

오픈 소스 데이터베이스 CUBRID 활용을 위한  
데이터베이스 운영자 과정



기술본부

CUBRID2008 R3.1 v20110320

본 문서는 나눔 글꼴로 작성되었습니다.

© 2009 CUBRID Co, Ltd. All rights reserved.

CUBRID

# 목 차

1. CUBRID 살펴보기
2. CUBRID 설치 및 기본환경 구성
3. 데이터베이스 구조
4. 데이터베이스 프로세스 관리
5. 데이터베이스 생성
6. 데이터베이스 관리
7. 데이터베이스 환경 설정
8. CUBRID HA
9. CUBRID BROKER



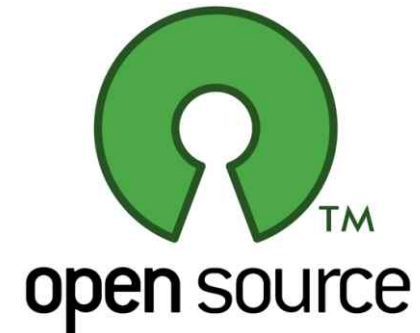
---

# 1. CUBRID 살펴보기



# CUBRID 소개

- 인터넷 서비스 최적의 DBMS 지향
- NHN에서 제품 개발을 하는 국내 유일의 오픈 소스 DBMS
  - 2008년 11월 오픈 소스 전환
- 서버(GPL), 인터페이스(BSD)의 유연한 오픈소스 라이선스
- SQL-92 표준 준수
- 트랜잭션 기반의 ACID 충족
- 온/오프라인 백업, 전체/증분 백업, 압축백업
- 시점 복구 지원
- HA



---

## 2. CUBRID 설치 및 기본환경 구성

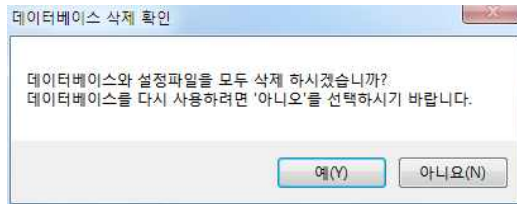


# 설치 환경

OS	CUBRID 2008 R3.1 Patch 1	용량	호환 OS	호환 CPU	일자
MS Windows 32bit	CUBRID-Windows-x86-8.3.1.1002.exe	61.25M	* Windows XP Home 32bit * Windows XP Pro * Windows 2003 * Windows Vista 32bit	* x86 * Intel EM64T * AMD64	11-03-08
MS Windows 64bit	CUBRID-Windows-x64-8.3.1.1002.exe	61.25M	* Windows Vista 64bit	* Intel EM64T * AMD64	11-03-08
Linux 32bit	CUBRID-8.3.1.1002-linux.i386.sh	83.77M	* CentOS 4이상 * Fedora 4 이상 * Ubuntu 6.10 이상 * openSUSE 11 이상 * Gentoo 2007 이상 * Asianux 2 이상 * Debian 4 이상	* x86 * Intel EM64T * AMD64	11-03-08
	CUBRID-8.3.1.1002-el5.i386.rpm	60.2M	* CentOS5 32bit	상동	11-03-08
Linux 64bit	CUBRID-8.3.1.1002-linux.x86_64.sh	86.63M	* CentOS 4이상 * Fedora 11 * Ubuntu 9.04	* Intel EM64T * AMD64	11-03-08
	CUBRID-8.3.1.1002-el5.x86_64.rpm	61.01M	* CentOS5 64bit	상동	11-03-08
Source RPM	CUBRID-8.3.1.1002-el5.src.rpm	83.93M	* CentOS 4이상 * Fedora 11 * Ubuntu 9.04	* Intel EM64T * AMD64	11-03-08
Source Code	CUBRID-8.3.1.1002.src.tar.gz	83.78M	-	-	11-03-08

- Windows 버전

- 업그레이드는 기존 제품 제거(uninstall)후 설치



- ◆ 마이그레이션 도구(migrate\_r30.exe) 수행 또는 데이터베이스 재구성
    - ◆ 예약어 증가에 따른 사전 점검 (check\_reserved.sql)

- Administrator 권한 설치 권장

- Microsoft Visual C++ 2008 재배포 패키지 필요

- ◆ ODBC/OLEDB 사용시에도 필요

- CUBRID Manager client 사용 위하여 JAVA 필요

- ◆ JRE 1.5 이상, 최신 버전 권장

## ● Windows 버전 설치

### CUBRID 2008 R3.1 for Windows

본 설치 프로그램은 컴퓨터에 CUBRID를 설치할 것입니다.

CUBRID 2008 R3.1 이전 버전에서 생성된 CUBRID 데이터베이스는 CUBRID 2008 R3.1와 호환되지 않습니다. CUBRID 2008 R3.1에서 사용하려면 마이그레이션이 필요합니다.

설치유형을 선택 하십시오.

전체 설치  
관리 도구 및 드라이버 설치

### 호환성 정보

다른 버전의 CUBRID가 각각 다른 기기에서 이용될 경우, CUBRID 서버 2008 R3.1 은 CUBRID 클라이언트 2008 R3.1 과만 호환됩니다. 계속 진행하시겠습니까?

예(Y)

아니요(N)

### 질문

입력한 디렉토리는 이미 존재합니다. 설치를 계속 진행한다면 이전 버전의 파일들이 본 버전의 파일로 덮어쓰기 될 것입니다. 계속 진행하시겠습니까?

예(Y)

아니요(N)

### Microsoft Visual C++ 2008 재배포 가능 패키지

Microsoft Visual C++ 2008 재배포 패키지가 설치되어 있지 않습니다. 계속 설치를 진행할 경우 CUBRID 2008의 일부 구성요소가 정상적으로 동작하지 않을 수 있습니다. (상세 정보는 CUBRID 다운로드 페이지에서 확인 가능합니다.) CUBRID 2008 설치를 중단 하시겠습니까?

예(Y)

아니요(N)

### CUBRID 설치

JAVA가 설치되어 있지 않거나 사용할 수 없는 상태이기 때문에 CUBRID 설치 완료 후 CUBRID Manager를 사용할 수 없습니다. CUBRID Manager를 사용하려면 CUBRID 설치 후 JAVA를 설치하시기 바랍니다.

확인

### 샘플 데이터베이스(demodb)를 확인하십시오.

샘플 데이터베이스(demodb)를 생성하겠습니까?

예(Y)

아니요(N)



# 기본 환경 구성

- 방화벽 설정 (LINUX, Windows 공통)
  - CUBRID Manager : 8001, 8002
    - ◆ 질의편집기 : 30000
  - 응용개발 : 33000
- Windows 추가 설정 (기본 설정시)
  - 질의편집기 : 30001 ~ 30040
  - 응용개발 : 33001 ~ 33040
- SELINUX 설정 (/usr/CUBRID 아래 설치 가정)

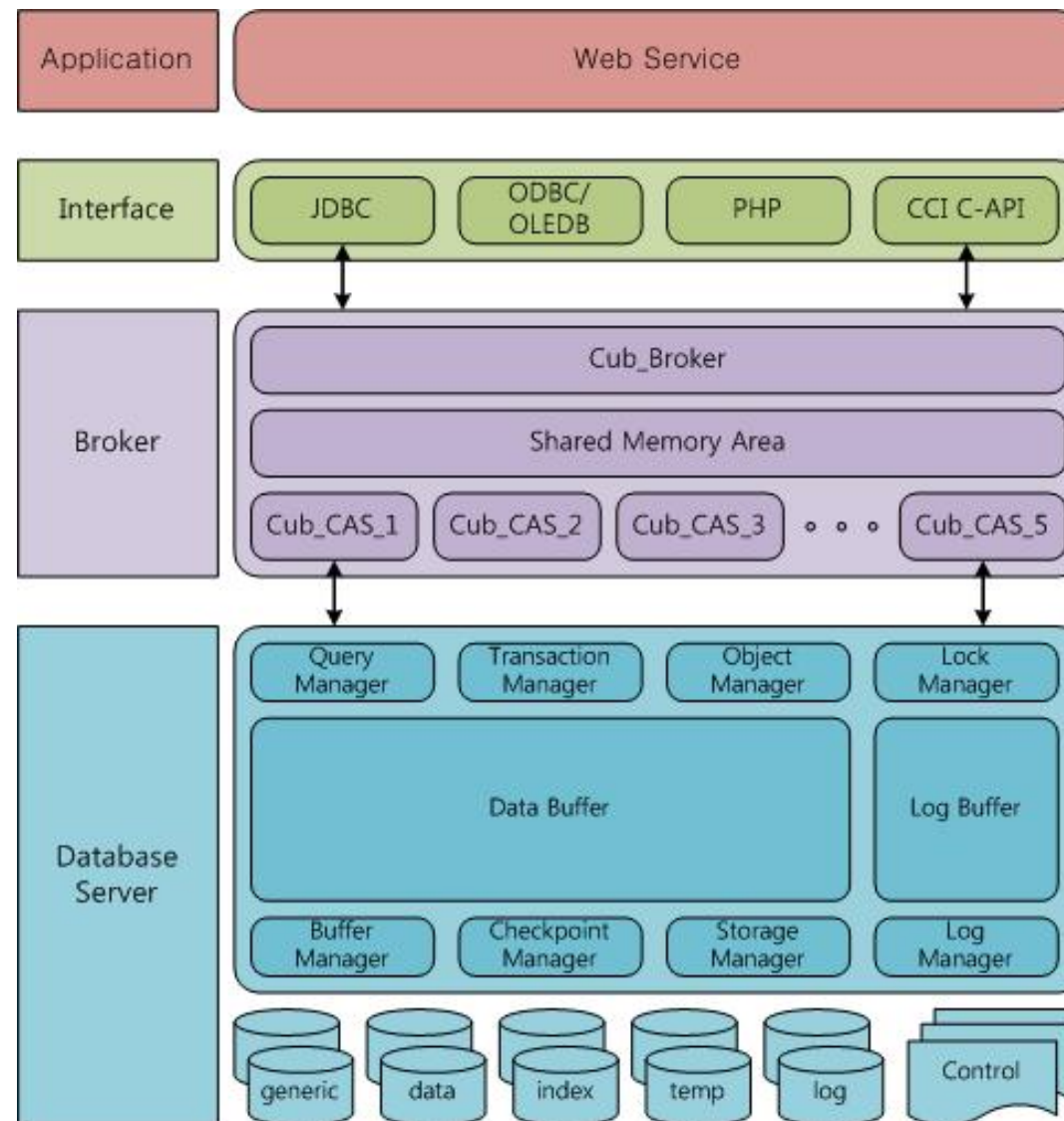
```
/sbin/restorecon -R -v /usr/CUBRID/lib/libcubridcs.so
/usr/bin/chcon -t texrel_shlib_t /usr/CUBRID/lib/libcubridcs.so
/sbin/restorecon -R -v /usr/CUBRID/lib/libcubridsa.so
/usr/bin/chcon -t texrel_shlib_t /usr/CUBRID/lib/libcubridsa.so
/sbin/restorecon -R -v /usr/CUBRID/lib/libcubrid.so
/usr/bin/chcon -t texrel_shlib_t /usr/CUBRID/lib/libcubrid.so
/sbin/restorecon -R -v /usr/CUBRID/lib/libbrokeradmin.so
/usr/bin/chcon -t texrel_shlib_t /usr/CUBRID/lib/libbrokeradmin.so
```

---

### 3. 데이터베이스 구조



# CUBRID 구조



# CUBRID 구조 [계속]



CUBRID/	CUBRID 의 홈 디렉토리
bin/	실행 파일이 위치한 디렉토리
compat/(LINUX)	7.x이하 버전 명령어가 위치한 디렉토리
conf/	환경 설정 파일이 위치한 디렉토리
cubridmanager/	cubrid manager client 가 위치한 디렉토리
databases/	databases.txt 와 sample 데이터베이스가 위치한 디렉토리
demo/(LINUX)	demodb생성 script가 저장된 디렉토리
doc/	매뉴얼 디렉토리
include/	include file 이 위치한 디렉토리
java/	JAVA SP 관련 파일이 위치한 디렉토리
jdbc/	CUBRID jdbc driver가 위치한 디렉토리
lib/	library 파일이 위치한 디렉토리
lib32/	32bit OS 를 위한 library 파일이 위치한 디렉토리
log/	각종 log 가 저장되는 디렉토리
msg/	CUBID 에서 사용되는 각종 메시지 파일이 저장된 디렉토리
share/	HA 관련 script가 저장된 디렉토리
tmp/	
var/	

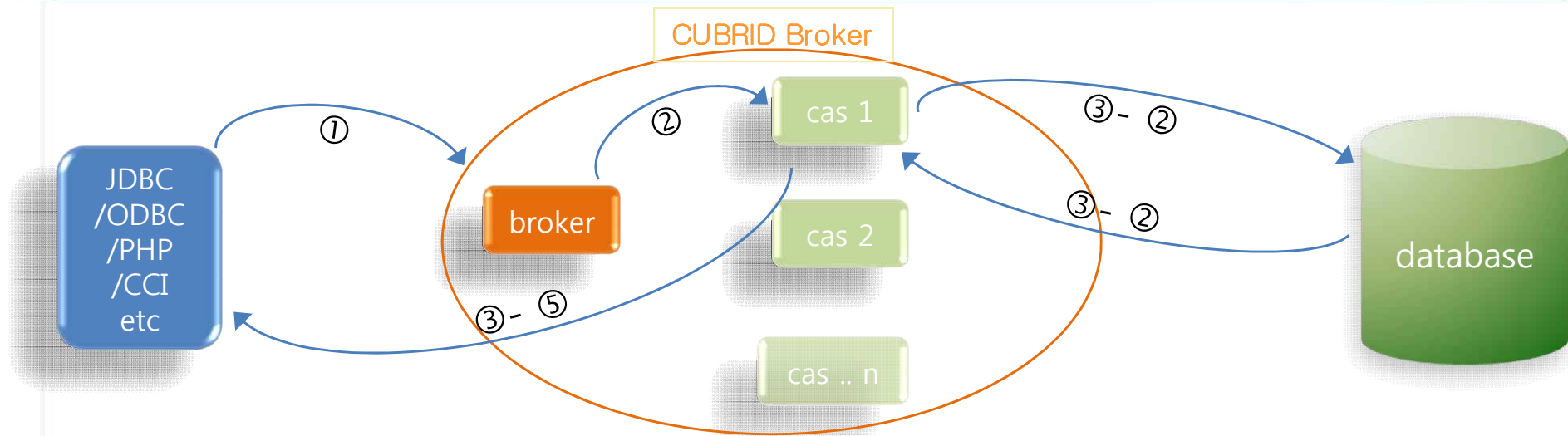


- cub\_master
  - 하나의 engine에 하나 씩 구동된다.
  - 클라이언트와 서버 프로세스 사이의 연결을 담당하는 프로세스다.
- cub\_server <db\_name>
  - 데이터베이스 구동 프로세스.
  - 구동되는 데이터베이스의 수 만큼 구동된다.
- cub\_auto
  - CUBRID 매니저 클라이언트 사용자의 인증처리 및 주기적인 자동화 작업과 진단 정보를 수집하는 기능을 수행.
- cub\_js
  - CUBRID 매니저 클라이언트로부터 전송된 사용자의 요구를 수행.



