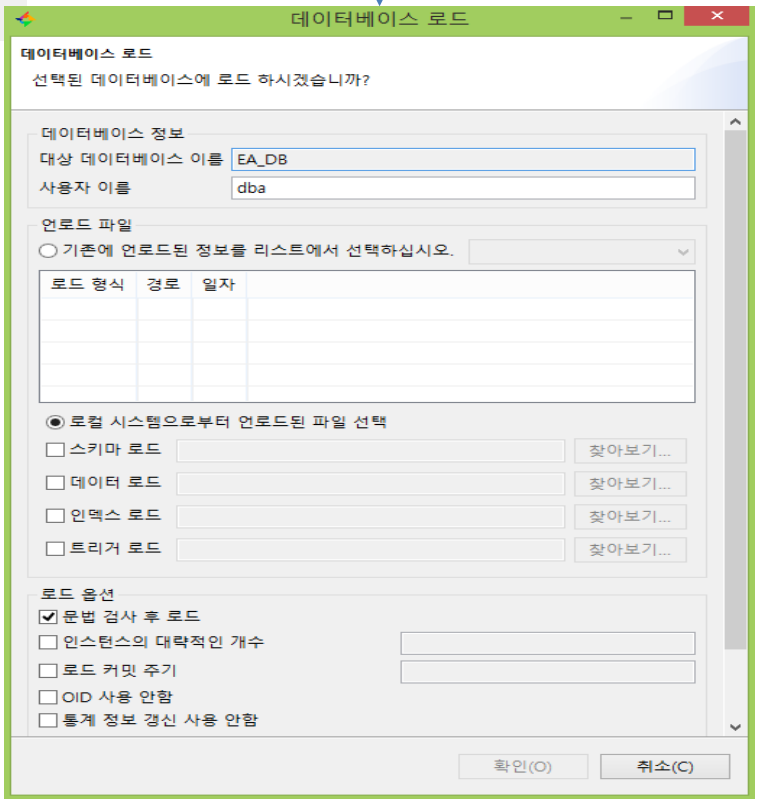


1. 데이터베이스 정보
   - 대상 DB이름
   - 언 로드 파일 저장 경로
2. 언 로드 대상
   - 스키마 : 전체, 선택, 미 선택으로 구분 가능
   - 데이터 : 선택, 미 선택으로 구분.
   - 아래 체크 박스로 선택이 가능함.
3. 언 로드 옵션
   - 식별자 처음과 끝에 “”사용 : 속성이름을 “”로 묶어서 출력
   - 참조테이블 포함 : 선택된 테이블만 받을 경우 활성화
   - 출력 파일의 접두어 : 결과 파일의 접두어 설정
   - 해쉬 파일 이름 : unload작업 공간에 대한 이름 설정여부
   - 캐시 페이지 개수 : 메모리에 캐시 되는 테이블의 수 지정
   - 인스턴스의 대략적인 개수 : 예상되는 레코드 수 지정
   - 하나의 디렉터리에 저장할 LO파일 개수: 개수 명시

데이터베이스 재구성 - 계속

- CUBRID Manager에서 재구성 하기(로드)





1. 데이터베이스 정보
   - 대상 DB이름
   - 사용자 이름
2. 언 로드 파일
   - 내부에 받아 놓은 언 로드 파일의 경로 선택
3. 로드 옵션
   - 문법 검사 후 로드 : 언로드 받은 파일에 대한 문법 검사
   - 문법 검사하지 않고 데이터 로드 : 검사 시간을 단축한다
   - 인스턴스의 대략적인 개수 : 예상되는 레코드 개수
   - 로드 커밋 주기 : 주기적인 커밋이 성능 향상에 필요하다
   - OID사용 안 함 : OID 체크 작업을 생략한다.
   - 통계 정보 갱신 사용 안 함 : 작업 시간 단축, 차후 작업 가능
   - 로드 시 발생하는 오류에 대한 제어 파일 : 특정 에러를 처리하는 방식에 대한 명세 파일 지정
   - 로드 하지 않을 테이블 기록 파일 : 명시된 테이블 작업 생략

데이터베이스 복사 및 이동

- 데이터베이스 재구성 시 백업 방법으로 복사 방법을 활용 가능

- 명령어 사용 : copydb

```
$ cubrid copydb   [OPTION]   src-database-name   dest-database-name
$ cubrid copydb  -F C:₩CUBRID₩databases₩demodb_bk
                 -E C:₩CUBRID₩databases₩demodb_bk
                 -L C:₩CUBRID₩databases₩demodb_bk
                   demodb demodb_bk
```

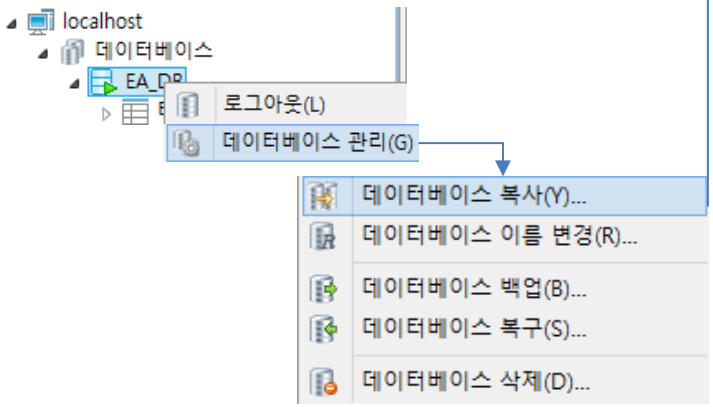
→ 원본 demodb를 "C:₩CUBRID₩databases₩demodb_bk" 경로에 로그 및 확장볼륨 포함하여 demodb_bk로 복사

옵션	인자	설명	기본값
-F	file_path	초기 데이터베이스 볼륨 디렉터리 패스	현재의 디렉터리
-L	log_file_path	데이터베이스 로그 볼륨 디렉터리 패스	초기 볼륨 디렉터리
-E	volext_path	데이터베이스 확장 볼륨 디렉터리 지정	초기 볼륨 디렉터리
-i	to_from_path	각 볼륨에 대해 복사할 경로를 지정한 파일 -E, -F옵션과 같이 사용할 수 없음	없음
-r		target 데이터베이스와 같은 것이 있으면 삭제	수행 않음
-d		복사 후 source 데이터베이스 삭제	수행 않음

데이터베이스 복사 및 이동 - 계속

- 현재의 데이터베이스를 다른 경로에 복사
- DB server 정지 상태에서만 수행된다.
- 로그볼륨 및 확장볼륨의 경로 지정 가능
- 사용자 계정 및 암호 동일하게 복사

1. 원본 데이터 베이스
   - 복사될 이름 및 경로 표시
2. 대상 데이터베이스
   - 원본으로부터 복사할 데이터베이스 이름/경로 명시
   - 확장된 볼륨에 대한 경로 명시
   - 로그 볼륨의 경로 명시
3. 옵션
   - 개별 볼륨 복사에 대한 설정



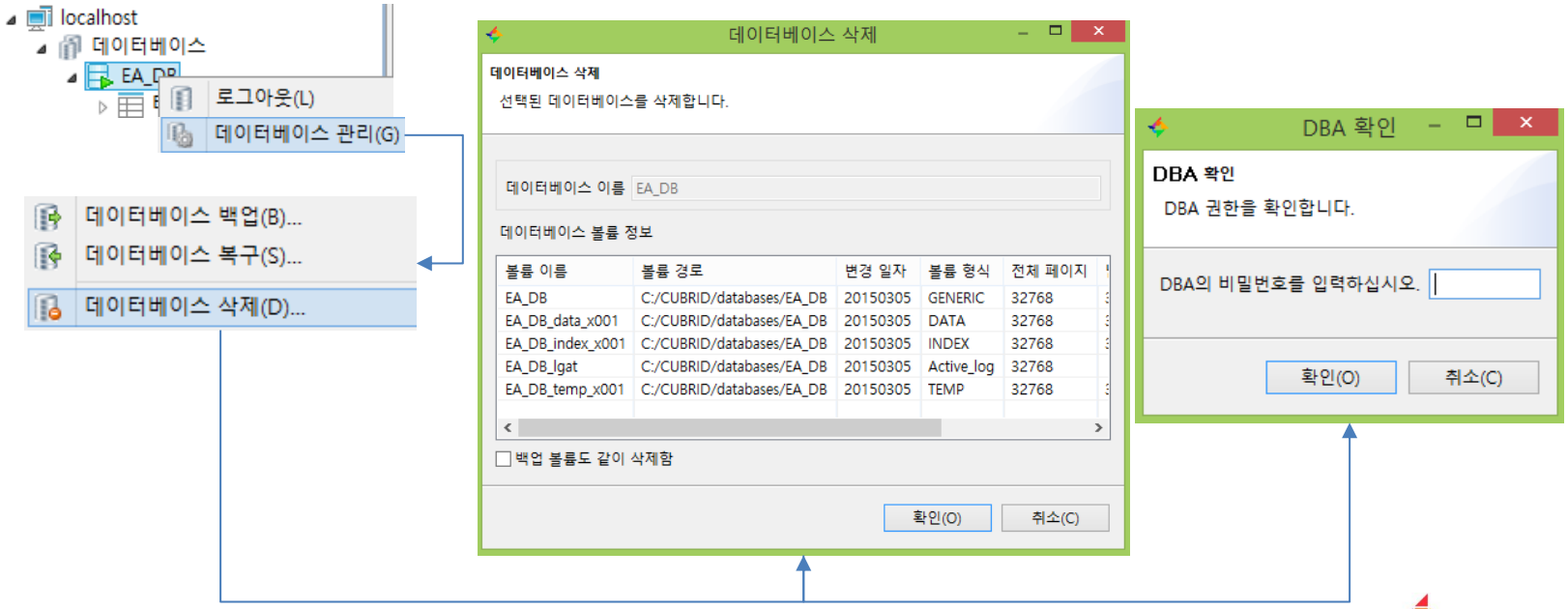
CUBRID

데이터베이스 삭제

- 명령어 사용 : deletedb

```
$ cubrid deletedb   [OPTION]   database-name
$ cubrid  deletedb -d edudb_rename
→ 백업볼륨을 포함하여 삭제
```

- DB server 정지 상태에서만 수행된다.
- 삭제를 수행한 데이터베이스와 관련된 모든 파일이 삭제된다.
  - 백업 볼륨/파일은 옵션을 부여하여 삭제한다.
  - 데이터베이스 삭제는 dba 비밀번호를 입력 이후 삭제된다.