- cub\_broker
  - 응용 클라이언트와 cub\_cas 사이의 연결을 중계하는 기능을 수행.
  - cub\_cas의 상태를 파악하여 접근 가능한 cub\_cas에게 요청을 전달하고, 해당 cub\_cas로부터 전달 받은 요청에 대한 처리 결과를 응용 클라이언트에게 반환.
  - Broker 하나 당 하나씩 생성되며 각 프로세스는 cub\_cas 프로세스를 cubrid\_broker.conf에 설정된 수 만큼 구동 시킨다.
- cub\_cas
  - 연결을 요청하는 모든 종류의 응용 클라이언트가 사용하는 공용 응용 서버 역할.
  - 데이터베이스 서버의 클라이언트로 동작하여 클라이언트의 요청에 의해 데이터베이스 서버와 연결을 제공.
  - 데이터베이스 서버의 클라이언트 라이브러리와 링크되는 프로그램으로 서버 프로세스에는 클라이언트 모듈로 동작하며, 쿼리 파싱이나 최적화, 실행 계획 생성 등의 작업이 클라이언트 모듈에서 수행.

# CUBRID 구조



① 응용에서 질의처리를 위해 미들웨어에 작업 요청

① 실 작업처리는 cas 가 수행하며, 프로세스 단위 작업

② 응용에서 단위 cas에게 직접 요청은 어려우므로, 이들 관리위한 broker 를 두고 broker 에게 작업 요청

② broker 는 사용가능한 cas 에게 작업을 할당

③ cas는 데이터베이스에 연결하여 작업 수행후 결과를 응용에 전달

① 데이터베이스는 사용자가 지정하며, 지정한 데이터베이스에 cas 가 연결하는 형식

② cas 는 데이터베이스와 연결하여 작업후, 연결해제 요청시 연결해제하지 않고 연결 유지(connection pool)

③ 이후 동일한 데이터베이스와 작업 요청시 기존 연결 재사용

④ 다른 데이터베이스와 작업 요청시 현재 연결을 끊고, 새로이 연결

① 연결 overhead 발생 → broker 별로 데이터베이스를 지정(사용자)하여 사용 권장

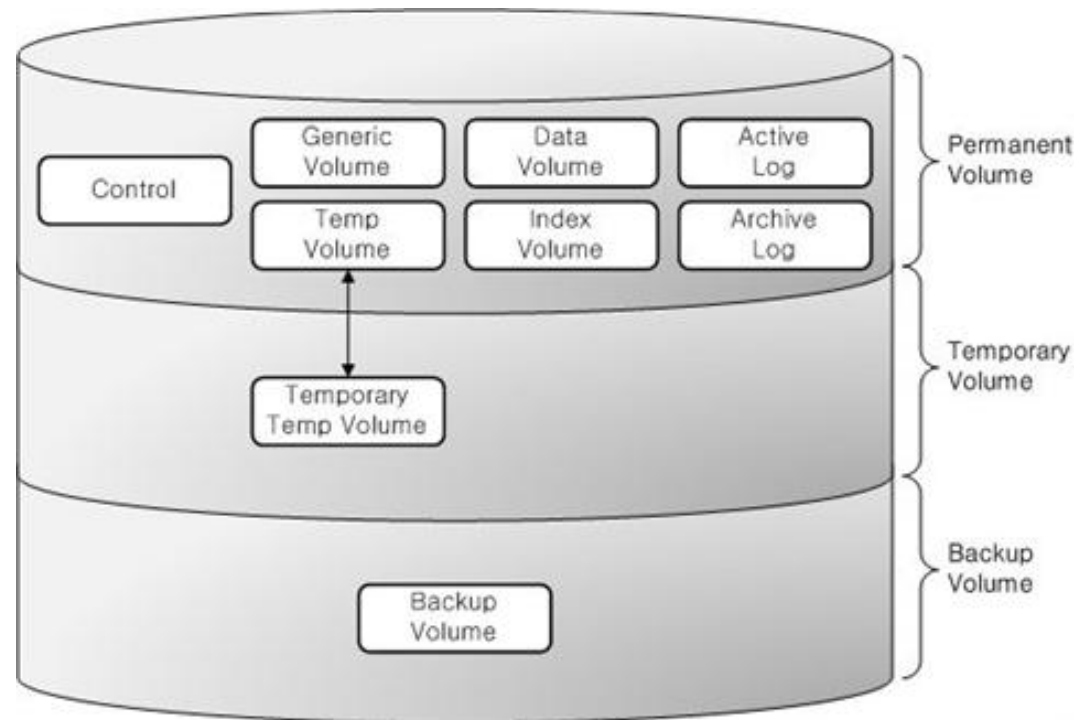
⑤ 작업결과를 응용에 전달

✓ 트랜잭션 처리중에는 하나의 cas가 하나의 응용에 종속됨, 트랜잭션 처리 종료전까지는 다른 요청 받지 않음

✓ 제한된 개수의 cas 를 통한 서비스 처리이므로, cas 사용시간 단축을 통한 다중 응용 처리 성능 향상

✓ select 도 트랜잭션의 일부

# 데이터베이스 구성 파일







- Information Volumes

- Information volume은 응용 프로그램의 모든 데이터를 저장한 파일이다.
- Information volume은 생성할 때 그 사용 목적을 명시할 수 있다.

- ◆ Data volume

- ▶ 스키마, 레코드, 멀티미디어 데이터와 같은 응용 프로그램의 데이터를 저장

- ◆ Index volume

- ▶ 인덱스와 질의 처리 속도를 높이거나 무결성 제약을 보증하는 해쉬(hash) 같은 사용자 데이터를 지원하는 정보

- ◆ Temp volume

- ▶ 질의 처리(join, group by, order by, subquery, create index..)나 정렬(sorting)위해 사용. 이것은 임시 목적으로 사용되는 permanent 볼륨이며, 뒤에서 소개할 임시 볼륨과 혼동하지 않아야 한다.

## ◆ Generic volume

- ▶ DB 생성시 초기 볼륨이며 data, index, temp 볼륨으로도 사용 가능하다.

```
cd C:\CUBRID\database\demodb
```

```
2009-10-19 오전 11:10      20,480,000 demodb
```

```
2009-10-19 오전 11:10      20,480,000 demodb_x001
```

```
2009-10-19 오전 11:10      20,480,000 demodb_x002
```



- Log Volumes

- Active 로그 볼륨은 데이터베이스에 반영된 가장 최근의 갱신 기록을 포함하는 볼륨이다.
- committed/aborted/active 트랜잭션의 상태를 기록한다.
- 매체 고장이 발생할 경우 데이터베이스 복구를 위해 사용된다.
- 각 데이터베이스 마다 하나의 active 로그 볼륨을 가진다.
- Active 로그로 할당된 스페이스가 모두 사용되면, active 로그의 내용은 새로운 로그(archive 로그)로 복사되어 보관된다.
  - ◆ 예 : demodb\_lgat(active log), demodb\_lgar\*(arcive log)

```
cd C:\CUBRID\database\demodb
```

```
2009-10-19 오전 11:15      20,480,000 demodb_lgat
```



- Control Information Volumes

- Volume Information

- ◆ 데이터베이스 볼륨에 관한 정보를 포함
    - ◆ 파일명 : demodb\_vinf

- Log Information

- ◆ 로그 볼륨에 관한 정보를 포함
    - ◆ 파일명 : demodb\_lginf

- Backup Volume Information

- ◆ 백업 볼륨에 관한 정보를 포함
    - ◆ 파일명 : demodb\_bkvinf

- databases.txt

- ◆ 데이터베이스의 위치(디렉토리) 정보를 담고 있는 파일

- cubrid.conf

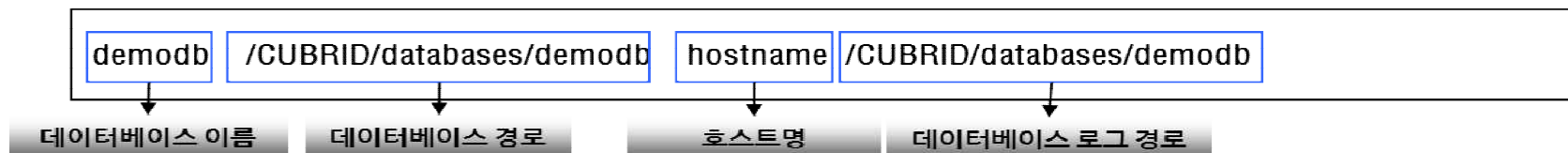
- ◆ CUBRID 시스템 파라미터의 설정값을 저장하는 파일

```
cd C:\WCUBRID\databases\demodb
2009-10-19 오전 11:10      191 demodb_lginf
2009-10-19 오전 11:10      191 demodb_bkvinf
cd C:\WCUBRID\databases
2009-10-19 오전 11:10      108 databases.txt
cd %CUBRID%\conf
2009-08-14 오후 08:27     3,504 cubrid.conf
```



- databases.txt

- 데이터베이스의 이름, 경로, 구축된 호스트이름 정보가 기록되어 있다.
- 데이터베이스 생성 시, databases.txt 파일에 생성된 데이터베이스에 관한 정보가 기록된다.
- CUBRID\_DATABASES 환경 변수가 명시되어 있는 경로에 저장된다.
- 환경변수가 지시하는 디렉토리에 존재하지 않는 경우는 현재 디렉토리를 참조한다.
- 이 파일에 대한 주의 사항
  - ◆ 호스트 이름이 변경되거나, 데이터베이스를 rm과 같은 명령어로 삭제했을 경우 이 파일도 수정해야 한다.
  - ◆ 데이터베이스 생성 및 삭제 시에 databases.txt 파일을 수정할 수 있어야 하므로, 데이터베이스를 생성하거나 삭제하는 사용자는 그 파일에 쓰기 접근을 가져야 한다. 이 디렉토리에 쓰기 권한이 없는 사용자가 데이터베이스를 생성한다면 데이터베이스 생성이 실패하게 된다. 따라서 DBA는 그 디렉토리에 사용자 쓰기 권한을 허용하거나 사용자 각자의 디렉토리에 databases.txt 파일을 생성하고 환경 변수를 설정하는 것이 바람직하다.



---

## 4. 데이터베이스 프로세스 관리





- Server Process

- 서버 프로세스는 데이터베이스당 하나가 동작한다.
- 서버 프로세스가 정상적으로 동작하기 위해서는 마스터 프로세스가 동작 중이어야 한다.
- client/server 모드로 동작하기 위해서는 서버 프로세스를 구동해야 한다.
- 서버 프로세스 구동
  - ◆ 마스터가 존재하지 않으면 마스터를 먼저 구동시킨다.
  - ◆ 이전에 비정상적으로 서버가 종료된 경우에 restart recovery가 수행된다. (경우에 따라 많은 시간이 경과됨)
- 서버 프로세스 종료
  - ◆ 현재 진행중인 트랜잭션 취소 후 종료된다.
  - ◆ kill 명령으로 서버를 종료 시키지 말 것.
  - ◆ 부득이한 경우에는 csq1 -S database로 데이터베이스를 회복시킬 것. 이때 회복 작업은 절대 중단시키지 말 것.
- 서버프로세스가 구동중인 상태에서 장비가 down되기 전 서버프로세스를 종료하여야 한다. 데이터베이스내의 트랜잭션이 정리가 안된 상태에서 종료되기 때문에 로그가 손상될 수 있기 때문이다.