CUBRID

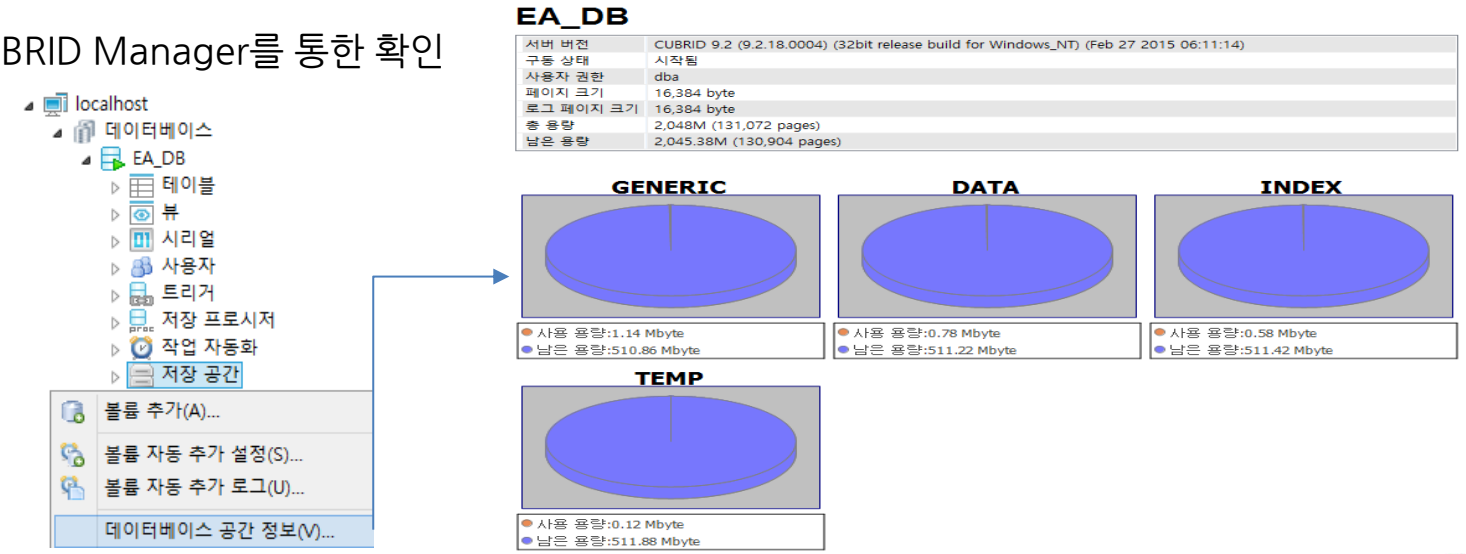
8. 모니터링

데이터베이스 사용량

- 명령어 : cubrid spacedb
  - 데이터베이스 각 볼륨 파일별 사용량 확인한다.
  - 데이터베이스 각 볼륨 용도별(DATA,INDE,TEMP) 총 사용량은 -s 옵션을 추가해 확인한다.

```
$ cubrid spacedb EA_DB;# 서버가 종료되었을 때는 -S 옵션 추가
Space description for database 'EA_DB' with pagesize 16.0K. (log pagesize: 16.0K)
Volid  Purpose    total_size    free_size   Vol Name
    0  GENERIC       512 M       502 M   /data/EA_DB
    1     DATA       512 M       512 M   /data/EA_DB_x001
    2    INDEX       512 M       512 M    /data/EA_DB_x002
    3     TEMP       512 M       512 M    /data/EA_DB_x003
------------------------------------------------------------------------
    4                 2.4 G         2.3 G
Space description for temporary volumes for database 'EA_DB' with pagesize 16.0K.
Volid  Purpose    total_size    free_size   Vol Name
```

- CUBRID Manager를 통한 확인



CUBRID

서비스 프로세스 상태 확인

- 명령어 : cubrid service status
  - 구동 중인 서비스(master, server, broker, manager) 상태 출력한다.

```
$ cubrid service status
@ cubrid master status
++ cubrid master is running.
@ cubrid server status
 Server demodb (rel 9.2, pid 15480)
@ cubrid broker status
  NAME              PID  PORT   AS   JQ          TPS          QPS   SELECT   INSERT   UPDATE   DELETE   OTHERS    LONG-
T    LONG-Q       ERR-Q  UNIQUE-ERR-Q  #CONNECT
================================================================================================================
================================================================================================================
===============
* query_editor       15344 30000    5   0             0            0      0      0      0      0      0    0/60.0    0/60.0
0          0       0
* broker1           15355 33000    5   0             0            0      0      0      0      0      0    0/60.0    0/60.0         0
0       0
@ cubrid manager server status
++ cubrid manager server is running.
```

  - Master status : 하나의 master 보여진다.
  - Server status : 구동된 수 만큼의 DB가 보여진다.
  - Broker status : 구동된 Broker가 보여진다.
  - Manager server status : Cubrid manager Client가 접속하여 관리가 가능하도록 해주는 Manager Server 구동 여부로써 not running일 경우 CUBRID Manager client는 접속 할 수 없다.

데이터베이스 트랜잭션 확인

- 명령어 : cubrid tranlist
  - cubrid tranlist는 대상 데이터베이스의 트랜잭션 정보를 확인하는 유틸리티로서, DBA 또는 DBA그룹 사용자만 수행할 수 있다.

```
$ cubrid tranlist [options] database_name
```

옵션	설명
databases_name	지정한 데이터베이스에 관한 트랜잭션 정보를 출력한다.
-u, --user=USER	로그인할 사용자 ID. DBA및 DBA그룹 사용자만 허용한다.(기본값 : DBA)
-p,--password=PASSWORD	사용자 비밀번호
-s, --summary	요약 정보만 출력한다(질의 수행 정보 또는 잠금 관련 정보를 생략).
--sort-key=NUMBER	해당 NUMBER 위치의 칼럼에 대해 오름차순으로 정렬된 값을 출력한다. 칼럼의 타입이 숫자인 경우는 숫자로 정렬되고, 그렇지 않은 경우 문자열로 정렬된다. 생략되면 "Tran index"에 대한 정렬값을 보여준다.
--reverse	역순으로 정렬된 값을 출력한다.

  - "cubrid tranlist demodb"는 "cubrid killtran -q demodb"와 비슷한 결과를 출력하나, 후자에 비해 "User name"과 "Host name"을 더 출력한다. "cubrid tranlist -s demodb"는 "cubrid killtran -d demodb"와 동일한 결과를 출력한다.

데이터베이스 트랜잭션 확인 - 계속

- tranlist 각 컬럼 의미

```
$ $ 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            3, 2, 1
e5899a1b76253   update ta set a = 5 where a > 0
------------------------------------------------------------------------------------------------------------------------
```

- Tran index: 트랜잭션 인덱스
- User name: 데이터베이스 사용자 이름
- Host name: 해당 트랜잭션이 수행되는 CAS의 호스트 이름
- Process id: 클라이언트 프로세스 ID
- Program name: 클라이언트 프로그램 이름
- Query time: 수행중인 질의의 총 수행 시간(단위: 초)
- Tran time: 현재 트랜잭션의 총 수행 시간(단위: 초)
- Wait for lock holder: 현재 트랜잭션이 락 대기중이면 해당 락을 소유하고 있는 트랜잭션의 리스트
- SQL_ID: SQL Text에 대한 ID. cubrid killtran 명령의 --kill-sql-id 옵션에서 사용될 수 있다.
- SQL Text: 수행중인 질의문(최대 30자).

데이터베이스 트랜잭션 확인 - 계속

- Tran index 의미

```
$ $ 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           3, 2, 1
e5899a1b76253  update ta set a = 5 where a > 0
------------------------------------------------------------------------------------------------------------------------
```

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

데이터베이스 트랜잭션 확인 - 계속

- 트랜잭션 확인

```
$ $ 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(ACTIVE)       public      test-server      1682     broker1_cub_cas_2    0.00        0.00              -1
*** empty ***
3(ACTIVE)       public      test-server      1683     broker1_cub_cas_3    0.00        0.00              -1
*** empty ***
4(ACTIVE)       public      test-server      1684     broker1_cub_cas_4    1.80        1.80           3, 2, 1
e5899a1b76253   update ta set a = 5 where a > 0
5(ACTIVE)       public      test-server      1685     broker1_cub_cas_5    0.00        0.00              -1
*** empty ***
------------------------------------------------------------------------------------------------------------------------

SQL_ID: e5899a1b76253
Tran index : 4
update ta set a = 5 where a > 0
```

  - 위의 예는 3개의 트랜잭션이 각각 INSERT문을 실행 중일 때 또 다른 트랜잭션에서 UPDATE문의 실행을 시도한다. 위에서 Tran index가 4인 update문은 3,2,1(Wait for lock holder)번의 트랜잭션이 종료되기를 대기하고 있다.

  - 화면에 출력되는 질의문(SQL Text)은 질의 계획 캐시에 저장되어 있는 것으로, INSERT 문은 질의 계획 캐시에 저장되지 않으므로 cubrid tranlist에 출력되지 않는다.

Broker 상태 확인

- 명령어 : broker status
  - Broker monitoring 명령어
  - 여러 옵션을 같이 사용할 수 있다.

```
$ cubrid broker status [options]
```

옵션	설명
broker_name	지정한 브로커에 관한 상태 정보를 출력한다. 옵션이 지정되지 않으면 전체 브로커의 상태 정보를 출력한다
-b	응용 서버에 관한 정보는 포함되지 않고, 브로커에 관한 상태 정보만 출력한다.
-q	작업 큐에 대기 중인 작업을 출력한다
-s	브로커에 관한 상태 정보를 지정된 시간마다 주기적으로 출력한다, 기본값은 1초이다. q를 입력하면 명령 프롬프트로 복귀한다.
-I	옵션은 -f 옵션과만 함께 쓰이며, 클라이언트 Waiting/Busy 상태인 CAS의 개수를 출력할 때 누적 주기(단위: 초)를 지정하기 위해 사용한다. -I SECOND 옵션을 생략하면 기본값은 1초이다.
-f	브로커가 접속한 DB 및 호스트 정보를 출력한다.

Broker 상태 확인 - 계속

- broker 상태 확인

```
$ cubrid broker status -s 1 query

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

  - ID : 브로커 내에서 순착적으로 부여한 CAS의 일련 번호
  - PID : 브로커 내 CAS 프로세스 ID
  - QPS : 초당 처리된 질의의 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: 질의수행 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

Broker 상태 확인 - 계속

- broker 단위 별 상태 확인

```
$ 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 질의의 수 / LONG_QUERY_TIME 파라미터의 값
  - ERR-Q : 에러가 발생한 질의의 수
  - UNIQUE-ERR-Q: Unique 에러가 발생한 질의의 개수로 -b -s 옵션 지정한 초 동안 갱신
  - #CONNECT: 브로커 시작 후 응용 클라이언트가 CAS에 접속한 회수

Broker 상태 확인 - 계속

- broker 접속 정보 및 시간 확인

```
$ 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 51168 IDLE      2015/03/11 16:23:42  demodb  localhost  2015/03/11 16:23:40  10.0.1.101

  9.2.13.0003        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

CUBRID

Broker 상태 확인 - 계속

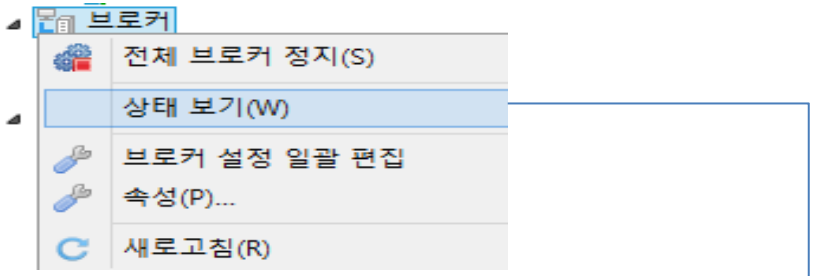
- AS(T W B Ns-W Ns-B), CANCELED 정보를 추가로 출력

```
// 브로커 상태 정보 실행 시 -f 옵션 추가. -I 옵션으로 N초 동안의 Ns-W, Ns-B를 출력하도록 초를 설정

$ cubrid broker status -b -f -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
  - AS(Ns-B): N초 동안 클라이언트 수행(Busy) 상태였던 CAS의 개수
  - CANCELED: 브로커 시작 이후 사용자 인터럽트로 인해 취소된 질의의 개수 (-I N 옵션과 함께 사용하면 N초 동안 누적된 개수)

Broker 상태 확인 - 계속

- CUBRID Manager에서 Broker상태 확인
  - CUBRID Manager Client에서는 아래와 같은 형식으로 Broker 상태를 확인 할 수 있다.
  - 각 메뉴 값은 command 명령어에 의해 보여졌던 결과 값과 동일하다.