- CUBRID 서비스 구동

- Windows

- ◆ 자동으로 구동됨

- ◆ 수동구동



- 명령어 이용

- ◆ cubrid service 명령어를 이용

```
[cubrid@edu ~]% cubrid service start
@cubrid master start
++ cubrid master is running.
@cubrid broker start
++ cubrid broker start: success
@cubrid manager server start
++ cubrid manager server start: success
```





- CUBRID 서비스 중지

- Windows



- 명령어 이용

- ◆ cubrid service 명령어를 이용

```
[cubrid@edu ~]% 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
```

# 서버 프로세스 관리

- 서버 프로세스 구동

- 큐브리드 매니저를 이용한 구동



- 명령어 이용 : % cubrid server start database\_name

- ◆ 로그파일 :

- \$CUBRID/log/server/<데이터베이스이름>\_yyyymmdd\_hhmm.err

- 서버프로세스 종료

- 큐브리드 매니저를 이용한 구동



- 명령어 이용 : % cubrid server stop database\_name



- 서버프로세스 상태확인
  - 구동 중인 데이터베이스 서버 리스트 출력 및 서버/마스터 종료
  - 사용방법 : cubrid server status 로 대체

valid options:

<b>-P, -server-list</b>	print server processes
<b>-R, -repl-list</b>	print replication processes
<b>-O, -all-list</b>	print all processes
<b>-S, -shutdown-server=NAME</b>	shutdown NAME database server
<b>-K, -shutdown-repl-server=NAME</b>	shutdown NAME replication server
<b>-k, -shutdown-repl-agent=NAME</b>	shutdown NAME replication agent
<b>-A, -shutdown-all</b>	shutdown all processes
<b>-h, -host=HOST</b>	connect to the master server on the given HOST

## -서버 구동확인

```
[cubrid@edu demodb]% cub_commdb -P
Server demodb (rel 8, pid 8427)
```

## -서버 구동확인

```
[cubrid@edu demodb]% cub_commdb -P
Server demodb (rel 8, pid 8427)
```



- 데이터베이스 자동시작
  - 큐브리드 서비스구동과 함께 데이터베이스 자동시작 설정
  - 환경 설정 파일(cubrid.conf)에 자동시작할 데이터베이스 이름 등록
    - ◆ “server=” 다음에 자동 시작할 데이터베이스를 공백없이 “,”로 구분하여 나열

# The list of database servers in all by 'cubrid service start' command.

# This property is effective only when the above 'service' property contains 'server' keyword.

server=demodb,edudb

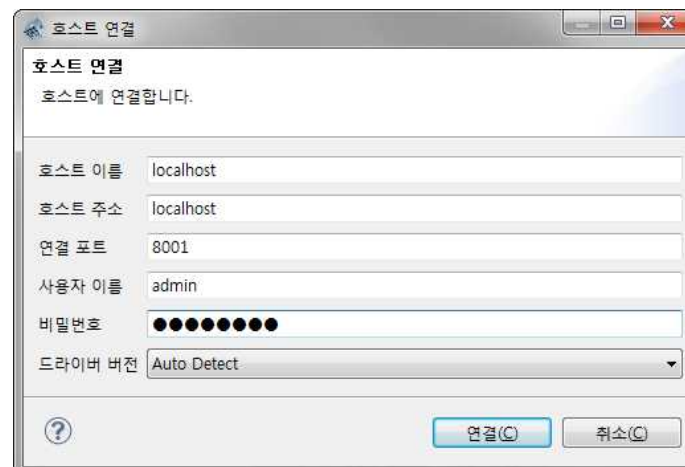
# CUBRID Manager Client

- CUBRID Manager Client

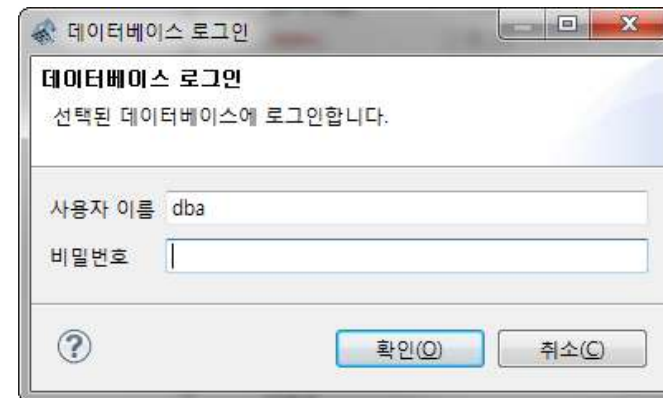
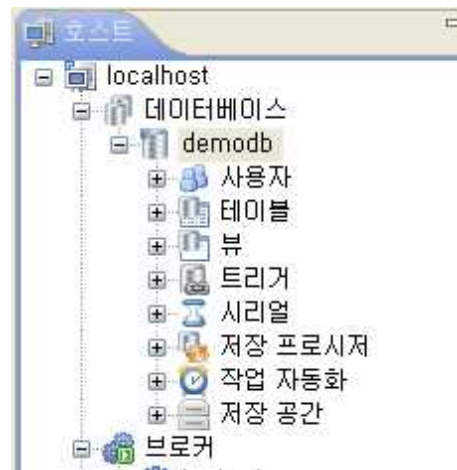
- 구동



- 로그인



- CUBRID Manager Client
  - 데이터베이스 로그인
    - ◆ 데이터베이스별 데이터베이스 사용자로 로그인



---

## 5. 데이터베이스 생성

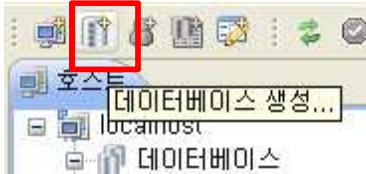




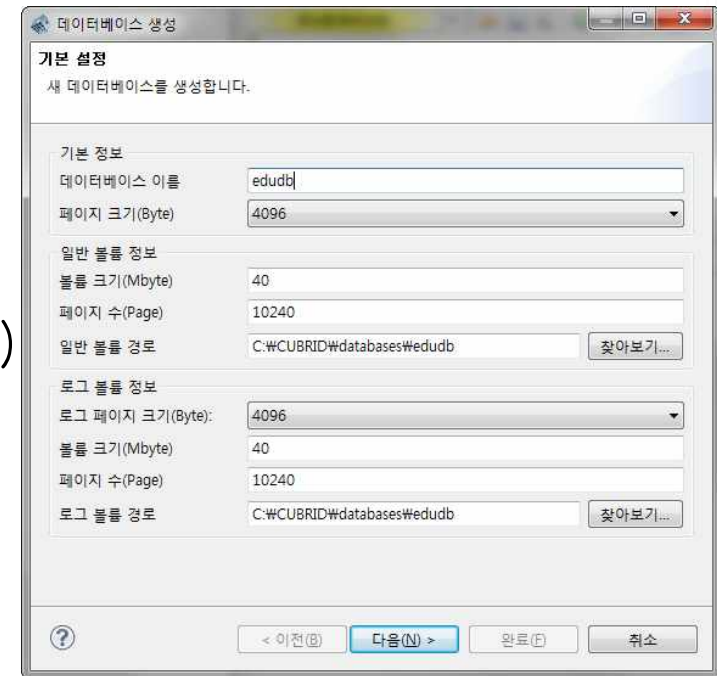
- 데이터베이스 생성시 고려 사항
  - 데이터베이스 이름 (최대 128자)
  - 데이터베이스 크기
    - ◆ 저장될 데이터의 크기 : 데이터 보전 연한 고려
    - ◆ 저장될 인덱스의 크기 : 산정 어려울 경우 데이터의 20 ~ 30% 수준
    - ◆ 질의 수행에 필요한 질의 처리용 볼륨(템프 볼륨) : 데이터의 20 ~ 30% 수준
      - ▶ 운영중 모니터링을 통한 부족 여부 확인
    - ◆ 사용할 로그의 크기 : 일반적으로 10만 페이지의 크기
      - ▶ 업무 시간중 확장되지 않는 적정 크기
      - ▶ 주기적인 백업 수행을 통한 로그 정리
    - ◆ 실 데이터베이스 크기 : 계산된 크기의 2-3배
  - 데이터베이스 위치
    - ◆ 디스크 I/O bottle-neck 최소화



- 데이터베이스 생성



- 데이터베이스 이름 지정
- 범용볼륨 정보(20M byte = 5,000 \* 4Kbyte)
  - ◆ 기본페이지 수 5,000
  - ◆ 페이지 수 4096bytes
  - ◆ 초기 데이터베이스 경로
- 로그볼륨정보(40M byte=10,000\* 4Kbyte)
  - ◆ 기본페이지 수 10,000
  - ◆ 로그볼륨 경로



- 볼륨 추가

- 데이터 / 인덱스 / 질의처리공간, 용도별 추가

데이터베이스 생성

추가 볼륨 설정

데이터베이스 생성의 추가 볼륨을 설정합니다.

추가 볼륨 정보

볼륨 이름: edudb\_data\_x001

볼륨 경로: C:\CUBRID\#databases\#edudb 찾아보기...

볼륨 형식: data

볼륨 크기(Mbyte): 40

페이지 수(Page): 10240

볼륨 추가

추가 볼륨 리스트

볼륨 이름	볼륨 형식	페이지 수	볼륨 경로
-------	-------	-------	-------

볼륨 삭제

? < 이전(B) 다음(N) > 완료(F) 취소

데이터베이스 생성

추가 볼륨 설정

데이터베이스 생성의 추가 볼륨을 설정합니다.

추가 볼륨 정보

볼륨 이름: edudb\_temp\_x002

볼륨 경로: C:\CUBRID\#databases\#edudb 찾아보기...

볼륨 형식: temp

볼륨 크기(Mbyte): 40

페이지 수(Page): 10240

볼륨 추가

추가 볼륨 리스트

볼륨 이름	볼륨 형식	페이지 수	볼륨 경로
edudb_data_x001	data	10240	C:\CUBRID\#databases\#edudb
edudb_index_x001	index	10240	C:\CUBRID\#databases\#edudb

볼륨 삭제

? < 이전(B) 다음(N) > 완료(F) 취소

# 데이터베이스 생성 [계속]

- 볼륨 자동 추가 설정
- dba 암호 설정

데이터베이스 생성

**자동 볼륨 추가 설정**  
데이터베이스의 자동 볼륨 추가를 설정합니다.

볼륨 형식 : 데이터  
☒ 볼륨 자동 추가 기능 사용  
여유 공간 비율(%) 5  
볼륨 크기(Mbyte) 40  
확장될 페이지 수 10240

볼륨 형식 : 인덱스  
☒ 볼륨 자동 추가 기능 사용  
여유 공간 비율(%) 5  
볼륨 크기(Mbyte) 40  
확장될 페이지 수 10240

? < 이전(B) 다음(N) > 완료(F) 취소

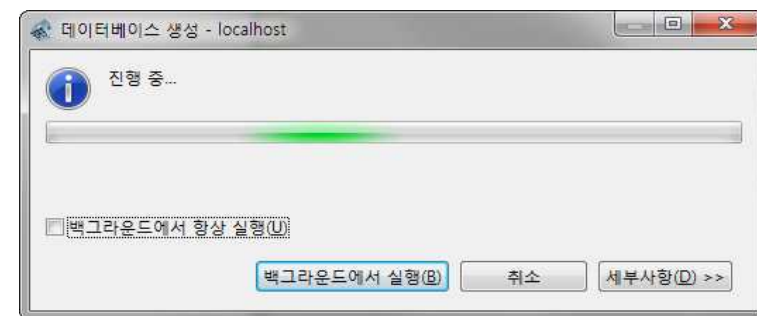
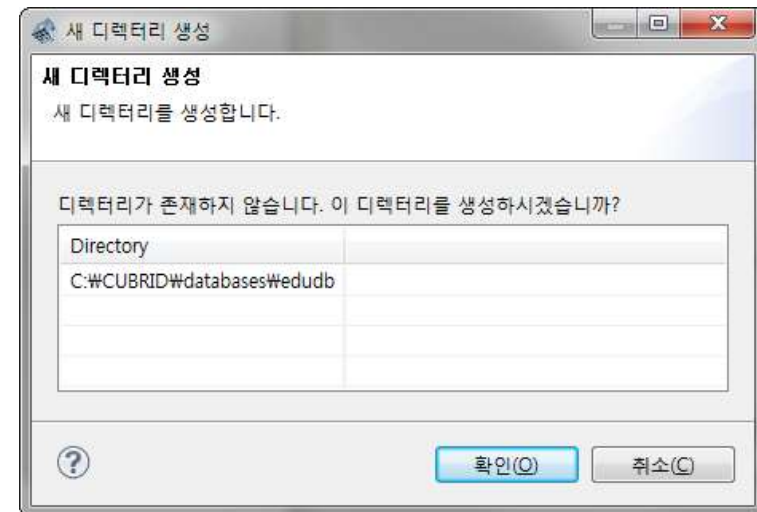
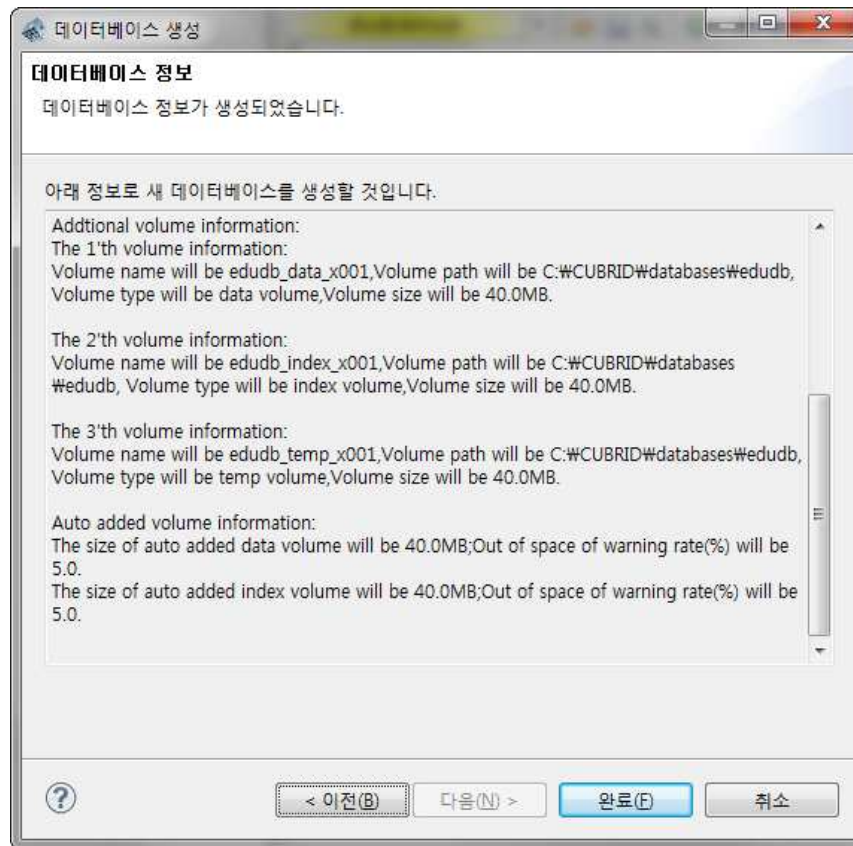
데이터베이스 생성

**DBA 비밀번호 설정**  
데이터베이스에 대한 DBA 사용자의 비밀번호를 설정합니다.

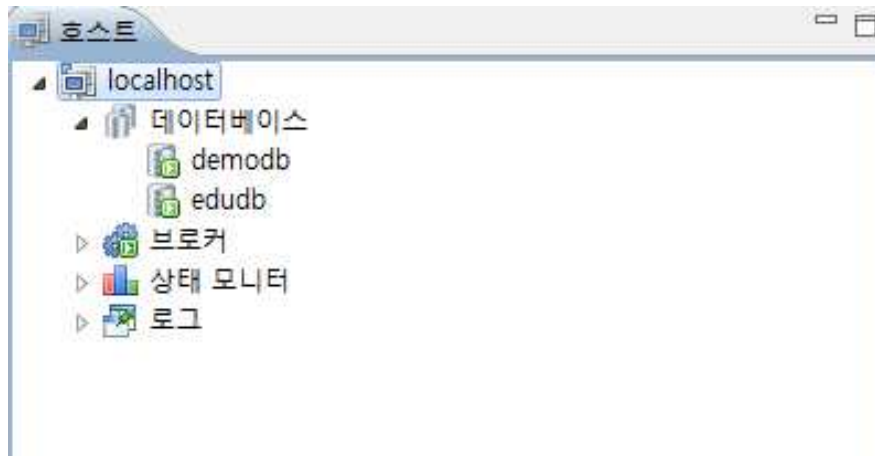
비밀번호 설정  
비밀번호   
비밀번호 확인

? < 이전(B) 다음(N) > 완료(F) 취소

- 내용 확인 및 생성



- 생성 확인
  - 생성 완료 후, 데이터베이스 서버 자동 구동





## ● 명령어 사용 : createdb

- ‘#’ 문자로 시작하는 데이터베이스명을 부여할 수 없다

```
% cubrid createdb [option] database_name
```

옵션	인 자	설 명	초기값
-F	file-path	초기 볼륨이 위치할 경로 지정	현재
-L	log-file-path	로그 볼륨이 위치할 경로 지정	현재
-B	lob-base-path	LOB파일이 저장될 위치 경로 지정	<File-path>/lob
-p	integer	초기 볼륨의 페이지 수 지정	5000
-l	log-page	로그 볼륨의 페이지 수 지정	5000
-s	Number	페이지 크기(Byte)	4096
-r		데이터베이스가 이미 존재하는 경우 기존 데이터베이스를 삭제하고 재생성함.	존재하면 에러 발생
-v		작업 내용 출력	수행 않음



- 명령어를 사용한 데이터베이스 생성

- ‘#’ 문자로 시작하는 데이터베이스명을 부여할 수 없다.

```
% cd W
% mkdir edudb
% cd edudb
```

```
% cubrid createdb edudb
```

```
Creating database with 5000 pages.
```

```
➔ 디폴트 설정(데이터베이스경로, 로그경로, pagesize, page수, 로그페이지크기)
```

```
CUBRID 2008 R1.4
```

- 생성파일

- ◆ 초기생성 되는 generic볼륨 : edudb
- ◆ 로그 정보 파일 : edudb\_lginf
- ◆ active로그볼륨 : edudb\_lgat
- ◆ 볼륨정보파일 : edudb\_vinf

- 변경파일 : databases.txt

```
edudb C:\Wedudb mycom C:\Wedudb
```

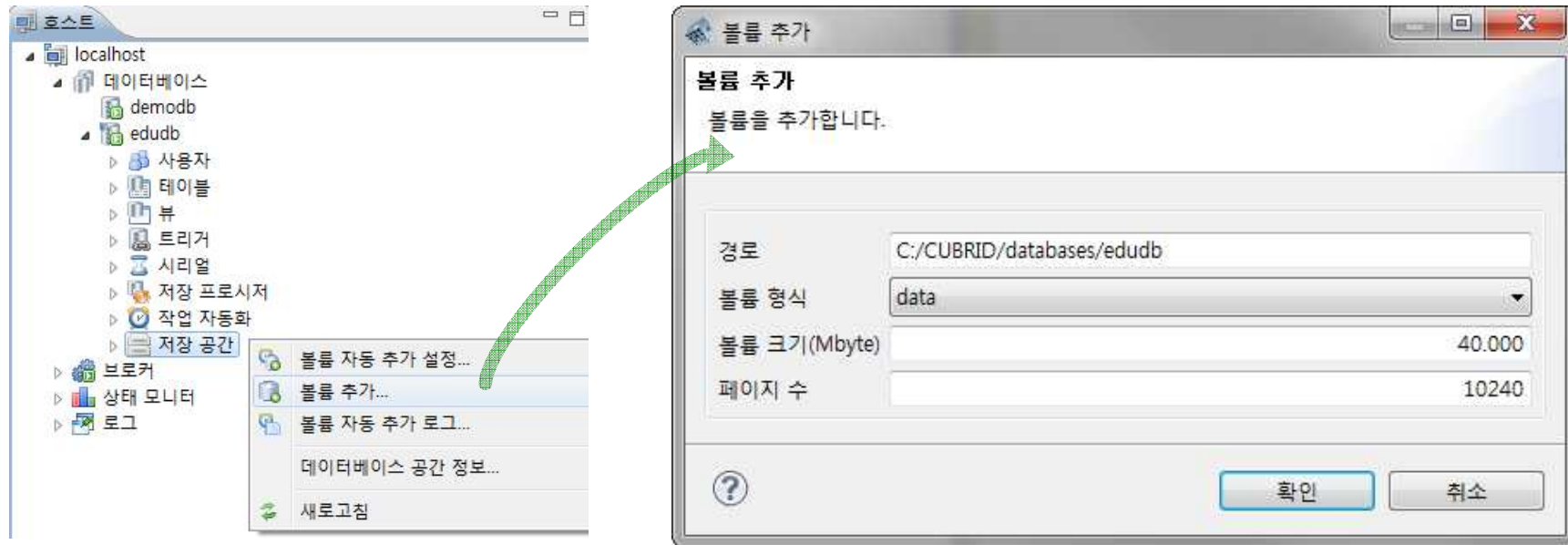
```
demodb C:\WCUBRID\DATABASE~1\demodb mycom C:\WCUBRID\DATABASE~1\demodb
```



- 왜 추가해서 사용하는가?
  - 데이터베이스가 초기화될 때 생성되는 generic 볼륨 하나로 운영하는 것보다 용도별 볼륨을 분산하는 것이 성능상 효율적이다.
  - 운영상 적합한 크기를 예측하여 사용용도별 볼륨을 확장하는 것이 필요하다.
- 볼륨추가 전 주의사항
  - 동시에 사용될 가능성이 높은 볼륨은 물리적으로 분산배치 하는 것을 권장한다. (예를 들면, data와 log, data와 index, 그리고 data와 temp)
  - permanent temp 볼륨의 부족으로 temporary temp 볼륨이 자동 생성되는 것도 성능을 저하시키므로 permanent temp 볼륨의 양을 충분하게 미리 설정해야 한다.



- 데이터베이스 트리에서 저장공간의 볼륨에서 볼륨추가를 선택



generic : 어떤 용도로도 사용 가능

data : 테이블, 레코드, 멀티미디어 등의 데이터 저장

index : 인덱스 정보 저장

temp : 질의 처리나 정렬할 때 사용

➤ 일반적으로 generic은 별도 추가하여 사용하지 않는다.



- 명령어를 이용한 볼륨생성 : addvoldb

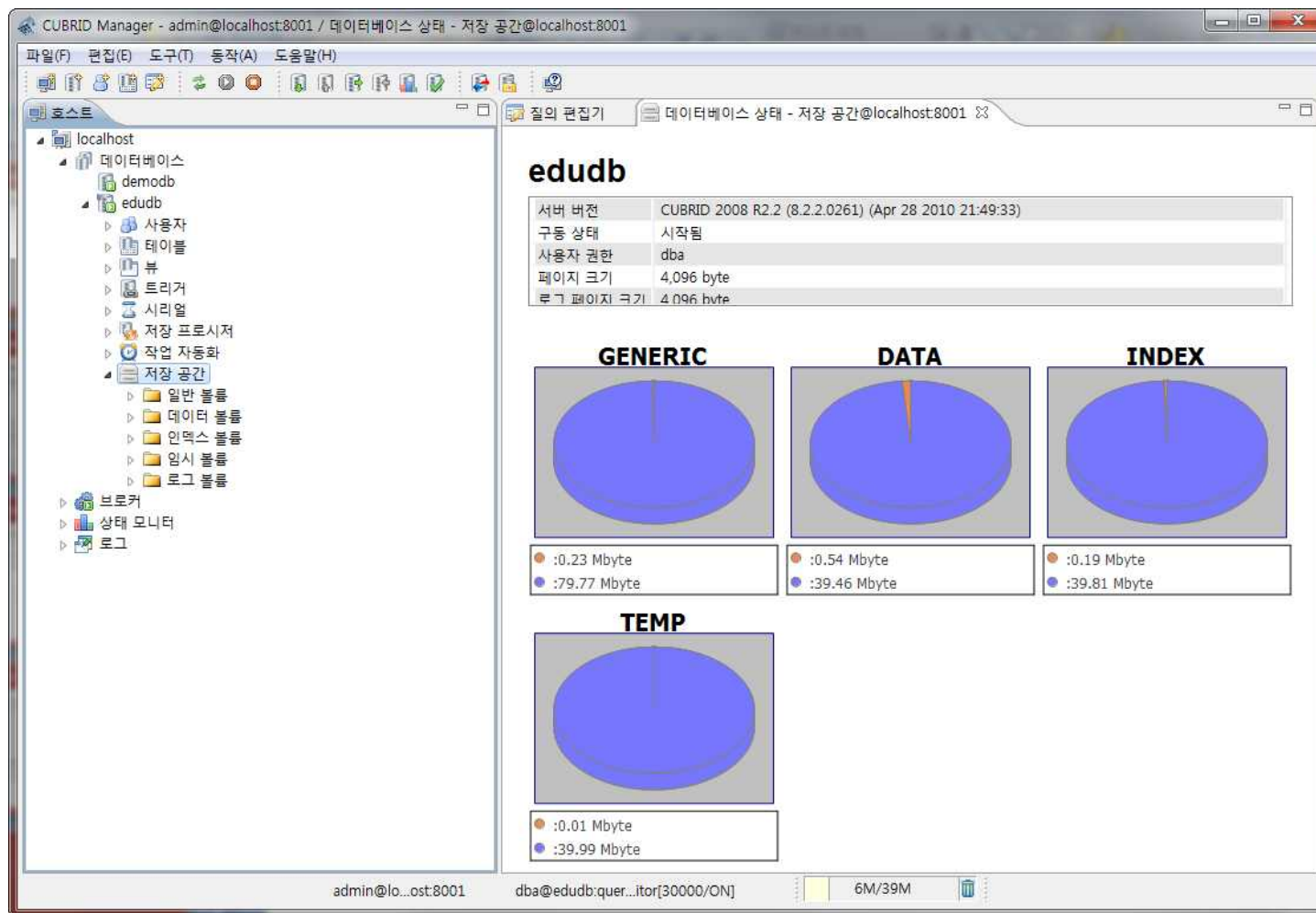
옵션	인자	설명	기본값
-S   -C		stand-alone, client-server mode 지정	
-f	vol_path	볼륨이 저장될 경로 지정	데이터베이스 경로
-p	vol_purpose	볼륨 용도 지정(generic, data, index, temp)	generic
-n	vol_name	볼륨의 이름	DB_x번호

```
% cubrid addvoldb [options] database_name number_of_pages
```

```
% cubrid addvoldb -S -p data edudb 500000
```

CUBRID 2008 R1.4

- 볼륨확장 확인





- 명령어를 이용한 볼륨확인 : spacedb

```
% cubrid spacedb [-S|-C] database_name
% cubrid spacedb -S edudb
```

- ID와 이름
- 각 볼륨의 용도
- 데이터베이스 page 크기
- 각 볼륨의 총 사용량과 여유 공간 크기
- 데이터베이스 총 사용량과 여유 페이지 크기

```
% cubrid spacedb -S edudb
```

CUBRID 2008 R1.4

Space description for database 'edudb' with pagesize 4096.