브로커 상태 - 전체@localhost:8001

NAME	STATUS	PID	PORT	AS	JQ	REQ	TPS	QPS
query_editor	ON	8304	30000	5	0	46	15	5
broker1	ON	8324	33000	5	0	0	0	0

Broker SQL 로그 확인

- 수행된 트랜잭션 기반으로 SQL 별 로그 기록
  - SQL별, 트랜잭션 별 수행시간 기록
  - SQL 로그 분석을 통한 처리 성능 분석
  - $CUBRID/log/broker/sql_log/<broker 이름>_<cas ID>.sql.log

```
1번)02/04 13:45:17.687  2번)(39) prepare 0 3번)select * from unique_tbl
02/04 13:45:17.687 (39) 4번)prepare srv_h_id 1 (PC)
02/04 13:45:17.687 (39) 5번)execute srv_h_id 1 select * from unique_tbl
02/04 13:45:17.687 (39) 6번)execute 0 tuple 1 time 0.000
02/04 13:45:17.687 (0) 7번)auto_commit
02/04 13:45:17.687 (0) auto_commit 0
*** 0.000

02/04 13:45:17.687 (38) prepare 0 insert into unique_tbl values (1)
02/04 13:45:17.687 (38) prepare srv_h_id 1
02/04 13:45:17.687 (38) execute srv_h_id 1 insert into unique_tbl values (1)
02/04 13:45:17.687 (38) execute 8번)error:-670 tuple 0 time 0.000, EID = 39
02/04 13:45:17.687 (0)  auto_rollback
02/04 13:45:17.687 (0) 9번)auto_rollback 0
*** 0.000
```

1번) 02/04 13:45:17 응용 클라이언트의 요청 시각

2번) (39) SQL 문 그룹의 시퀀스 번호
- prepare 0 : prepared statement인지 여부

3번) Select * from unique_tbl 요청한 SQL

4번) prepare srv_h_id 1(PC) : 플랜 캐시에 저장되어 있는 내용을 사용

5번) execute srv_h_id 1 select * ... : 실행 SQL 문.
- Statement pooling한 경우는 WHERE 절의 binding 변수가 ?로 표시된다.

6번) execute 0 tuple 1 time 0.000
- 1개의 row가 실행되고, 소요 시간은 0.000초

7번) auto_commit: 자동으로 커밋

8번) 두번째 insert 질의는 : -670 오류(Unique)

9번) auto_rollback 0은 에러가 없이 트랜잭션이 완료

CUBRID

에러 로그

- 데이터베이스 서비스 중 또는 구동/종료 등 DB 작동 관련 발생한 서버 에러 로그 기록
  - 장애 원인 파악
  - $CUBRID/log/server/<db 이름>_<yyyymmdd날짜>_<server ID>.err

```
Time: 11/29/11 13:54:06.263 - ERROR *** ERROR CODE = -116, Tran = 0
Database “edudb" is unknown, or the file "databases.txt" cannot be accessed.
```

- SQL 수행중 발생한 Broker 에러 로그 기록
  - SQL 로그와 같이 사용하면 오류 파악이 용이하다.
  - $CUBRID/log/broker/error_log/<broker 이름>_<cas ID>.err

```
Time: 02/04/10 13:45:17.687 - SYNTAX ERROR *** ERROR CODE = -493, Tran = 1, EID = 38
Syntax: Unknown class "unknown_tbl". select * from unknown_tbl
```

9. 환경 설정

환경파일

- 데이터베이스 환경설정
  - 데이터베이스 환경설정은 $CUBRID/conf/cubrid.conf 파일이다.
  - cubrid.conf 파일은 CUBRID 데이터베이스 시스템 파라미터의 설정 값을 저장하는 파일이다.
  - 환경설정 대/소문자 구분하지 않지만 일반적으로 소문자로 설정한다.
  - 데이터베이스 별로 다를 수 있는 내용들은 데이터베이스 별로 별도로 지정하는 것이 좋다.

```
[commom]
max_clients=50
# 모든 데이터베이스 공통 설정
[@demodb]
max_clients=100
# demodb를 위한 설정
```

  - 서버와 클라이언트 파라미터로 나뉘며 파라미터 변경할 경우 DB나 해당 프로세스를 재 구동 해야 한다.
    - SQL을 사용하여 클라이언트 파라미터 변경 시 재 구동이 필요 없다.
  - 파라미터 설정 구문 규칙
    - 대/소문자를 구분 없다.
    - 파라미터 이름과 설정값은 동일한 라인에 입력되어야 한다.
    - 등호 기호(=)를 사용하며, 양 옆에는 공백문자를 사용할 수 있다.
    - 파라미터 값이 문자열인 경우 따옴표 없이 문자열만 입력, 해당 문자열에 공백 문자가 포함된 경우에는 따옴표를 사용한다.

데이터베이스 환경 설정

- 접속 관련 파라미터
  - cubrid_port_id
    - 마스터 프로세스가 사용하는 포트를 설정하기 위한 파라미터로 기본값은 1523이다.
    - 방화벽에 의해 1523 포트가 차단된 경우에는 마스터 프로세스가 정상적으로 구동할 수 없어 DB와 연결이 불가능하다.
  - max_clients
    - 데이터베이스 서버에 동시 연결을 허용하는 클라이언트(일반적으로 브로커 용용 서버(CAS))의 최대 개수를 지정하기 위한 파라미터다. 이 파라미터의 기본값은 100이다
    - 데이터베이스 서버 프로세스 하나 당 동시에 접속할 수 있는 클라이언트의 최대 개수를 의미한다.
    - max_clients 파라미터는 cubrid_broker.conf 환경파일에 MAX_NUM_APPL_SERVER 값보다 10~30개 이상 크게 설정하는 것을 권장한다.

- 메모리 관련 파라미터
  - data_buffer_size
    - 데이터베이스 서버가 메모리 내에 캐시하고 있는 데이터 캐쉬 크기이며 기본값은 512M가 이다.
    - 실제 데이터베이스의 크기 및 실제 메모리의 크기, 기타 다른 프로세스의 개수 및 크기를 고려해서 할당해야 한다.
    - 이 파라미터의 값을 너무 크게 설정하면 과도하게 시스템 메모리가 점유되므로 운영체제에 의해 버퍼 풀이 스와핑(swapping)되는 현상이 발생할 수 있다
  - sort_buffer_size
    - 정렬을 수행하는 질의에서 사용되는 버퍼의 크기를 설정하기 위한 파라미터로 Active client request 마다 하나의 sort buffer가 할당되고 기본값은 2M가 이다.
    - 서버는 각 클라이언트의 정렬 요청마다 하나의 정렬 버퍼를 할당하며, 정렬을 완료한 후에는 할당되었던 버퍼 메모리를 해제한다.
    - 정렬을 수행하는 질의로는 SELECT 정렬 질의 뿐만 아니라 인덱스 생성 질의도 포함된다

데이터베이스 환경 설정 - 계속

- 디스크 관련 파라미터
  - temp_file_max_size_in_pages
    - 질의 수행 시 필요한 임시 볼륨(temp volume)은 일시적 임시 볼륨과 영구적 임시 볼륨으로 구분되는데, 이 파라미터의 값은 일시적 임시 볼륨에만 적용된다.
    - 이 파라미터 기본값은 -1이며 무한대로 임시 볼륨이 생성되어 디스크의 여유 공간이 부족해져 시스템 운영에 문제가 발생할 수 있다.
    - 임시 볼륨은 큰 크기의 임시 공간이 필요한 질의를 수행하면서 일시적 임시 볼륨이 기대 이상으로 증가될 수 있다.
    - 예상하는 영구적 임시 temp 볼륨을 미리 확보하고, 하나의 질의가 수행될 때 일시적 임시 볼륨에서 사용되는 공간의 최대 크기를 제한하는 것이 좋다.
    - 이 파라미터를 0으로 설정되면 일시적 임시 볼륨이 생성되지 않는다.
  - temp_volume_path
    - 볼륨(temporary temp volume)의 디렉터리를 지정하는 파라미터로 기본값은 데이터베이스 생성 시에 설정된 볼륨 위치이다.

- 오류 메시지 관련 파라미터
  - error_log
    - 데이터베이스 서버에 오류가 발생하는 경우 <database_name>_<date>_<time>.err 저장하고 $CUBRID/log/server 디렉터리에 저장된다.
  - error_log_size
    - error_log_size는 에러 로그 파일에서 기록되는 최대 라인 수를 지정하는 파라미터로 기본값은 약 8M이다.
    - 이 파라미터의 설정 값에 도달하면 <database_name>_<date>_<time>.err.bak 파일이 생성된다.
  - call_stack_dump_on_error
    - 데이터베이스 서버에서 오류가 발생했을 때 콜-스택을 덤프 할 지 결정하기 위한 파라미터로 DB 오류를 보다 자세하게 확인이 필요할 때 사용된다, 기본값은 no이다.
    - no로 설정되면 모든 오류에 대해서 콜-스택을 덤프 하지 않고 yes로 설정되면 모든 오류에 대해서 콜-스택을 덤프 한다.

CUBRID

데이터베이스 환경 설정 - 계속

- 동시성/잠금 파라미터
  - lock_timeout
    - 잠금 대기 시간을 지정하기 위한 클라이언트 파라미터로 지정된 시간 이내에 잠금이 허용되지 않으면 해당 트랜잭션이 취소되고 오류가 반환된다, s, min, h 단위를 지정할 수 있으며 예를 들어, 1s는 1초가 되며, 10s는 10초가 설정 된다.
    - 기본값인 -1로 설정하면 잠금이 허용될 때까지의 대기 시간이 무제한이고, 0으로 설정하면 잠금을 대기하지 않는다.
  - lock_escalation
    - 행에 대한 잠금이 테이블 잠금으로 확대되기 전에 개별 행에 허용되는 최대 잠금의 개수를 설정하기 위한 파라미터로 기본값은 100,000이다.
    - lock_escalation 파라미터의 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 고립수준을 1에서 6까지의 정수 값 또는 문자 스트링으로 설정한다.
    - SERIALIZABLE: 트랜잭션이 끝날 때 까지 접근 불가하다.
    - REPEATABLE: SELECT에서 S_LOCK이 트랜잭션 종료될 때 까지 계속 유지한다.
    - READ UNCOMMITTED: 완료되지 않은 트랜잭션에 Read를 허용한다.