Volid	Purpose	total_pages	free_pages	Vol Name
-------	---------	-------------	------------	----------

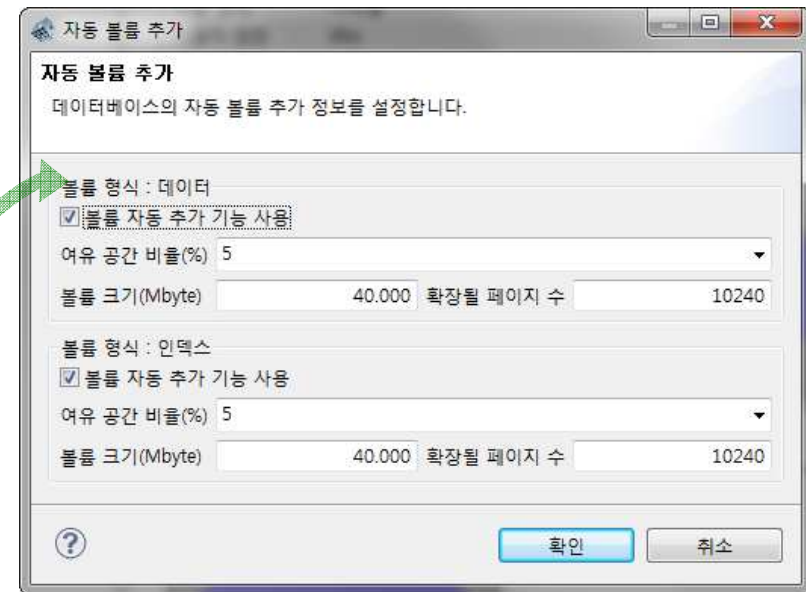
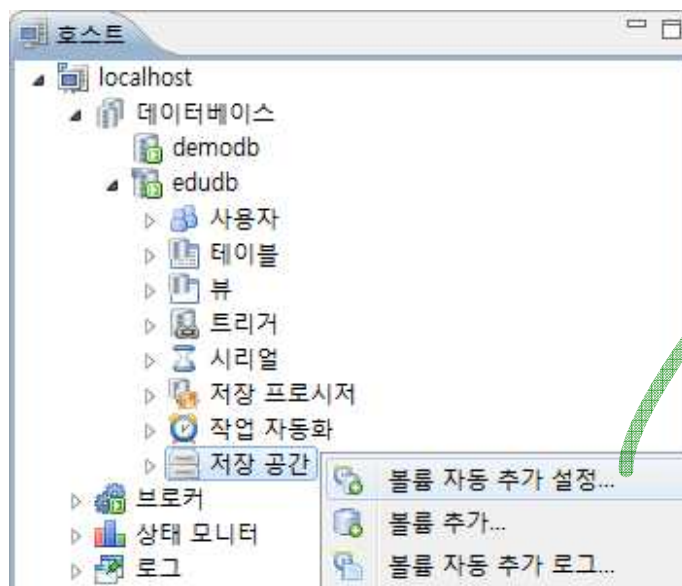
0	GENERIC	5000	4781	C:\Wedudb\Wedudb
1	DATA	5000	4997	C:\Wedudb\Wedudb_x001

---

2		10000	9778	
---	--	-------	------	--

- 확장 자동화

- Data, Index 볼륨의 여유 공간이 부족할 경우 볼륨 자동생성 설정
- 확장될 볼륨의 기준(여유공간 비율) 및 자동확장이 되는 볼륨의 크기를 지정
- Cubrid Manager에서 지원



---

## 6. 데이터베이스 관리

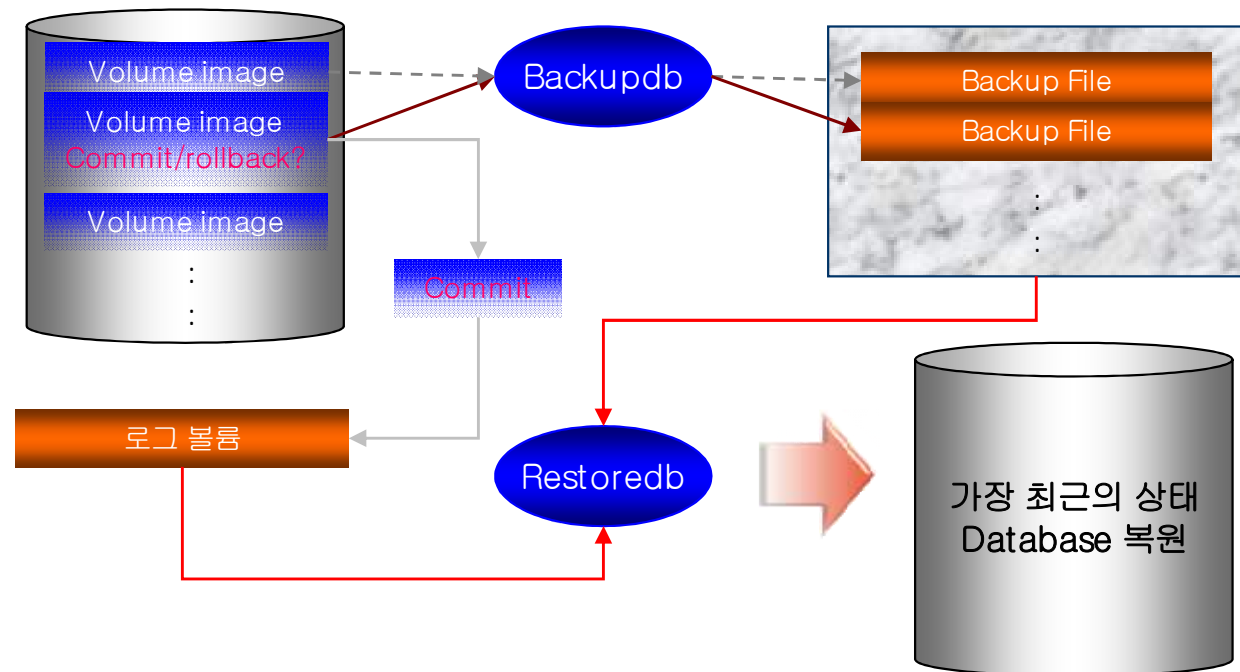




- 데이터베이스 백업 시 참고사항
  - 데이터베이스 백업은 데이터베이스의 이미지를 파일 등으로 덤프하는 것이다.
  - OS 유틸리티를 사용한 데이터베이스 백업은 권장하지 않는다.
  - 데이터베이스 백업 시점에 불필요한 로그 아카이브 볼륨들을 정리할 수 있다.
  - 백업은 매일 받는 것을 권장하며, DBA가 수작업으로 하는 것보다는 자동 백업을 하는 것이 편리하다.
  - 백업 볼륨은 외부 저장장치에 백업하는 것이 안전하다.
  - 데이터베이스를 새로운 버전으로 migration했다면 새로 생성한 데이터베이스를 즉시 백업해야 한다. 이전 버전의 백업 볼륨은 새로운 버전 복구에 사용 불가능하기 때문이다.

- Backup Volume

- Backup 볼륨은 백업할 시기의 데이터베이스의 “fuzzy”스냅샷이다.
- 이 볼륨은 백업 작업을 하기 전까지는 생성되지 않는다.
- 백업 볼륨을 이동하거나 이름을 변경하지 않으면 이후의 백업 시에 이전 볼륨을 덮어쓴다.
- 이 볼륨은 데이터베이스를 가장 최근의 상태로 복구하기 위해서 로그 볼륨과 함께 사용된다.







- 데이터베이스 백업 정책

- 백업할 데이터의 선택

- ◆ 전체 데이터베이스 또는 일부만 백업할 것인가?
    - ◆ 데이터의 유효 또는 보존 기간은 얼마로 할 것인가?
    - ◆ 데이터베이스와 함께 백업되어야 할 다른 파일은 있는가?

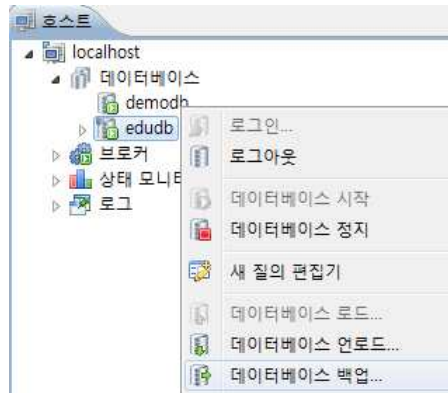
- 백업 방법

- ◆ 사용 가능한 백업 툴 및 백업 장비
    - ◆ Full/incremental 백업(0, 1, 2 level)
    - ◆ on-line/off-line 백업
      - ▶ on-line 백업 시 archive log 증가하며 백업 본과 같이 보관해야 완전 복구

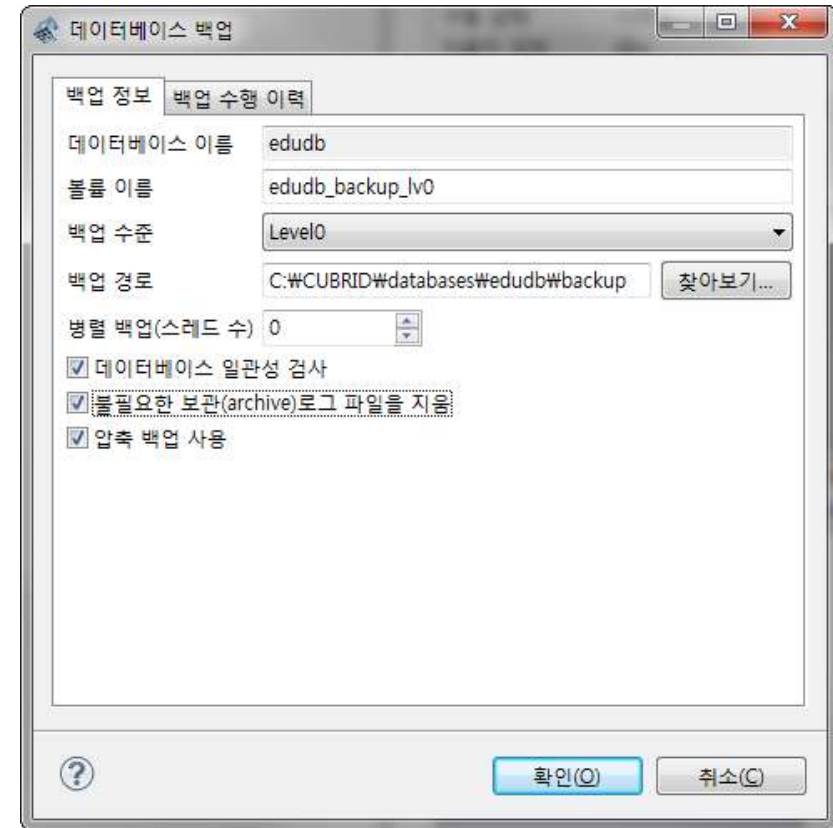
- 백업의 시기

- ◆ 데이터베이스의 활동이 가장 적은 시기

# 데이터베이스 백업 [계속]



- 백업 볼륨의 이름을 명시한다.
- 백업 레벨을 선택한다. 현재 상위 백업이 없을 경우 level0만 표시된다.
- 백업 레벨은 level2까지 이름을 있다.
- 백업 파일이 저장될 위치를 기록한다.
- 데이터베이스 일관성(consistency)를 체크한다.
- 백업 후 archive 로그 삭제를 할 경우 체크한다.
- 백업 시 사용할 thread 개수를 정한다. 0일 경우 자동으로 설정 된다.
- 데이터백업 수행 시 압축을 사용한다.





## ● 명령어 이용 : backupdb

```
% cubrid backupdb [options] database_name
```

■ disk와 tape 등의 미디어 지정 가능

■ 백업볼륨파일 및 백업정보파일 생성

옵션	인자	설명	기본값
-S -C		stand-alone, client-server mode 지정	-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 점검	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



- 명령어를 이용한 데이터베이스 백업

```
% cubrid backupdb -S demodb
```

```
➔ % cubrid backupdb -S -D C:\CUBRID\database\demodb -I 0 demodb
```

```
% cubrid backupdb -S demodb
```

```
CUBRID 2008 R2.2
```

```
Backup Volume Label: Level: 0, Unit: 0, Database demodb, Backup Time: Sun Jul 10 23:29:34 2009
```

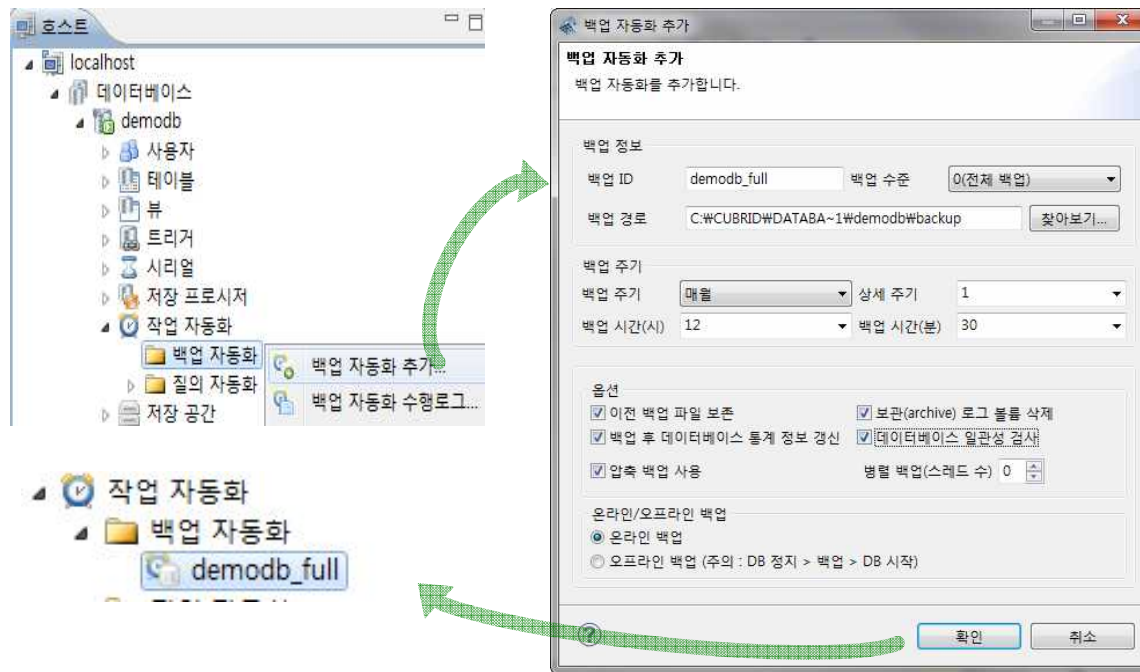
- 생성파일

- ◆ demodb\_bk0v000, demodb\_bkvinf

# 데이터베이스 백업 자동화

## ● 데이터베이스 백업 자동화

- 선택한 데이터베이스에 대하여 백업을 설정
- 주기적인 백업 설정이 가능
- 기존 백업에서 사용되는 옵션을 부여할 수 있음
- 자동화 수행에 대한 오류로그를 볼 수 있다.