설정 값	설 명
TRAN_SERIALIZABLE(6)	SERIALIZABLE
TRAN_REP_CLASS_REP_INSTANCE(5)	REPEATABLE READ CLASS with REPEATABLE READ INSTANCES
TRAN_REP_CLASS_COMMIT_INSTANCE TRAN_READ_COMMITTED(4)	REPEATABLE READ CLASS with READ COMMITTED INSTANCES
TRAN_READ_UNCOMMITTED TRAN_REP_CLASS_UNCOMMIT_INSTANCE(3)	**REPEATABLE READ CLASS with READ UNCOMMITTED INSTANCES**
TRAN_COMMIT_CLASS_COMMIT_INSTANCE(2)	READ COMMITTED CLASS with READ COMMITTED INSTANCES
TRAN_COMMIT_CLASS_UNCOMMIT_INSTANCE(1)	READ COMMITTED CLASS with READ UNCOMMITTED INSTANCES

데이터베이스 환경 설정

- 로깅 관련 파라미터
  - log_buffer_size
    - log_buffer_size는 메모리에 캐시 되는 로그 버퍼의 크기를 설정하는 파라미터로 기본값은 128 * log_page_size (log_page_size가 16K이면 2M) 이다.
    - 이 파라미터의 설정 값이 크면 데이터베이스 수정 연산이 많고, 길고 큰 트랜잭션이 많은 환경에서는 디스크 I/O가 감소되어 성능이 향상될 수 있다.
  - force_remove_log_archives
    - force_remove_log_archives는 log_max_archives로 지정한 개수의 최근 보관 로그(log archive) 파일을 제외한 나머지 파일의 삭제 허용 여부를 지정하는 파라미터로서, 기본값은 yes이다.
    - 파라미터 값을 yes로 설정하면, 아래 log_max_archives로 지정한 개수의 최근 보관 로그 파일을 제외한 나머지 파일이 삭제된다.
    - 파라미터 값을 no로 설정하면, 보관 로그 파일이 삭제되지 않지만, 예외적으로 ha_mode를 on으로 설정하면 HA 관련 프로세스에 필요한 보관 로그 파일과 log_max_archives로 지정한 개수의 최근 보관 로그 파일을 제외한 나머지 파일이 삭제된다.
  - log_max_archives
    - 보존할 보관 로그 파일의 최대 개수를 설정하는 파라미터로 최소값은 0이다
    - 이 파라미터는 force_remove_log_archives 의 설정에 따라 동작이 달라질 수 있다.
    - 예를 들어, cubrid.conf의 log_max_archives가 3이고 force_remove_log_archives가 yes이면, 최근 3개의 보관 로그 파일만 유지하고 네 번째 보관 로그가 생성될 때에는 이전에 생성된 보관 로그 파일을 자동으로 삭제한다.

데이터베이스 환경 설정 - 계속

- 질의 캐시 관련 파라미터
  - max_plan_cache_entries
    - 메모리에 캐시 하는 질의 실행 계획의 최대 개수를 설정하는 파라미터로 기본값은 1000개 이다.

- 트랜잭션 처리 관련 파라미터
  - async_commit
    - 비 동기 식 커밋 기능을 활성화시키는 파라미터로 기본값인 no이고, yes로 설정하면 비 동기식 커밋을 수행한다.
    - 비 동기 식 커밋은 로그 플러시를 백그라운드에서 수행하여 커밋 작업의 성능을 향상시키는 기능이다.
    - 로그 플러시가 수행되기 전에 데이터베이스 서버에 장애가 발생하면 이미 커밋 완료된 트랜잭션을 복구할 수 없어 데이터의 손실이 발생해 사용에 주의가 필요하다.
  - group_commit_interval_in_msecs
    - group_commit_interval_in_msecs은 그룹 커밋을 수행하는 간격을 밀리초(mesc) 단위로 지정하는 파라미터로 기본값인 0 으로 설정되면 그룹 커밋을 수행하지 않는다.
    - 그룹 커밋 이란 지정된 시간 동안 발생한 여러 번의 커밋을 그룹으로 취합하여 커밋 로그가 동시에 디스크에 플러시 되도록 하여 커밋 작업의 성능을 향상시키지만 상황에 따라 데이터의 손실이 발생할 수 있어 사용에 주의가 필요하다.

데이터베이스 환경 설정 - 계속

- 기타 파라미터
  - service
    - service는 CUBRID 서비스 시작 시 자동으로 시작하는 프로세스를 등록하는 파라미터로 server, broker, manager, heartbeat 의 네 종류 프로세스가 있다. 일반적으로 service=server, broker, manager와 같이 세 종류 프로세스를 등록한다, 만약 모든 DB가 HA 구성되어 있다면 service=heartbeat, broker, manager 설정하면 자동 시작된다.
  - server
    - Server는 CUBRID 서비스 시작 시 자동으로 시작하도록 하는 데이터베이스 서버 프로세스들의 이름을 등록하는 파라미터다. 해당 데이터베이스들의 이름을 쉼표(,)로 구분하여 나열한다.
  - Java_stored_procedure
    - Java_stored_procedure는 Java 가상 머신(Java Virtual Machine, JVM)을 실행하여 Java 저장 프로시저(Java stored procedure)를 사용하게 하기 위한 파라미터다.
    - 기본값인 no로 설정하며 JVM이 실행되지 않고, yes로 설정하면 JVM이 실행되어 Java 저장 프로시저(Java stored procedure)를 사용할 수 있다. 따라서, CUBRID에서 Java 저장 프로시저를 사용할 계획이 있는 경우에는 반드시 파라미터를 yes로 설정해야 한다.
  - block_ddl_statement
    - 데이터 정의 문(Data Definition Language, DDL)을 제한하는 파라미터로 기본값은 no로 설정하지 않는다.
  - block_nowhere_statement
    - UPDATE/DELETE문에 조건(Where)이 없는 경우 질의를 수행하지 않는 파라미터로 기본값은 no로 설정하지 않는다.

broker 환경 설정

- 환경 설정 변경
  - 설정 파일 : $CUBRID/conf/cubrid_broker.conf
  - 편집기를 통하여 수정가능하며, 브로커 재 구동될 경우 설정 적용한다.
  - 브로커 재 구동 없이 설정 변경이 가능한 파라미터가 있으며 아래 broker_changer 명령어를 이용한다.

```
$ broker_changer  <br-name>  <conf-name>  <conf-value>
$ broker_changer broker1 sql_log on
OK
```

- 동적 변경 가능 환경변수
  - ACCESS_MODE, ACCESS_LOG, ACCESS_LOG_MAX_SIZE, SESSION_TIMEOUT
  - LOG_DIR, SLOW_LOG, SLOW_LOG_DIR, SQL_LOG_MAX_SIZE, SQL_LOG, ERROR_LOG_DIR
  - 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, STATEMENT_POOLING,
  - CONNECT_ORDER, KEEP_CONNECTION, PREFERRED_HOSTS, RECONNECT_TIME

- 환경변수 및 설정 값이 잘못되어 있을 경우 재 구동 시 오류가 발생하며 구동되지 않는다.

broker 환경 설정 - 계속

- 브로커 설정 정보 확인
  - **cubrid broker info**는 현재 "실행 중"인 브로커 파라미터의 설정 정보(cubrid_broker.conf)를 출력한다.

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

[%query_editor]
SERVICE                 =ON
APPL_SERVER             =CAS
MIN_NUM_APPL_SERVER     =5
MAX_NUM_APPL_SERVER     =40
APPL_SERVER_SHM_ID      =30000
APPL_SERVER_MAX_SIZE    =0
SESSION_TIMEOUT         =300
LOG_DIR                 =log/broker/sql_log
SLOW_LOG_DIR            =log/broker/sql_log/
ERROR_LOG_DIR           =log/broker/error_log
LOG_BACKUP              =OFF
SOURCE_ENV              =
SQL_LOG                 =ALL
SLOW_LOG                =ON
:
```

broker 환경 설정 - 계속

- 브로커 접속 파라미터
  - ACCESS_MODE
    - 기본값은 RW로 읽기/쓰기 모두 가능한 상태이며 만약 access_mode가 RO 또는 SO 경우 쓰기가 안 된다.
  - BROKER_PORT
    - 브로커의 포트 번호를 지정하기 위한 파라미터로 시스템 내에서 유일한 값이면서 65,535 이하의 값이어야 한다
    - 새로운 브로커를 생성할 경우 BROKER_PORT와 APPL_SERVER_SHM_ID 같은 값을 입력한다.
  - APPL_SERVER_SHM_ID
    - CAS가 이용하는 공유 메모리 ID를 지정하기 위한 파라미터로 시스템 내에서 유일한 값이어야 한다. 기본값은 해당 브로커의 포트와 동일한 값이다.
  - APPL_SERVER_PORT
    - Windows 운영체제에서만 추가하는 파라미터로 응용 클라이언트와 통신하는 CAS의 통신 포트를 지정하는 파라미터다. Linux 운영체제에서는 응용 클라이언트와 CAS가 통신하기 위해 유닉스 도메인 소켓(Unix domain socket)을 사용하므로, APPL_SERVER_PORT 가 사용되지 않는다,
    - Windows 운영체제에서 응용 클라이언트와 CUBRID 브로커 사이에 방화벽이 존재한다면 반드시 BROKER_PORT 및 APPL_SERVER_PORT 에서 설정된 통신 포트를 열어야 한다.(MAX_NUM_APPL_SERVER 값과 동일하게 설정)

- 브로커 접속 제한 파라미터
  - ACCESS_CONTROL
    - 브로커에 접속하는 응용 클라이언트를 제한하기 위한 파라미터다. 기본값은 OFF이다.
  - ACCESS_CONTROL_FILE
    - 브로커에 접속을 허용하는 데이터베이스 이름, 데이터베이스 사용자 ID, IP 목록을 저장한 파일 이름을 지정하는 파라미터다.
    - IP 목록은 하나의 브로커 내에서 <*db_name*>:<*db_user*> 별로 최대 256 라인까지 작성될 수 있다.

broker 환경 설정 - 계속

- 브로커 컨넥션(CAS) 파라미터
  - MAX_NUM_APPL_SERVER
    - 동시 접속할 수 있는 CAS의 최대 개수를 설정하는 파라미터로, 기본값은 40이다.
  - MIN_NUM_APPL_SERVER
    - 기본적으로 대기하고 있는 CAS 프로세스의 최소 개수를 설정하는 파라미터로, 기본값은 5이다.
  - TIME_TO_KILL
    - 자동 추가된 CAS 중 유휴 상태의 CAS를 제거하기 위한 기준 시간을 설정하는 파라미터다, 기본값은 120(초)이다.
    - STATUS가 IDEL 상태로 TIME_TO_KILL 시간 이상 유지되면 해당 CAS를 제거한다.

- 브로커 트랜잭션 및 질의관련
  - LONG_QUERY_TIME
    - 장기 실행 질의로 판단될 질의 실행 시간을 설정하는 파라미터로 기본값은 60 (초)이며 설정 값을 초과한 질의는 $CUBRID/log/broker/sql_log 위치에 slow 로그로 SQL이 남겨진다..
  - LONG_TRANSACTION_TIME
    - 장기 실행 트랜잭션(long-duration transaction)으로 판단될 트랜잭션의 실행 시간을 설정하는 파라미터로 기본값은 60 (초)이며 설정 값을 초과한 질의는 $CUBRID/log/broker/sql_log 위치에 slow 로그로 SQL이 남겨진다.
  - MAX_QUERY_TIMEOUT
    - 질의 수행의 타임아웃을 설정하는 브로커 파라미터로, 질의 수행을 시작한 후 지정 시간을 초과하면 수행하던 질의를 멈추고 롤백하며 기본값은 0이며, 무한 대기를 의미한다.
  - SESSION_TIMEOUT
    - CAS가 트랜잭션 시작 이후 커밋 혹은 롤백 하지 않은 채로 아무런 요청이 없는 상태에서 이 파라미터가 설정한 시간을 초과하면 해당 응용 클라이언트와 CAS 간의 접속이 종료되며 기본값은 300 (초)이다
  - 트랜잭션 및 질의 설정 시 참조사항
    - 값 뒤에 ms, s, min, h의 단위 지정이 가능하며, 각각 milliseconds, seconds, minutes, hours를 의미한다

broker 환경 설정 - 계속

- 브로커 로그 파라미터
  - LOG_DIR
    - SQL 로그가 저장되는 디렉터리를 지정하는 파라미터로, 기본값은 log/broker/sql_log이다.
  - SLOW_LOG
    - SLOW SQL 로깅 여부를 지정하는 파라미터다. 기본값은 ON이다. 이 값이 ON이면 LONG_QUERY_TIME 시간을 초과한 장기 실행(long-duration query) 질의문 또는 에러가 발생한 질의 문이 SLOW SQL 로그 파일에 저장된다.
  - SLOW_LOG_DIR
    - SLOW SQL 로그 파일이 생성되는 디렉터리를 지정한다. 기본값은 log/broker/sql_log이다.
  - SQL_LOG
    - 기본값은 ON이며 모든 SQL로그를 남기며 OFF, ERROR, NOTICE, TIMEOUT 파라미터가 있다.
  - SQL_LOG_MAX_SIZE
    - SQL_LOG_MAX_SIZE는 SQL 로그 파일과 SLOW SQL 로그 파일의 최대 크기를 지정하는 파라미터다. 값 뒤에 B, K, M, G로 단위를 붙일 수 있으며, 각각 Bytes, Kilobytes, Megabytes, Gigabytes를 의미한다. 단위 생략 시 K로 지정된다. 기본값은 10,000 (KB)이다.
    - SQL_LOG 파라미터가 ON 으로 설정된 경우에 생성되는 SQL 로그 파일의 크기가 파라미터의 설정 값에 도달하면 broker_name_id.sql.log.bak이 생성된다.
  - ACCESS_LOG
    - 브로커의 접속 로그를 저장할 것인지 지정하는 파라미터로 기본값은 OFF이다, 브로커 접속 로그 파일명은 broker_name.access이고, $CUBRID/log/broker 디렉터리에 저장된다.
  - ACCESS_LOG_DIR
    - 브로커 접속 로그(ACCESS_LOG) 파일이 생성되는 디렉터리를 지정한다. 기본값은 log/broker이다.
  - ACCESS_LOG_MAX_SIZE
    - 브로커 접속 로그(ACCESS_LOG) 파일의 최대 크기를 지정하며, 기본값은 10M 이고, 최대 2G로 설정이 가능하다.
  - ERROR_LOG_DIR
    - log/broker/error_log이다. 브로커 에러 로그 파일명은 broker_name_id.err이다.

broker 추가

- 브로커(BROKER) 추가
  - 특정 업무로의 사용을 위해서, 혹은 다른 database 로의 서비스를 위해서 broker 를 추가할 수 있다.
  - 특히 CUBRID BROKER는 connection pooling 개념을 사용하므로 여러 개의 database 를 대상으로 서비스를 한다면 database 별로 별로의 broker 를 사용하여야 connection에 따른 overhead 를 줄일 수 있다.
    - 추가 후 재 구동하여 추가된 브로커를 구동시킬 수 있다.
  - broker 추가를 위해서는 $CUBRID/conf/cubrid_broker.conf 을 직접 편집한다.
    - 기존 내용을 그대로 복사하여 마지막에 추가한다.
    - Broker 이름을 변경한다. BROKER_PORT 와 PPL_SERVER_SHM_ID 를 변경한다