1. 백업 아이디 : 중복되지 않도록 명시
2. 백업 경로 : 백업을 받을 경로 명시
3. 백업 수준 : 수행할 백업레벨
4. 백업 주기, 상세주기
5. 백업시간 : 백업을 수행할 시간
6. 온라인/오프라인 백업
  - 온라인백업 : 서버 구동 중 백업
  - 오프라인백업 : 서버 중지 후 백업  
백업후 서버 재구동
7. 옵션
  - 기존백업 옵션과 동일



- 데이터베이스 복구사례
  - 트랜잭션이 비정상 종료되어 로그가 손상된 경우
    - ◆ 비정상트랜잭션 회복하거나 로그복구
      - ➔ CUBRID Manager에서 지원하지 않는다.
  - 디스크의 media failure가 발생할 경우
  - 데이터베이스를 특정시점으로 복원하여야 할 경우
    - ◆ 백업된 데이터베이스와 로그를 이용해서 복구한다. 이때 필요한 볼륨은 백업 볼륨과 archive log, active log이다.

- 로그 복구

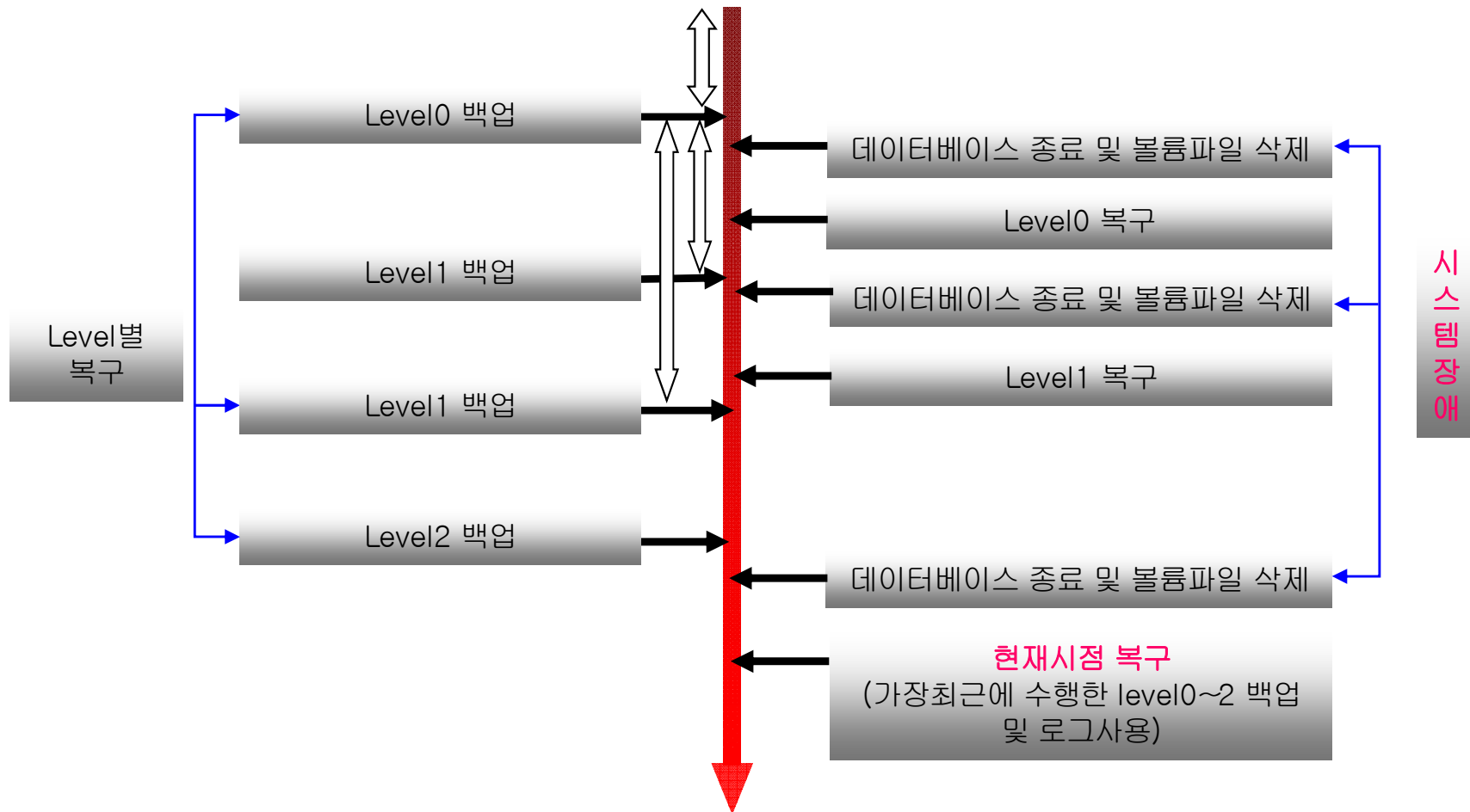
- 데이터베이스 서버가 비정상적으로 종료된 경우, 예를 들면, 데이터베이스 서버를 kill 명령어로 중지시켰거나 시스템이 갑자기 셧다운 되거나 리부팅된 경우에는 데이터베이스를 복구해야 한다. csq! 인터프리터를 stand-alone 모드로 실행하면 복구가 된다. 단, 회복 작업 시간이 매우 길더라도 절대 중단해서는 안 된다.

```
% csq! -S database_name
```

- 치명적인 오류로 인해 로그를 복구하는 과정에서도 복구가 안될 경우 emergency\_patchlog를 이용하여 로그를 복구하거나 재 생성하여 데이터베이스를 구동시킬 수 있다.
- 위의 방법으로 복구 후 csq!를 수행하여도 정상구동이 되지 않으면 “-r” 옵션을 사용하여 복구 한다.

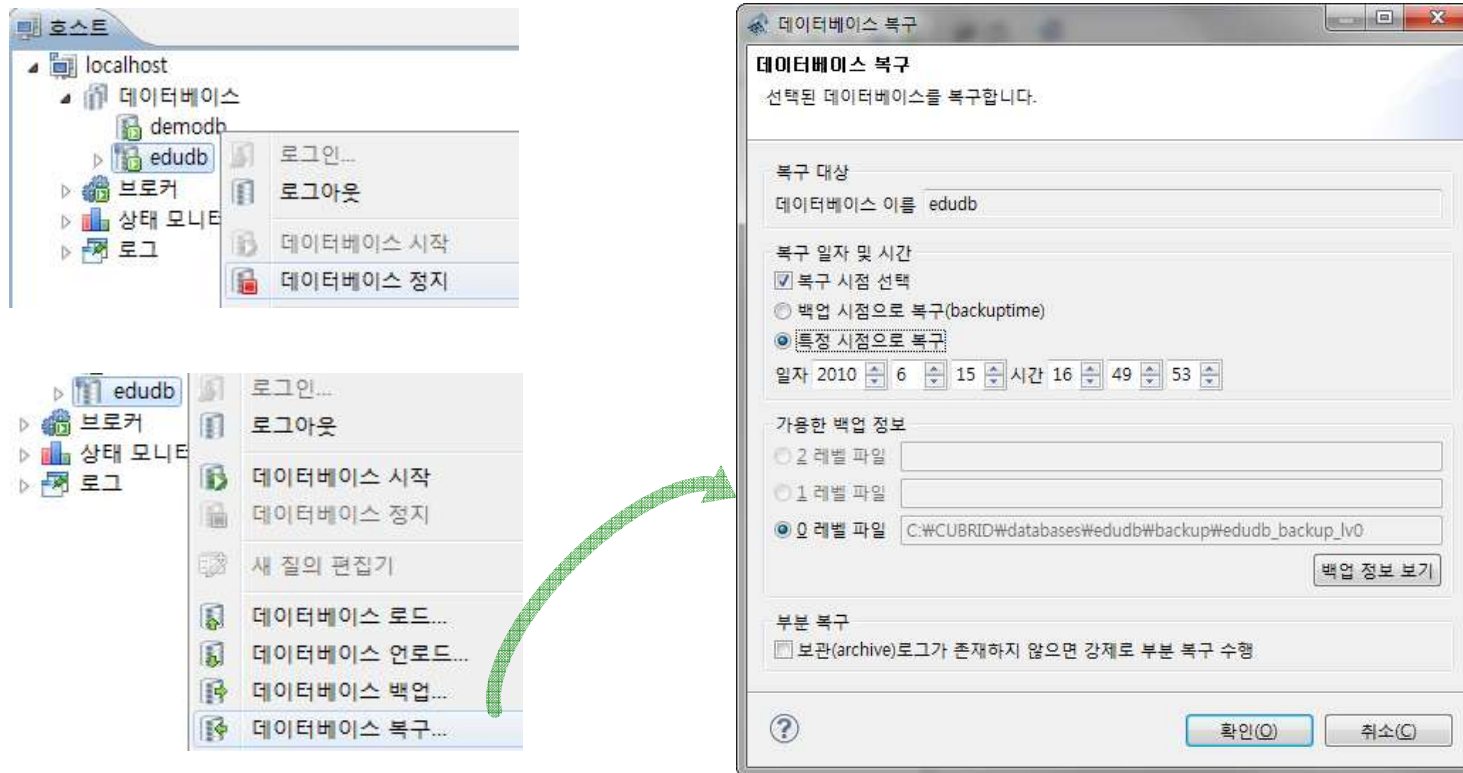
```
% cubrid emergency_patchlog demodb  
➔ emergency patch fail.  
% cubrid emergency_patchlog -r demodb
```

- 백업볼륨을 이용한 데이터베이스 복구시나리오





- 데이터베이스 복구 방법
  - 데이터베이스 복구는 서버 종료 후 가능
  - default로 데이터베이스 현재시점 복구



## ● 백업정보

### ■ 백업을 수행한 정보(stand-alone 모드에서 지원)

Database Name: C:\WCUBRID\DATABASE~1\demodb\demodb

DB Creation Time: Tue Mar 31 22:36:32 2009

Pagesize: 4096

Backup Level: 0 (FULL LEVEL)

Start\_Lsa: -1|-1

Last\_Lsa: 4451|2076

Backup Time: Sun Apr 10 23:29:34 2009

Backup Unit Num: 0

Release: 8.1.4

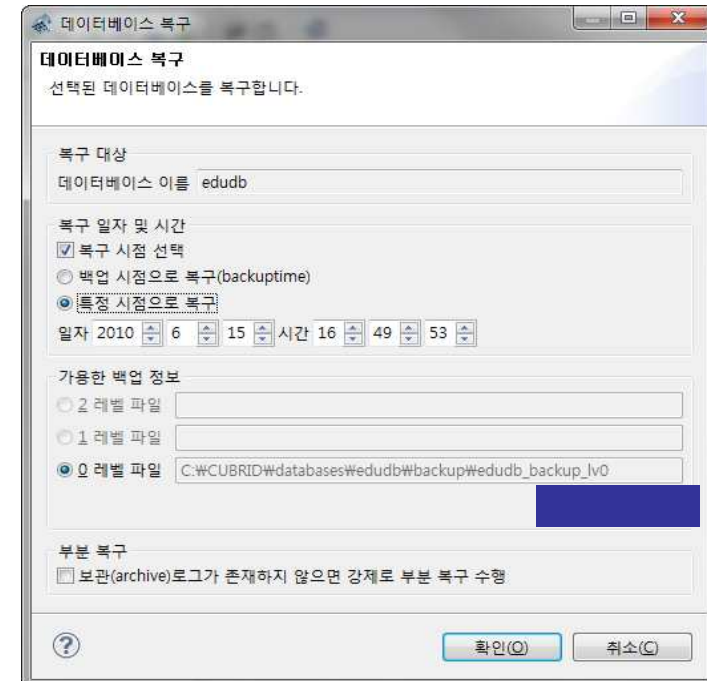
Disk Version: 8

Backup Pagesize: 131072

Zip Method: 0 (NONE)

Zip Level: 0 (NONE)

Include Active Log: YES



Database Volume name: C:\WCUBRID\DATABASE~1\demodb\demodb\_vinf

Volume Identifier: -5, Size: 223 bytes (1 pages) 볼륨 정보파일

Database Volume name: C:\WCUBRID\DATABASE~1\demodb\demodb

Volume Identifier: 0, Size: 20480000 bytes (5000 pages) 구성 볼륨

Database Volume name: C:\WCUBRID\Databases\demodb\demodb\_lginf

Volume Identifier: -4, Size: 131 bytes (1 pages) 로그 정보파일

Database Volume name: C:\WCUBRID\Databases\demodb\demodb\_lgat

Volume Identifier: -2, Size: 20480000 bytes (5000 pages) active 로그



## ● 명령어 사용 : restoredb

```
% cubrid restoredb [options] database_name
```

옵션	인자	설명	기본값
-r	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 디렉토리나 장치명을 지정	초기 로그 볼륨의 위치
-p		archive 로그가 없을 경우 무시하고 수행하여 사용자 인터페이스를 받지 않을 경우	
-u		데이터베이스 위치 파일내에 지정된 경로로 데이터베이스와 로그 볼륨을 복구(databases.txt)	
-list		백업을 수행하지 않고 백업 볼륨 목록을 출력할 경우	

# 실습 - 데이터베이스 복원

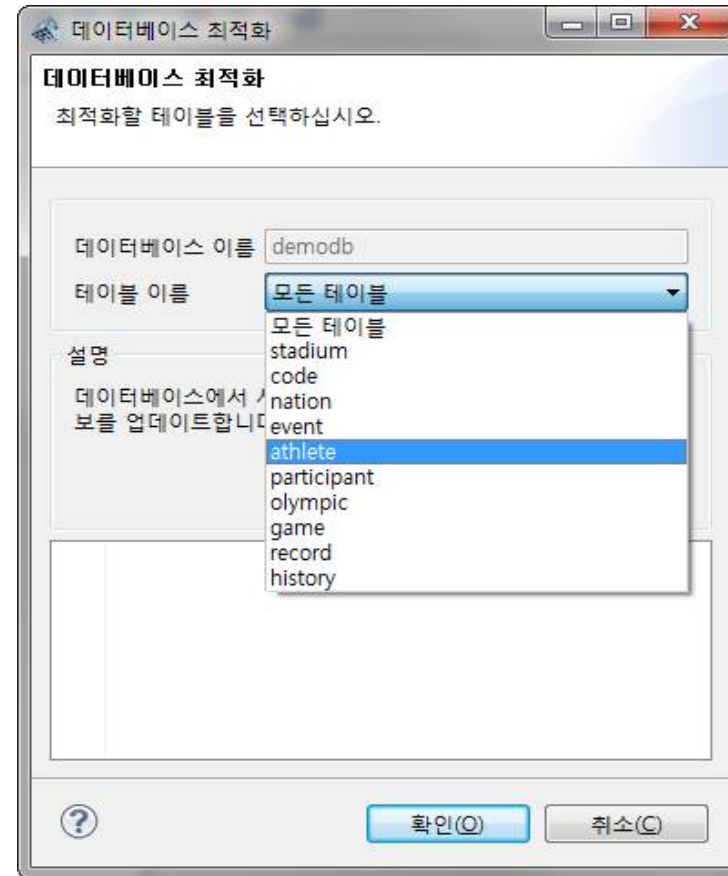
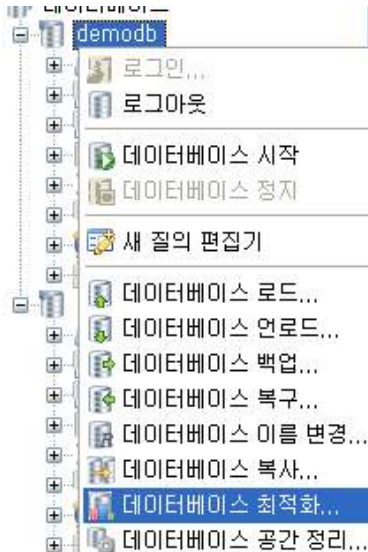
- 다음 순서와 같이 데이터베이스를 복원한다.
  - 데이터베이스 중지
  - 백업파일을 제외한 데이터베이스 구성파일 이동
    - 데이터베이스 구성파일 : demodb\*
  - restoredb명령을 이용하여 수행
    - 구성파일이 이동되므로 archive 로그 파일을 무시고 복원



- 통계정보갱신

- 큐브리드는 질의 실행 계획을 생성하기 위해 질의 최적화기를 사용한다.
- 질의 최적화기는 테이블에 있는 레코드들의 수, 접근하는 페이지들의 수, 필드 값들의 분산 같은 통계 정보를 사용한다.(예: 필드들의 최소/최대 값)
- 테이블(또는 데이터베이스)가 광범위하게 수정되었을 때, 질의 프로세스를 최적화하기 위해 `optimizedb` 명령을 사용할 수 있다.
- SQL수준인 “UPDATE STATISTICS”문으로 모든 테이블와 특정 테이블에 대하여 통계정보를 갱신할 수 있다.
- 데이터베이스 재구성을 완료 후 반드시 통계정보갱신을 수행하여야 한다.
- 주기적 수행을 권장 한다.

- 통계정보갱신 방법



- 데이터베이스 최적화 선택
- “모든테이블”의 경우 모든 테이블의 통계정보를 갱신
- 테이블 선택시 선택한 테이블의 통계정보를 갱신



- 명령어 이용 : optimizedb

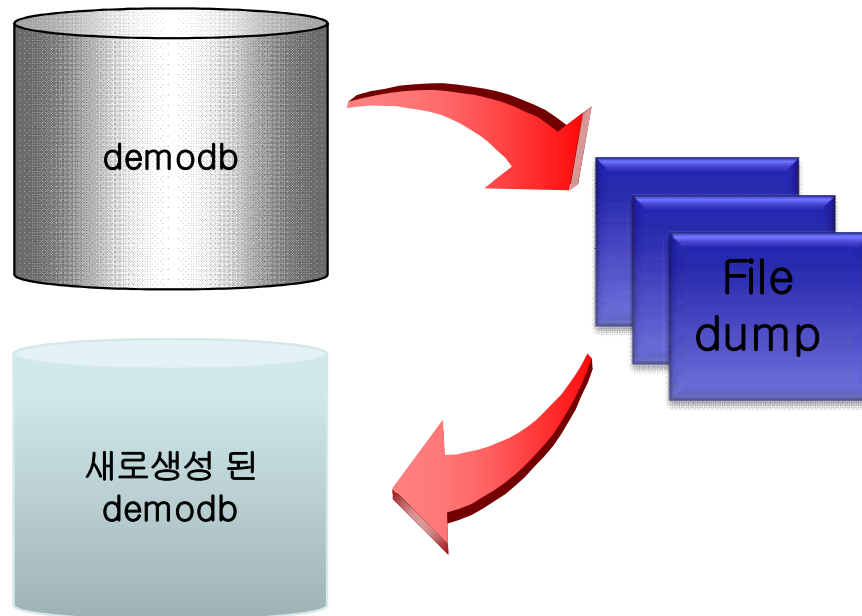
```
% cubrid optimizedb [options] database_name
```

- stand-alone 상태에서 수행 가능
- SQL 을 이용하여 수행 가능

```
update statistics on {테이블이름|all classes};  
update statistics on all classes;  
update statistics on game;
```

- 데이터베이스 재구성 방법

- 언로드 : 원본 데이터베이스를 파일로 내려 받기
- 백업 : 작업오류를 대비해 backupdb/copydb/renamedb를 사용한 백업
- Rebuilding : 기존의 데이터베이스를 삭제 및 새로운 데이터베이스 생성
- 로드 : 언로드 받은 파일을 새로운 데이터베이스에 적재

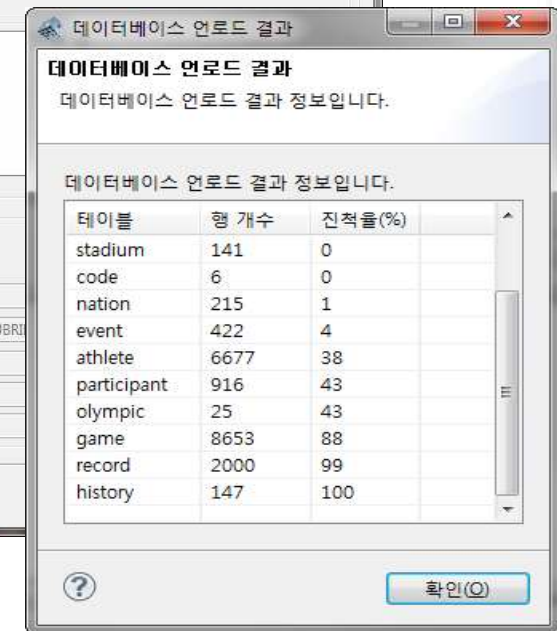
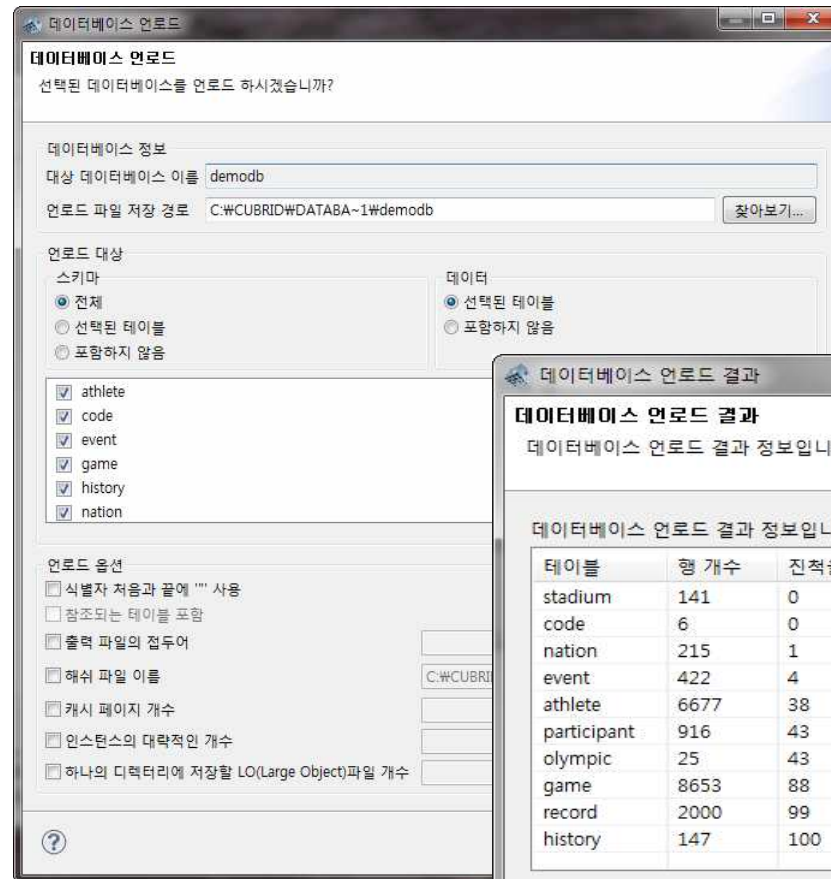
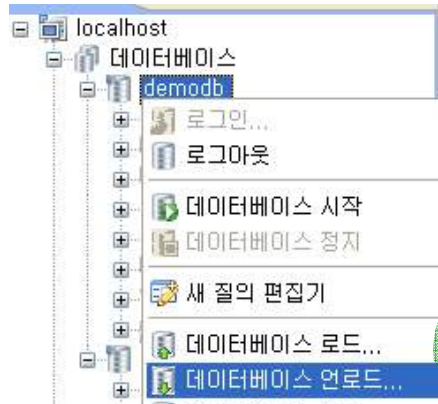






- 현재의 데이터베이스를 파일로 내려 받는다.
- 특정테이블에 대하여 백업을 하고 싶을 경우
- 데이터베이스의 전체 스키마를 확인할 경우
- 생성파일
  - 스키마 파일 : 데이터베이스의 스키마 정의 포함하는 파일(demodb\_schema)
  - 객체 파일 : 데이터베이스의 데이터를 포함한 파일(demodb\_objects)
  - 인덱스 파일 : 데이터베이스에 정의된 인덱스를 포함하는 파일(demodb\_indexes)
  - 트리거 파일 : 데이터베이스에 정의된 트리거를 포함하는 파일(demodb\_trigger)

## ● 데이터베이스 언로드 방법



### 1. 대상디렉토리

- unload 받은 파일이 저장될 위치 지정

### 2. 언로드 대상

- 스키마 : 전체 혹은 부분 테이블인지 선택

- 데이터 : 전체 혹은 부분 테이블인지 선택

### 3. 해시 파일 디렉토리 지정

- 개체 모델 사용시 사용

- 테이블 참조 관련 정보 저장파일 생성

### 4. 기타 옵션을 부여하여 수행



## ● 명령어 이용 : unloaddb

% cubrid unloaddb [OPTION] database-name

% cubrid unloaddb demodb

➔ demodb의 모든 스키마 및 데이터를 파일로 출력

옵션	인자	설명	기본값
-S   -C		stand-alone, client-server mode 지정	-C
-i	input-file	unload 받고자 하는 테이블의 이름을 기술하는 파일. 지정하지 않으면 전체테이블에 해당함	없음(모든 테이블)
-include-reference		-i 옵션에서 지정한 테이블의 레코드가 참조하는 레코드를 함께 unload	실행 없음 (지정한 테이블만 해당)
-input-class-only		-i 옵션에서 입력 파일에 지정한 테이블만 출력	시스템 테이블 언로드
-O	output_directory	파일을 출력한 경로 지정	현재 디렉토리
-s   -d		스키마만 unload   데이터만 unload	전체
-output-prefix	output_prefix	실행 결과 파일의 첫 문자열(prefix)를 지정	데이터베이스 이름
-hash-file	hash_filename	해쉬 파일 이름	시스템 자동 생성
-v		실행 과정 출력	수행 없음



- 특정 테이블 언로드

- 언로드 받을 테이블리스트 파일 생성

```
% more table.list
olympic
game
history
```

➔ 파일의 끝에 엔터 입력(Wn)

- 언로드 수행

```
% cubrid unloaddb -i table.list --input-class-only demodb -v
```

CUBRID 2008 R2.2

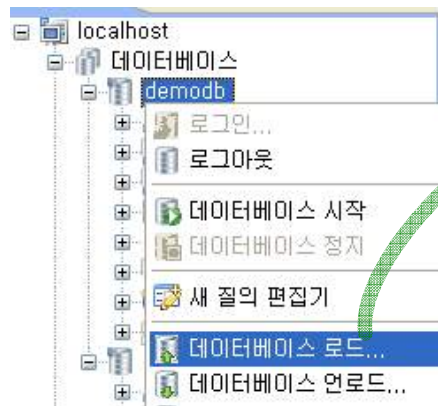
Class Name	Total Instances
olympic	25 (100% / 0%)
game	8653 (100% / 45%)
history	147 (100% / 45%)

8825 objects dumped.



- 언로드 받은 파일들을 데이터베이스로 등록
  - 언로드 형식으로 작성된 파일도 수행 가능하며 해당 테이블에 대하여 권한이 있는 계정으로 수행
- stand-alone 모드에서 DBA 권한으로 수행
  - 언로드한 데이터를 로딩하는 것이므로 각종 시스템 테이블이 포함되어 있어 dba 권한으로 수행
- 언로드된 데이터량을 확인 하고 적정 크기의 데이터베이스가 존재하여야 함
- 기존 데이터가 존재하는 데이터베이스에 로드작업을 할 경우 로드될 데이터가 적합한지 확인 한다.