```
# 기존 내용
[%BROKER1]
SERVICE                 =ON
BROKER_PORT             =33000
MIN_NUM_APPL_SERVER     =5
MAX_NUM_APPL_SERVER     =40
APPL_SERVER_SHM_ID      =33000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR           =log/broker/error_log
SQL_LOG                 =ON
TIME_TO_KILL            =120
SESSION_TIMEOUT         =300
KEEP_CONNECTION         =AUTO
CCI_DEFAULT_AUTOCOMMIT  =ON
```

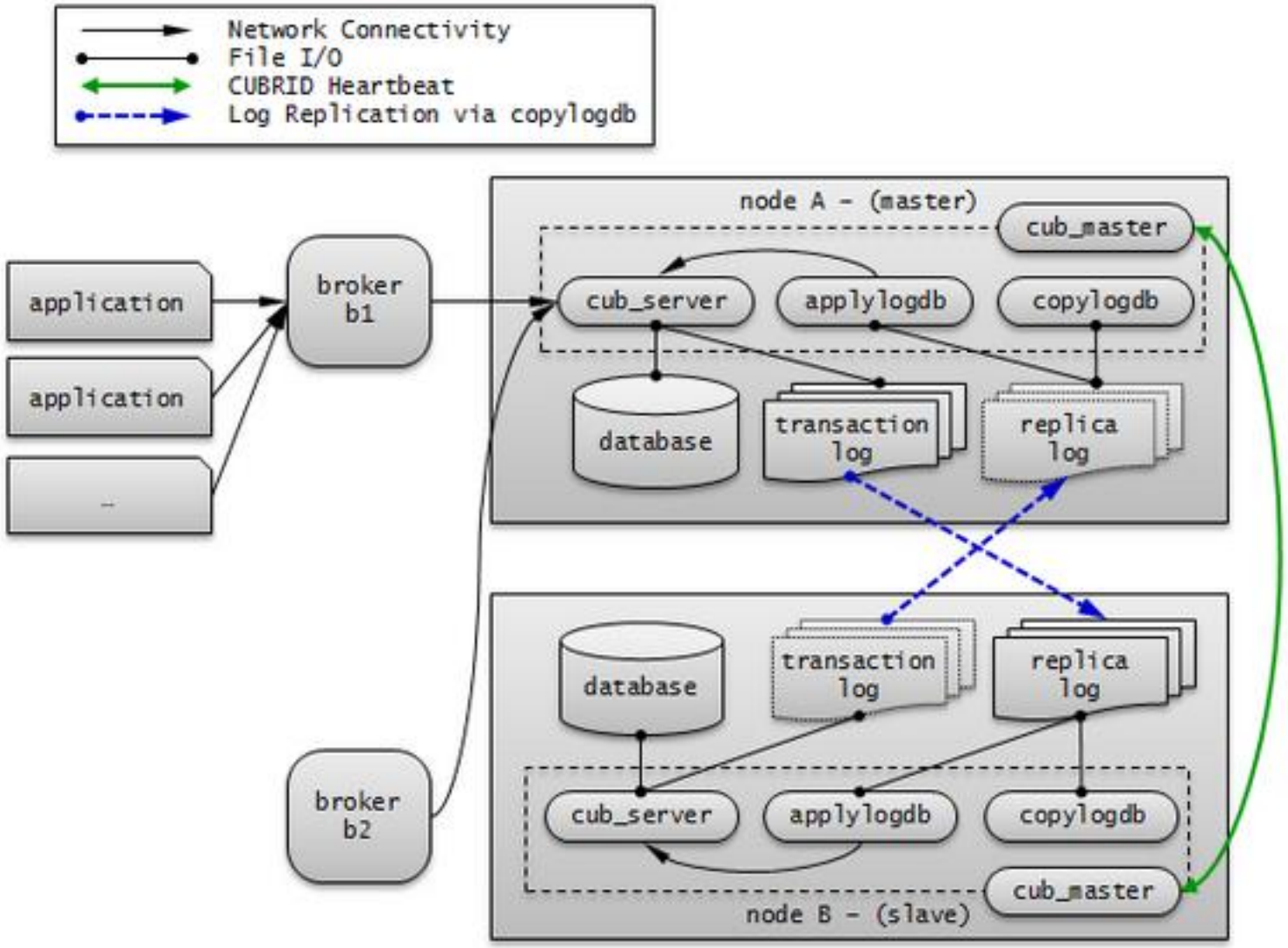
```
# 추가할 내용(반드시 수정해야할 부분)
[%BROKER_COPY]
SERVICE                 =ON
BROKER_PORT             =51000
MIN_NUM_APPL_SERVER     =5
MAX_NUM_APPL_SERVER     =40
APPL_SERVER_SHM_ID      =51000
LOG_DIR                 =log/broker/sql_log
ERROR_LOG_DIR           =log/broker/error_log
SQL_LOG                 =ON
TIME_TO_KILL            =120
SESSION_TIMEOUT         =300
KEEP_CONNECTION         =AUTO
CCI_DEFAULT_AUTOCOMMIT  =ON
```

10. CUBRID HA

CUBRID HA 구조

- 개요
  - High Availability(고가용성) 기능을 CUBRID에 적용한 것이 CUBRID HA 기능이다.
  - CUBRID HA기능으로 구성된 DB서버는 데이터베이스를 항상 동기화된 상태로 유지하여 서비스를 제공한다.
  - CUBRID HA는 Active 서버에 예상치 못한 장애가 발생하면 자동으로 Fail-over되서 Stand-by 서버로 서비스를 수행하도록해 서비스 중단시간을 최소화한다.
    - CUBRID HA 기능은 Active 서버와 Stand-by 서버에 항상 동기화된 데이터를 유지한다.
    - 만약, 서비스를 제공 중이던 마스터 노드(master node)에 예상치 못한 장애가 발생하여 액티브 서버가 정상적으로 동작하지 못하면 슬레이브 노드(slave node)의 스탠바이 서버가 액티브 서버를 대신하여 중단 없는 서비스를 제공할 수 있다.
    - CUBRID HA 기능은 시스템과 CUBRID의 상태를 실시간으로 감시하고 장애가 발생하면 자동 failover를 수행하기 위해 heartbeat 메시지를 사용한다.
  - CUBRID의 HA 기능은 shared-nothing 구조이며, Active 서버에서 Stand-by 서버로 데이터를 동기화하기 위해 다음 두 단계를 수행한다.
    - 트랜잭션 로그 다중화: 액티브 서버에서 생성되는 트랜잭션 로그를 실시간으로 다른 노드에 복제한다.
    - 트랜잭션 로그 반영: 실시간으로 복제된 트랜잭션 로그를 분석하여 스탠바이 서버에 데이터를 반영한다.

CUBRID HA 구조도



CUBRID

CUBRID HA 기본개념

- 노드와 그룹
  - 노드는 CUBRID HA를 구성하는 논리적인 단위이다.
    - 마스터 노드: 복제의 대상이 되는 노드로, 액티브 서버를 사용한 읽기, 쓰기 등 모든 서비스를 제공한다.
    - 슬레이브 노드: 마스터 노드와 동일한 내용을 갖는 노드로, 마스터 노드의 변경이 자동으로 반영된다. 스탠바이 서버를 사용한 읽기 서비스를 제공하며 마스터 노드 장애 시 failover가 일어난다.

- 프로세스
  - CUBRID HA 노드는 하나의 마스터 프로세스(cub_master), 하나 이상의 데이터베이스 서버 프로세스(cub_server), 하나 이상의 복제 로그 복사 프로세스(copylogdb), 하나 이상의 복제 로그 반영 프로세스(applylogdb)로 이루어져 있다.
  - 하나의 데이터베이스를 설정하면 데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스가 구동된다.
    - 마스터 프로세스(cub_master): heartbeat 메시지를 주고 받으며 CUBRID HA 내부 관리 프로세스들을 제어한다.
    - 데이터베이스 서버 프로세스(cub_server): 사용자에게 읽기, 쓰기 등의 서비스를 제공한다.
    - 복제 로그 복사 프로세스(copylogdb): 그룹 내의 모든 트랜잭션 로그를 복사한다.
    - 복제 로그 반영 프로세스(applylogdb): 복제 로그 복사 프로세스에 의해 복사된 로그를 노드에 반영한다.

- 서버
  - 서버란 데이터베이스 서버 프로세스를 논리적으로 표현하는 단어로, 상태에 따라 액티브 서버(active server), 스탠바이 서버(standby server)로 나눈다.
    - 액티브 서버: 마스터 노드에 속하는 서버로, active 상태이다. 액티브 서버는 사용자에게 읽기, 쓰기 등 모든 서비스를 제공한다.
    - 스탠바이 서버: 마스터 노드 외의 노드에 속하는 서버로, standby 상태이다. 스탠바이 서버는 사용자에게 읽기 서비스만을 제공한다.

CUBRID HA 기본개념 - 계속

- Heartbeat 메시지
  - HA 기능을 제공하기 위한 핵심 구성 요소로, 마스터 노드, 슬레이브 노드, 레플리카 노드가 다른 노드의 상태를 감시하기 위해 주고 받는 메시지이다.
  - 기본 59901포트의 UDP통신으로 heartbeat check를 수행한다.

- Fail-over와 Fail-back
  - Fail-over란, 마스터 노드에 장애가 발생하여 서비스를 제공할 수 없는 상태가 되면 우선순위가 가장 높은 슬레이브 노드가 자동으로 마스터 노드가 되는 것이다.
  - Fail-back은 마스터 노드였던 장애 노드가 복구되면 마스터 노드가 되는 것이며, 현재 CUBRID HA는 서버의 자동 Fail-back을 지원하지 않는다.

- Broker 모드
  - Broker는 서버에 Read Write, Read Only 모드 중 한 가지로 접속할 수 있으며, 사용자가 Broker 모드를 설정할 수 있다.
  - databases.txt에 설정된 호스트에 순차적으로 연결을 시도한다.
    - Read Write : 읽기, 쓰기 서비스를 제공하는 Broker이다. 이 Broker는 일반적으로 액티브 서버에 연결하며, 연결 가능한 액티브 서버가 없으면 스탠바이 서버에 연결한다.
    - Read Only : 읽기 서비스를 제공하는 Broker이다. 이 Broker는 가능한 스탠바이 서버에 연결하며, 스탠바이 서버가 없으면 액티브 서버에 연결한다.

CUBRID HA 환경설정

- cubrid.conf
  - $CUBRID/conf 디렉터리에 위치
  - ha_mode
    - CUBRID HA 기능을 설정하는 파라미터다. 기본값은 off이다.
    - off : CUBIRD HA 기능을 사용하지 않는다.
    - on : CUBRID HA 기능을 사용하며, 해당 노드는 failover의 대상이 된다. 마스터, 슬레이브 노드에 설정.
  - log_max_archives
    - 보존할 보관 로그 파일의 최소 개수를 설정하는 파라미터다. CUBRID가 설치된 영역의 가용 디스크 공간과 백업 정책 (1 level 백업의 경우 보관 로그 파일 삭제 불가), 1일 보관 로그 생성 개수에 따라 적절하게 적용
  - force_remove_log_archives
    - log_max_archives로 지정한 개수의 최근 보관 로그(log archive) 파일을 제외한 나머지 파일의 삭제 허용 여부를 지정하는 파라미터로서, ha_mode가 on인 경우 no로 설정하여 HA 관련 프로세스에 필요한 보관 로그 파일이 삭제되지 않도록 설정한다.
  - 설정 예시

```
ha_mode=on
log_max_archives=20
force_remove_log_archives=no
```

CUBRID HA 환경설정 - 계속

- cubrid_ha.conf
  - $CUBRID/conf 디렉터리에 위치, CUBRID의 HA의 전반적인 설정 정보를 담고 있다.
  - ha_node_list
    - CUBRID HA 그룹 내에서 사용할 그룹 이름과 failover의 대상이 되는 멤버 노드들의 호스트 이름을 명시한다.
    - 이 파라미터에서 명시한 멤버 노드들의 호스트 이름 및 해당 노드의 호스트 이름은 반드시 /etc/hosts에 등록되어 있어야 한다.
  - 설정 예시

```
[common]
ha_node_list=cubrid@Master_node:Slave_node
ha_db_list=demodb
ha_ping_hosts=192.168.0.xxx
```

CUBRID HA 환경설정 - 계속

- databases.txt
  - $CUBRID_DATABASES 디렉터리에 위치하며, db_hosts 값을 설정하여 Broker가 접속하는 서버의 순서를 결정할 수 있다.
  - 설정예시

```
#db-name        vol-path              db-host                  log-path              lob-base-path
demodb    /home/cubrid/DB/   Master_node:Slave_node   /home/cubrid/DB/log   file:/home/cubrid/DB/lob
```

- jdbc 설정
  - JDBC에서 CUBRID HA 기능을 사용하려면 Broker에 장애가 발생했을 때 연결할 Broker의 연결 정보를 연결 URL에 추가로 지정해야 한다. CUBRID HA를 위해 지정되는 속성은 장애가 발생했을 때 연결할 하나 이상의 Broker 노드 정보인 althosts와 Broker의 장애 복구 후 재 연결을 시도하는 주기인 rctime이다.
  - 설정 예시

```
Connection connection =
DriverManager.getConnection("jdbc:cubrid:primary_broker:33000:demodb:::?charset=utf-
8&althosts=secondary_broker:33000&rctime=600", "dba", "");
```

CUBRID HA 환경설정 - 계속

- cubrid_broker.conf
  - $CUBRID/conf 디렉터리 위치에 브로커 환경파일(cubrid_broker.conf)로 브로커 모드를 설정한다.
  - ACCESS_MODE : Broker의 모드를 설정한다. 기본값은 RW이다.
    - RW(Read Write), RO(Read Only)를 값으로 설정할 수 있다.
  - Read Write: 읽기, 쓰기 서비스를 제공하는 브로커이다. 이 브로커는 일반적으로 액티브 서버에 연결하며, 연결 가능한 액티브 서버가 없으면 스탠바이 서버에 연결한다.
    - 일시적으로 스탠바이 서버와 연결되면 트랜잭션이 끝날 때마다 스탠바이 서버와 연결을 끊고, 다음 트랜잭션이 시작되면 다시 액티브 서버와 연결을 시도한다.(Connection Pooling을 사용할 경우 유지 됨)
    - 스탠바이 서버와 연결되면 읽기 서비스만 가능하며, 쓰기 요청에 대해서는 서버에서 오류가 발생한다.
  - Read Only: 읽기 서비스를 제공하는 브로커이다. 이 브로커는 가능한 스탠바이 서버에 연결하며, 스탠바이 서버가 없으면 액티브 서버에 연결한다.
    - 액티브 서버와 연결된 후에는 스탠바이 서버가 있어도 연결은 끊기지 않으며, cubrid_broker reset 명령을 실행해야만 기존 연결을 끊고 새롭게 스탠바이 서버에 연결할 수 있다.
    - Read Only 브로커에 쓰기 요청이 전달되면 브로커에서 오류가 발생하므로, 액티브 서버와 연결되어도 읽기 서비스만 가능하다.

CUBRID HA 설정 예제

- HA 기본 구성
  - Master/Slave 1:1 구성의 예제 방법으로 CUBRID HA 고유의 기능인 장애 시 무 중단(nonstop) 서비스 기능에 초점을 맞춘 구성이다.
  - 설정 예시
    - CUBRID HA 설정 방법
      - Master와 Slave 모두 동일하게 설정한다
      - cubrid.conf 파일의 마지막 라인에 HA 파라미터를 등록/설정한다

```
ha_mode=on
log_max_archives=20
force_remove_log_archives=no
```

      - cubrid_ha.conf 파일에 주석을 제거하고 계정명과 호스트 명 그리고 DB 명을 설정한다.

```
ha_port_id=59901
ha_node_list=cubrid@db_server01:db_server02
ha_db_list=testdb
ha_ping_hosts=192.168.0.233
```

      - databases.txt 파일에 HA구성한 DB명과 호스트 명(마스터/슬레이브)을 설정한다.

```
#db-name        vol-path        db-host                    log-path
testdb          /DB/testdb      db_server01:db_server02    /DB/testdb/log
```

CUBRID HA 설정 예제 - 계속

- cubrid_broker.conf 파일의 설정 예
  - Slave DB를 Read 서비스 활용할 경우 ACCESS_MODE=RO 설정한다.

```
[%broker1]
BROKER_PORT             =33000
... ...
ACCESS_MODE              =RW
[%broker2]
BROKER_PORT             =34000
... ...
ACCESS_MODE              =RO
```

- /etc/hosts 설정
  - /etc/hosts 정보에 Master/Slave IP와 hostname을 등록한다.

```
#> vi /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1               localhost.localdomain   localhost       newTest1
192.168.0.131             db_server01
192.168.0.132             db_server02
```

- JDBC 설정
  - Master와 Slave 서버 IP, Port, DB정보를 설정한다.

```
Connection connection = DriverManager.getConnection("jdbc:cubrid:
192.168.0.131:33000:testdb:::?charset=utf-8&althosts=192.168.0.132:33000&rctime=600",
"dba", "");
```


CUBRID

CUBRID HA 구동 및 모니터링

- 구동
  - 해당 노드의 CUBRID HA 구성 요소(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)를 모두 구동해야 한다.
  - 구동하는 순서에 따라 마스터 노드와 슬레이브 노드가 결정되므로, 순서를 주의해야 한다.
    - 사용 방법

```
$ cubrid heartbeat start
```

  - 위 명령으로 Broker, 매니저 서버는 구동되지 않으므로 구동되어 있지 않은 경우에 별도로 각각 구동해야 한다.
  - CUBRID HA 구성 요소, Broker, 매니저 서버 구동을 함께 수행하고자 하는 경우에는 $CUBRID/conf/cubrid.conf의 service 파라미터 부분에 server를 삭제하고 heartbeat 을 포함해준 후 cubrid service start 명령으로 기동한다.
    - 설정 예시

```
...
# Any combinations are available with server, broker, manager and heartbeat.
service=heartbeat, broker, manager
...
```

    - 사용 방법

```
$ cubrid service start
```

CUBRID HA 구동 및 모니터링 - 계속

- 중지
  - 해당 노드의 CUBRID HA 구성 요소(데이터베이스 서버 프로세스, 복제 로그 복사 프로세스, 복제 로그 반영 프로세스)를 모두 종료해야 한다. 이 명령을 실행한 노드는 종료되고 HA 구성에 있는 다음 순위의 슬레이브 노드로 failover가 일어난다.
    - 사용 방법

```
$ cubrid heartbeat stop
```

  - 위 명령으로 Broker, 매니저 서버는 종료되지 않으므로 별도로 각각 종료해야 한다.
  - cubrid.conf의 service 파라미터에 heartbeat이 설정되어 있고 CUBRID HA 구성 요소, Broker, 매니저 서버 중지를 함께 수행하고자 하는 경우
    - 사용 방법

```
$ cubrid service stop
```

- 모니터링
  - CUBRID HA 그룹 정보와 CUBRID HA 구성 요소의 정보를 확인할 수 있다.
    - 사용 방법

```
$ cubrid heartbeat status
```

  - CUBRID HA의 복제 상태를 모니터링.
    - 사용 방법

```
$ cubrid applyinfo [option] database-name
예) $ cubrid applyinfo -L /home/cubrid/CUBRID/databases/testdb_master_node -r
master_node -a testdb
```

CUBRID HA 구동 및 모니터링 - 계속

- 옵션설명

옵션	기본값	설명
-r	none	트랜잭션 로그를 복사하는 대상 노드의 이름을 설정한다. 이 옵션을 설정하면 대상 노드의 액티브 로그 정보(Active Info.)를 출력한다.
-a		cubrid applyinfo를 수행한 노드(localhost)의 복제 반영 정보(Applied Info.)를 출력한다. 이 옵션을 사용하기 위해서는 반드시 –L 옵션이 필요하다.
-L	none	상대 노드의 트랜잭션 로그를 복사해 온 위치를 설정한다. 이 옵션이 설정된 경우 상대 노드에서 복사해 온 트랜잭션 로그의 정보(Copied Active Info.)를 출력한다.
-p	0	–L 옵션을 설정한 경우 설정 가능한 것으로 복사해 온 로그의 특정 페이지 정보를 출력한다.
-v		더 자세한 내용을 출력한다.

CUBRID HA 주의사항

- HA 지원 플랫폼
  - 현재 CUBRID HA 기능은 Linux 계열에서만 사용할 수 있다. CUBRID HA 그룹의 모든 노드들은 반드시 동일한 플랫폼으로 구성해야 한다
- 테이블 기본 키(primary key)
  - 기본 키가 설정된 테이블에 대해서만 CUBRID HA 그룹 내의 노드 간 데이터 동기화가 가능하다.
  - CUBRID HA 그룹 내의 노드 간 특정 테이블의 데이터가 동기화되지 않는다면 해당 테이블에 적절한 기본 키가 설정되어 있는지 확인해야 한다.
- JAVA 저장 프로시저(java stored procedure)
  - CUBRID HA에서 Java 저장 프로시저 환경 구축은 복제되지 않으므로, Java 저장 프로시저를 사용하려면 모든 노드에 각각 Java 저장 프로시저 환경을 설정해야 한다.
- CUBRID 메서드
  - CUBRID HA는 복제 로그를 기반으로 CUBRID HA 그룹 내의 노드 간 데이터를 동기화하므로 복제 로그를 생성하지 않는 메서드를 사용하면 CUBRID HA 그룹 내 노드 간 데이터 불일치가 발생할 수 있다, 따라서 CUBRID HA 환경에서는 메서드 사용을 권장하지 않는다..
- UPDATE STATISTICS 문
  - 통계 정보를 갱신하는 UPDATE STATISTICS 문은 슬레이브 노드에 복제되지 않는다.

CUBRID HA 주의사항 - 계속

- Stand-alone 모드
  - CUBRID의 stand-alone 모드에서 수행한 작업에 대해서는 복제 로그가 생성되지 않는다. 따라서 stand-alone 모드로 csql 등을 통해 작업 수행 시 CUBRID HA 그룹 내 노드 간 데이터 불일치가 발생할 수 있다.

- 시리얼 캐시(serial cache)
  - 시리얼 캐시는 성능 향상을 위해 시리얼 정보를 조회하거나 갱신할 때 Heap에 접근하지 않고 복제 로그를 생성하지 않는다. 따라서 시리얼 캐시를 사용하는 경우 CUBRID HA 그룹 내 노드 간 시리얼의 현재 값이 일치하지 않는다.

- CUBRID 데이터베이스 백업
  - 이는 지정한 데이터베이스를 백업하는 명령으로 -r 옵션을 사용하면 백업을 수행한 후 복구에 필요하지 않은 로그를 삭제한다. 하지만 이 옵션으로 인해 로그가 사라지는 경우 CUBRID HA 그룹 내의 노드 간 데이터 불일치가 발생할 수 있으므로 -r 옵션을 사용하지 않아야 한다

- INCR/DECR 함수
  - HA 구성의 슬레이브 노드에서 클릭 카운터 함수인 INCR/DECR 함수를 사용하면 오류를 반환한다.

- LOB(BLOB/CLOB) 타입
  - CUBRID HA에서 LOB 칼럼 메타 데이터(Locator)는 복제되고, LOB 데이터는 복제되지 않는다. 따라서 LOB 타입 저장소가 로컬에 위치할 경우, 슬레이브 노드 또는 failover 이후 마스터 노드에서 해당 칼럼에 대한 작업을 허용하지 않는다.

CUBRID

11. Trouble Shooting

CUBRID Server Trouble Shooting

- **데이터베이스가 구동이 안 될 경우**
  - 환경변수 설정이 제대로 되었는가?
    - env를 실행하여 PATH, CUBRID, CUBRID_DATABASES 등 설정이 정상적인지 확인한다.
  - 볼륨이 손상되지 않았는가?
    - 볼륨이 손상이 되면 백업파일을 이용하여 복구한다.
  - 데이터베이스를 생성한 사용자가 맞는가?
    - $CUBRID_DATABASES/databaes.txt 파일을 열어 해당 데이터베이스 정보가 존재하는 지 확인한다.
  - 다른 사용자가 stand-alone 모드로 데이터베이스를 사용하고 있지 않는가?
    - ps -ef 로 프로세스를 확인한 후 해당 프로세스가 종료되면 DB를 구동한다.
  - 디스크 공간이 부족하지 않는가?
    - df -h 로 CUBRID가 설치된 디스크 및 데이터 볼륨이 존재하는 디스크 공간을 확인하여 부족한 경우 불필요한 파일을 삭제하여 디스크 공간을 확보한다.
  - Log Active 볼륨 파일이 존재하는가?
    - $CUBRID_DATABASES/databases.txt 내용 중 log-path 부분을 확인하고 해당 경로에 <database_name>_lgat 파일이 존재하는 지 확인한다
    - 존재하지 않는 경우 cubrid emergency_patchlog 명령어를 이용해서 임시 로그 생성을 시도하고 실패하는 경우 백업파일을 이용하여 데이터베이스 복구 작업 실행한다.
  - cubrid master start에 실패하는가?
    - cubrid master start: fail 발생하는 경우 %hostname 명령 실행한 결과로 나오는 해당 hostname이 /etc/hosts에 명시되어 있는지 확인한다.

CUBRID Server Trouble Shooting - 계속

- **서버를 비정상 종료한 경우**
  - 데이터베이스 복구 및 rollback을 위해 stand-alone 모드 명령인 "csql -S <database-name>"을 사용하여 서버 복구 작업을 수행한 이후 "cubrid server start <database-name>" 명령으로 서버를 구동해야 한다. csql을 수행할 때 시간이 오래 걸려도 연결될 때까지 기다려야 한다.

- **데이터베이스 접속이 안 될 경우**
  - 서버에 접속되지 않는 경우의 대부분은 데이터베이스 서버나 Broker서비스가 구동 되지 않은 경우가 많다. 이런 경우 서버에서 cubrid server status 명령을 이용하여 데이터베이스 서버가 구동되어있는지를 확인하고 cubrid broker status 명령을 수행하여 Broker 서비스 상태를 확인한다.
  - 서버에 접속되지 않는 다른 경우는 클라이언트가 잘못된 데이터베이스 서버 정보를 가지고 있는 경우이다. 이런 경우는 우선 데이터베이스의 이름, 데이터베이스 서버의 hostname 또는 IP, broker port을 확인하여 클라이언트의 정보를 변경시켜준다.
  - 여기까지 이상이 없는 경우는 WAS서버의 접속 정보를 확인해 본다.

- **Log볼륨이 삭제된 경우의 대처 방법은?**
  - log active volume 및 log archive volume은 임의로 편집, 삭제 및 이동해서는 안된다. 그러나 이들 로그가 삭제된 경우, 데이터베이스서버를 종료 시킨 후, emergency_patchlog 명령을 이용하여 다음과 같이 임시 로그 생성을 시도한다.
  - cubrid emergency_patchlog -r database-name

CUBRID Server Trouble Shooting - 계속

- **Log Archive볼륨을 삭제 하여도 괜찮습니까?**
  - log archive volume은 log active volume이 일정량 적재되면 생성되는 것으로 데이터베이스에 commit되지 않는 변경 사항까지 기록하고 있기 때문에, media failure가 발생 시 데이터베이스 복구를 가능하게 한다. 정상 수행 중에 발생한 불필요한 archive log는 log 볼륨정보 파일(dbname_lginf)에 기록되지만, 자동으로 삭제되지는 않는다. 이러한 log는 임의로 삭제할 수 없으며 데이터베이스를 백업하는 utility인 backupdb의 -r 옵션을 사용해서 제거해야 한다.
  - 작업도중에 데이터베이스가 다운되었을 경우도 archive log가 생성된다. 이때는 stand-alone 모드로 데이터베이스를 회복(recovery)하면 다운되기 이전의 변경 사항을 rollback하면서 archive log도 삭제된다.
  - CUBRID HA 구성되어 있는 경우에는 HA 관련 프로세스에 필요한 보관 로그 파일이 삭제되지 않도록 하기 위해 cubrid.conf 파일에 force_remove_log_archives=no, log_max_archives=20 과 같이 파라미터를 설정하여 자동 관리되도록 한다.

- **데이터베이스의 Lock을 확인 하는 방법은?**
  - cubrid.conf 파일의 isolation_level의 값에 따라 lock의 레벨이 정해진다. 다른 lock과 관련된 파라미터 값들은 lock_escalation, lock_timeout, deadlock_detection_interval_in_secs 등 이다. 사용 중인 어플리케이션의 특성에 따라 적절한 locking 정책을 결정해서 각 파라미터 값들을 조정해야 한다.
  - 현재 데이터베이스에 접근하고 있는 클라이언트 어플리케이션들 가운데 lock 충돌과 동시성 제어 문제를 파악하기 위해 lockdb utility를 사용한다. 이 utility의 출력 정보는 서버의 클라이언트 정보, 시스템 클래스, 사용자 정의 클래스, 인스턴스 그리고 페이징 locking이다.
    - cubrid lockdb [option] database_name@localhost

CUBRID Server Trouble Shooting - 계속

- **데이터베이스에 실행중인 트랜잭션을 확인하고, 문제를 발생시키는 트랜잭션을 중단하는 방법은?**
  - 현재 실행중인 트랜잭션의 확인 및 트랜잭션 중단은 cubrid killtran 유틸리티를 이용한다. 이 유틸리티는 DBA만 수행할 수 있다.

```
$cubrid killtran -u dba testdb@localhost
Tran index      User name      Host name      Process id      Program name
------------------------------------------------------------------------------
     1(+)            dba      myhost             664          cub_cas
     2(+)            dba      myhost            6700            csql
     3(+)            dba      myhost            2188          cub_cas
     4(+)            dba      myhost             696            csql
     5(+)         public      myhost            6944            csql
------------------------------------------------------------------------------
```

  - killtran 유틸리티를 사용해서 testdb의 현재 트랜잭션 정보를 출력한다. 출력된 트랜잭션 중에서 제거하고자 하는 트랙잭션의 index 값을 확인한다.

```
$cubrid killtran -u dba -i 1 testdb
Ready to kill the following transactions:
Tran index      User name      Host name      Process id      Program name
------------------------------------------------------------------------------
     1(+)            dba      myhost            4760            csql
------------------------------------------------------------------------------
Do you wish to proceed ? (Y/N)y
Killing transaction associated with transaction index 1
```

  - 제거할 트랜잭션 1에 -i 옵션을 사용하면 다시 한 번 삭제 여부를 확인한 후에 1번 트랜잭션이 강제로 종료된다.

CUBRID Server Trouble Shooting - 계속

- **서비스 지연 현상이 발생하는 경우 확인 방법 및 종료 방법**
  - cubrid broker status 를 이용한 현재 서비스 상태 확인한다.

```
$cubrid broker status -f -s 1 broker1
% broker1- cub_cas [4307,33000] /home/cubrid/CUBRID/log/broker//broker1.access /home/cubrid/CUBRID/log/broker//broker1.err
 JOB QUEUE:0, AUTO_ADD_APPL_SERVER:ON, SQL_LOG_MODE:ALL:100000
 LONG_TRANSACTION_TIME:60.00, LONG_QUERY_TIME:60.00, SESSION_TIMEOUT:300
 KEEP_CONNECTION:AUTO, ACCESS_MODE:RW
------------------------------------------------------------------------------------------------------------------------
ID   PID   QPS   LQS PSIZE STATUS       LAST ACCESS TIME            DB          HOST   LAST CONNECT TIME      CLIENT IP
SQL_LOG_MODE
------------------------------------------------------------------------------------------------------------------------
 1  4308     0     0 25956 BUSY   2011/09/25 10:35:02          demodb       localhost 2011/09/25 10:34:56    192.168.32.1
-
SQL:
 2  4309     0     0 24912 IDLE       2011/09/25 10:32:52          -           -              -                  -
 3  4310     0     0 24912 IDLE       2011/09/25 10:32:52          -           -              -                  -
 4  4311     0     0 24912 IDLE       2011/09/25 10:32:52          -           -              -                  -
 5  4312     0     0 24912 IDLE       2011/09/25 10:32:52          -           -              -                  -
```

  - 항목 중 status를 확인하여 현재 상태가 IDLE 또는 CLOSE_WAIT 가 아닌 BUSY, CLIENT_WAIT 이면 현재 사용 중 이라는 뜻이다.
  - busy상태가 길어지고 LAST ACCESS TIME과 현재 시간의 차이가 클 경우 $CUBRID/log/broker/sql_log/에 존재하는 SQL LOG를 확인한다.
  - 해당하는 SQL LOG는 해당 <broker명>_<cas아이디>.sql.log 이며 위 예제로 표현하면 broker1_1.sql.log이다.

CUBRID Server Trouble Shooting - 계속

- **서비스 지연 현상이 발생하는 경우 확인 방법 및 종료 방법 -계속**

```
$tail -f broker1_1.sql.log
09/25 10:35:02.365 (0) *** elapsed time 0.000
09/25 10:35:02.366 (0) check_cas 0
09/25 10:35:02.367 (0) set_db_parameter lock_timeout 1000
09/25 10:35:02.380 (0) end_tran COMMIT
09/25 10:35:02.381 (0) end_tran 0 time 0.000
09/25 10:35:02.381 (0) *** elapsed time 0.015
09/25 10:35:02.381 (0) END OF LOG
09/25 10:45:05.959 (0) get_db_parameter isolation_level
09/25 10:45:05.998 (1) prepare 2
select * from olympic  where rownum between 1 and 5000;
09/25 10:45:06.324 (1) prepare srv_h_id 1
09/25 10:45:06.408 (1) execute srv_h_id 1 select * from olympic  where rownum between 1 and 5000;
09/25 10:45:06.481 (1) execute 0 tuple 25 time 0.073
09/25 10:45:06.568 (0) end_tran COMMIT
09/25 10:45:06.570 (0) end_tran 0 time 0.002
09/25 10:45:06.571 (0) *** elapsed time 0.614
...
```

  - 위와 같은 진행 상태를 확인 할 수 있으며 어느 부분에서 작업이 지연되고 있는지 파악 가능하다.
  - 해당 작업을 강제 종료하고자 할 때는 cubrid broker status 모니터링 결과 중 STATUS 상태에 따라 작업내용이 달라진다.
    - CLIENT_WAIT : 접속된 응용을 종료 시키고 계속 CLIENT_WAIT이 유지되는 경우 해당 cas의 PID를 확인하여 kill -9 PID 실행으로 종료.
    - BUSY : 해당 cas의 PID를 확인하여 kill -9 PID 실행으로 cas 종료. 이후 cubrid killtran 유틸리티를 사용하여 killtran 모니터링 결과 중 Process id가 종료 시킨 cas의 PID와 일치하는 트랜잭션의 index를 확인하여 종료 시킨다.

CUBRID Server Trouble Shooting - 계속

- **Broker가 구동이 안 될 경우**
  - Broker 를 구동 시 “Shared memory open error.” 라는 메시지가 출력되면서 Broker가 시작 되지 않는 경우가 발생할 수 있다. 이 경우에는 아래와 같은 사항을 확인 후 각 상황에 맞는 처리가 필요하다.
    - Broker가 이미 시작되어 있을 수 있다. 이는 cubrid broker status를 사용하여 확인할 수 있다.
    - Shared memory가 해제되지 않은 경우가 있을 수 있다. 이는 ipcs로 shared memory 정보를 확인할 수 있으며 CUBRID 사용자가 소유권을 가지고 있는 shared memory를 해제해야 한다. Shared memory 해제는 ipcrm을 사용하여 수행할 수 있다.

```
$ipcs
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 4587521    root       644        52         2
0x00000000 4620291    root       644        16384      2
0x00000000 4653060    root       644        268        2
0x00030001 5799942    cubrid     644        10064      2
0x00030000 5832711    cubrid     644        122072     6
0x00033000 5865480    cubrid     644        94492      6

$ipcrm -m 5799942
$ipcrm -m 5832711
$ipcrm -m 5865480
```

    - Broker가 다른 사용자로 구동되어 있을 수 있다. 이 경우 해당 사용자로 로그인 하여 Broker를 중지하여야 한다.

CUBRID Server Trouble Shooting - 계속

- **데이터베이스 볼륨 중 임시볼륨의 용도는?**
  - 데이터 볼륨의 임시 볼륨은 질의 처리(order by, group by, subquery, join query, create index…) 사용된다.
  - 초기 임시 볼륨(permanent temp)이 부족하면, 추가 임시 볼륨(temporary temp)이 생성되는데, 이는 성능 저하를 초래하므로 충분한 양의 임시 볼륨을 미리 할당하는 것이 필요하다.
  - space 제한은 temp_file_max_size_in_pages 파라미터에서 지정한다. 초기 임시 볼륨은 영구적인데 반해, 추가로 생성된 임시 볼륨은 데이터베이스 서버가 종료되거나 시스템의 shutdown, 데이터베이스 서버를 재 구동할 때 제거된다.

- **Broker가 비정상 종료한 경우**
  - Broker가 비정상 종료할 경우 Broker를 구동한 후 cubrid broker status 를 수행하면 PSIZE가 -1 또는 비정상적일 경우가 발생한다. 이 경우는 cubrid broker restart 명령을 사용하여 Broker를 restart한다.

- **에러로그 확인 방법**
  - CUBRID 운영 중 발생하는 에러는$CUBRID/log/server/ 디렉터리에 DB명_년월일_시분.err 파일로 존재한다.

CUBRID Manager Trouble Shooting

- **CUBRID Manager로 Manager Server에 접속 문제**
  - CUBRID Manager Server가 정상적으로 실행중인지 확인 한다, cub_auto, cub_js 프로세스가 정상적으로 실행중인지는 다음의 명령어로 확인할 수 있다. Linux는 ps -ef | grep " cub " 윈도우는 작업관리자의 프로세스탭에서 cub_auto.exe, cub_js.exe가 실행중인지 확인한다, 만약, 확인할 수 없다면 cubrid manager start를 시도한다.
  - 간혹 CUBRID DBMS를 업그레이드하면서 기존 버전이 언인스톨되지 않은 상태에서 설치할 경우 정상적으로 시작되지 않은 경우가 있다, 필히 기존 버전을 언인스톨한 후 설치해야 하며, 언인스톨시 $CUBRID/conf와 databases는 백업을 권장한다.

- **CUBRID Manager 접속 비밀번호를 분실한 경우 초기화 방법**
  - Linux 초기화 방법
    - 1. $CUBRID/conf/cm.pass를 다른 이름으로 변경 한다.
    - 2. 다른 유저로 CUBRID를 설치 한다.
    - 3. 신규로 설치한 $CUBRID/conf/cm.pass 파일을 비밀번호를 분실한 서버의 $CUBRID/conf로 복사 한다.
    - 4. Manager 접속 초기 비밀번호 admin / admin을 사용하여 연결하신 후 비밀번호를 변경한다.
  - Windows 초기화 방법
    - {CUBRID 설치 폴더}/conf/cm.pass 파일을 편집하여 admin에 해당하는 줄을 아래의 값으로 교체한 다음 저장하고 CUBRID Manager로 다시 접속하면 기본 비밀번호인 admin으로 접속할 수 있게 된다,
      접속 후 바로 비밀번호 변경 대화창이 뜨면 변경해서 사용한다.

```
admin:6e85f0f80f030451dc9e98851098dfb2
```

CUBRID Manager Trouble Shooting- 계속

- **CUBRID Manager로 서버 접속 시 연결 시간 초과로 접속이 안 되는 현상**
  - Linux CUBRID 포트가 방화벽(iptables) 설정이 INPUT, OUTPUT 모두 설정되어 있는지 확인한다.
    - # netstat -an | grep (1523, 30000, 33000, 8001-2)
    - # service iptables stop

- **CUBRID Manager로 demodb를 시작하자마자 종료가 되며, standalone이라는 메시지도 출력**
  - 서버에서 standalone이 나오는 경우는 서버가 완전히 종료되지 않은 상태에서 다시 시작을 했을 경우 발생되며 완전히 종료한 후에 다시 시작한다.

- **CUBRID Manager SQL 실행 결과와 오류 메시지 깨져서 보이는 경우**
  - 데이터베이스와 CUBRID 질의 편집기 문자셋이 맞이 않아 한글이 깨져 보이므로 DB와 동일한 문자셋으로 설정한다.
  - 오류 메시지가 깨어지는 경우 대부분이 리눅스 서버를 윈도우 클라이언트가 접속했을 경우로 CUBRID DBMS의 환경 변수를 확인 한다, 대부분 $CUBRID_LANG 값과 클라이언트의 환경이 맞지 않을 경우가 대부분일 것이며 이 경우 오류 메시지가 깨어질 수 있다.
  - 다음 4가지 경우와 같이 서버와 클라이언트의 환경을 맞추어주면 오류 메시지는 깨어지지 않는다.
    - 서버(리눅스) + 클라이언트(윈도우) : 서버의 $CUBRID_LANG=ko_KR.euckr 또는 $CUBRID_LANG=en_US.utf-8로 설정.
    - 서버(리눅스) + 클라이언트(리눅스) : 서버의 $CUBRID_LANG과 클라이언트의 $LANG을 동일하게 설정.
    - 서버(윈도우) + 클라이언트(리눅스) : 서버의 $CUBRID_LANG과 클라이언트의 $LANG을 ko_KR.euc-kr로 설정.
    - 서버(윈도우) + 클라이언트(윈도우) : 문제 발생하지 않음.
    - 만약, 영문으로만 출력을 원하신다면 $CUBRID_LANG=en_US.UTF-8로 변경하시면 됨.

CUBRID Manager Trouble Shooting- 계속

- **csql에서는 Java Procedure 실행시 한글 깨지지 않는데, CUBRID Manager에서는 한글이 깨지는 현상**
  - CUBRID Manager와 데이터 및 Java Procedure의 문자집합을 동일하게 맞춰주어야 깨어지지 않는다, 연결시 문자집합에 UTF-8 또는 EUC-KR 등 서로 동일하게 맞춰주면 이상없이 결과가 출력된다.

- **CUBRID 매니저에서 데이터 크기가 큰 경우 조회와 로딩 실패**
  - Java Heap Memory 부족으로 인해 발생되는 현상이다, 이런 경우 로컬 시스템에 따라 특정 사이즈를 초과하는 경우에만 조회/입력 등의 작업을 실패하게 된다, Query Browser, Migration Toolkit도 동일하다.
  - CUBRID Manager 설치 폴더(C:₩CUBRID₩cubridmanager)에 있는 cubridmanager.ini를 텍스트 편집기로 빨간색 부분(Java Heap Memory 최대값)을 변경하고 큐브리드 매니저를 재시작하면 된다, 기본은 512MB인데, 이를 로컬 작업 환경에 맞게 증가시켜주면 된다. (예: 1024MB)

```
-startup
plugins/org.eclipse.equinox.launcher_1.1.0.v20100507.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.0.v20100503
-nl
en_US
-vmargs
-Xms40m
-Xmx1024m
```

Q&A

감사합니다.