## ● 데이터베이스 로드 방법

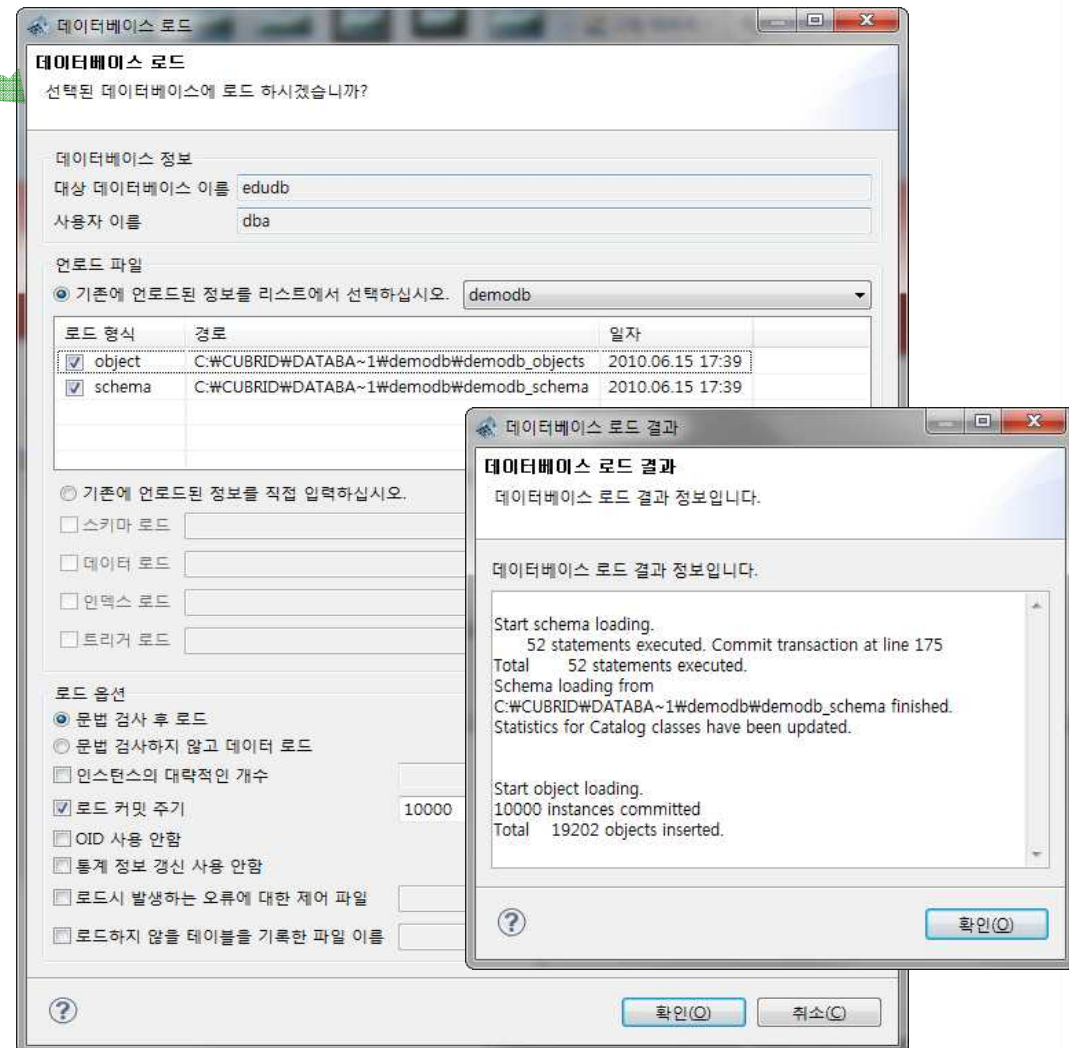


### 1. 언로드 파일

- 언로드 받은 이력을 이용하여 선택
- 이력이 없을 경우 직접 선택
  - 스키마, 데이터, 인덱스, 트리거

### 2. 로드 옵션

- 문법 검사 유무 확인
- 커밋주기 : 1만건이 무난
- OID 사용안함 : 관계형 모델일 경우
- 로드할 테이블 선택





## ● 명령어 사용 : loaddb

옵션	인자	설명	기본값
-u	user-name	데이터베이스 사용자 이름.	public
-p	password	사용자 암호	없음
-f		문법 검사하지 않고 데이터 로드만 수행, loaddb 수행 시간 단축	문법검사 및 로드 수행
-v		실행 과정 출력, -c 옵션이 있는 경우 commit된 레코드의 수 출력	수행 않음
-c	periodic-commit	periodic-commit 값만큼 주기적으로 commit 수행. -i, -s 옵션 사용될 경우에는 수행된 SQL문이 됨	commit 없음
-no-oid		oid참조 정보를 만들지 않으며 수행속도 향상, 단 다른 테이블참조하는 형태의 설계가 없어야 함	수행 않음
-s	string[:integer]	스키마 정의 파일. 특정 라인을 지정하여 로드 가능	수행 않음
-i	string[:integer]	인덱스 정의 파일. 특정 라인을 지정하여 로드 가능	수행 않음
-d	string	object input file 이름	수행 않음
-ignore-class-file	File	명시된 파일 안의 테이블들에 대하여 수행하지 않음 특정 테이블에 대하여 로드를 수행하지 않을 경우 사용	수행 않음

- loaddb 명령어 사용 순서

- 로드를 수행하는 순서는 스키마 데이터 인덱스 트리거 순이다.
- 예는 언로드 받은 파일이 스키마, 데이터, 인덱스, 트리거가 있다고 가정

```
% cubrid loaddb -u dba -s demodb_schema edudb
```

```
% cubrid loaddb -u dba -no-oid -d demodb_objects -c 10000 edudb
```

```
% cubrid loaddb -u dba -i demodb_indexes edudb
```

```
% cubrid loaddb -u dba -i demodb_trigger edudb
```



# 데이터베이스 복사 및 이동

- 현재의 데이터베이스를 다른 경로에 복사
- stand-alone 모드에서 동작
- 로그볼륨 및 확장볼륨의 경로 지정 가능
- 사용자 계정 및 암호 동일하게 복사

## 1. 원본 데이터베이스

- 복사될 이름 및 경로 표시

## 2. 대상 데이터베이스

- 원본으로부터 복사할 데이터베이스

이름/경로 명시

- 확장된 볼륨에 대한 경로 명시

- 로그볼륨의 경로 명시

## 3. 옵션

- 개별 볼륨 복사 설정

복사될 각 볼륨의 이름 변경 및 경로를 변경

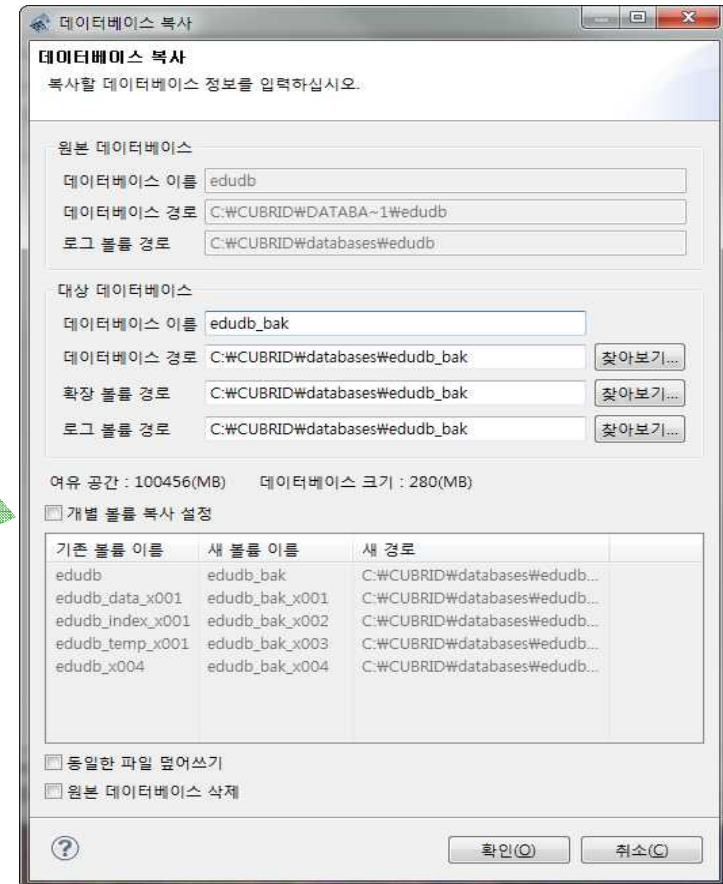
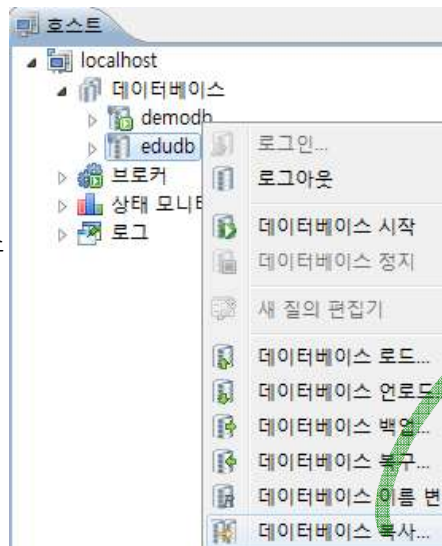
- 덮어쓰기

기존의 데이터베이스가 있을 경우 덮어쓰기

- 원본삭제

복사 후 원본데이터베이스 삭제

❖ 데이터베이스 이동에 사용





## ● 명령어 사용 : copydb

```
% cubrid copydb [OPTION] src-database-name dest-database-name
```

```
% cubrid copydb -f C:\WCUBRID\databases\wedbdb_bk
```

```
    -E C:\WCUBRID\databases\wedbdb_bk
```

```
    -L C:\WCUBRID\databases\wedbdb_bk
```

```
    edbdb edbdb_bk
```

➔ 원본 edbdb를 “C:\WCUBRID\databases\wedbdb\_bk” 경로에 로그볼륨 및 확장볼륨 포함하여 edbdb\_bk로 복사

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

# 데이터베이스 이름변경

- 이미 작성된 데이터베이스 이름을 변경
- stand-alone 모드에서 동작

## 1. 새 데이터베이스 명

-변경할 데이터베이스 이름을 명시

## 2. 옵션

-백업볼륨 삭제 : 기존의 백업 삭제 유무

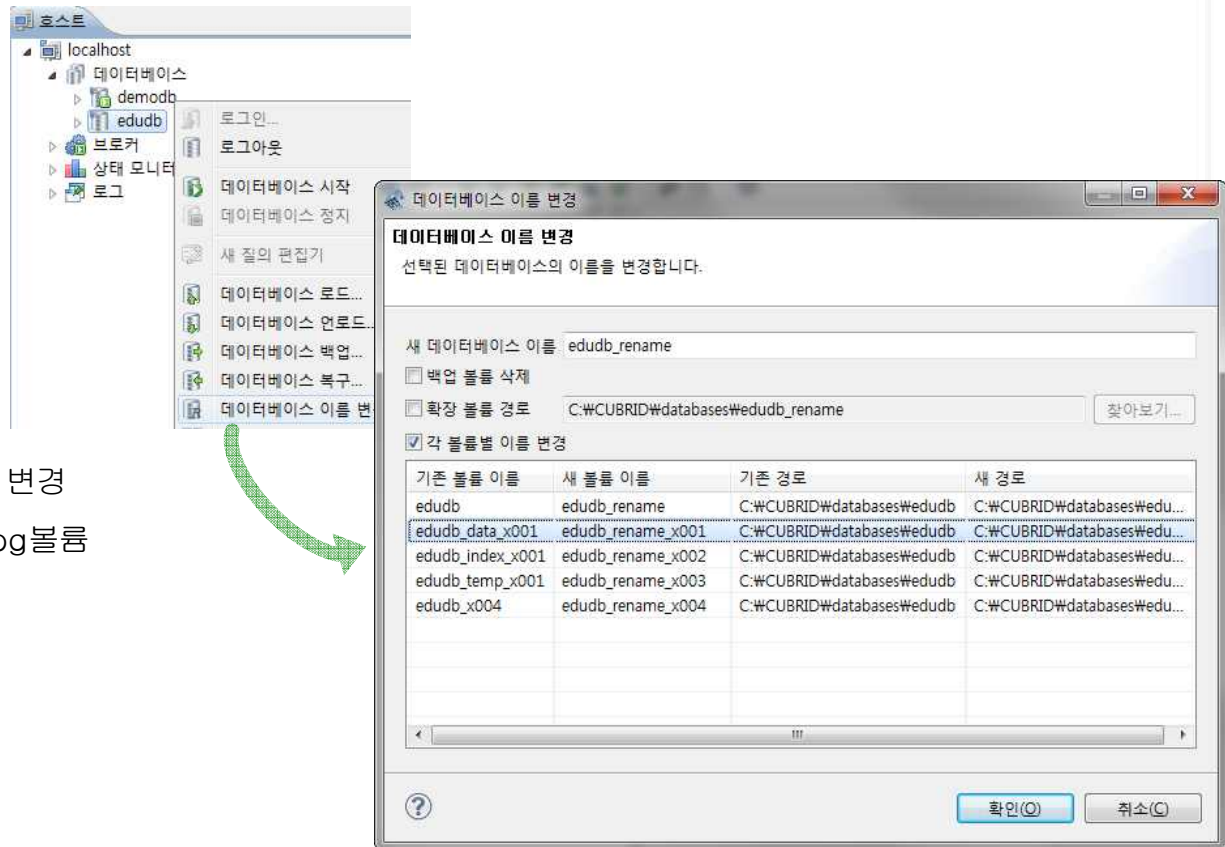
-확장볼륨 경로

추가된 볼륨의 경로를 변경할 경우 명시

-각 볼륨별 이름변경

볼륨들에 대하여 볼륨의 이름 및 경로를 변경

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



- 명령어 사용 : renamedb

```
% cubrid renamedb [OPTION] src-database-name dest-database-name
```

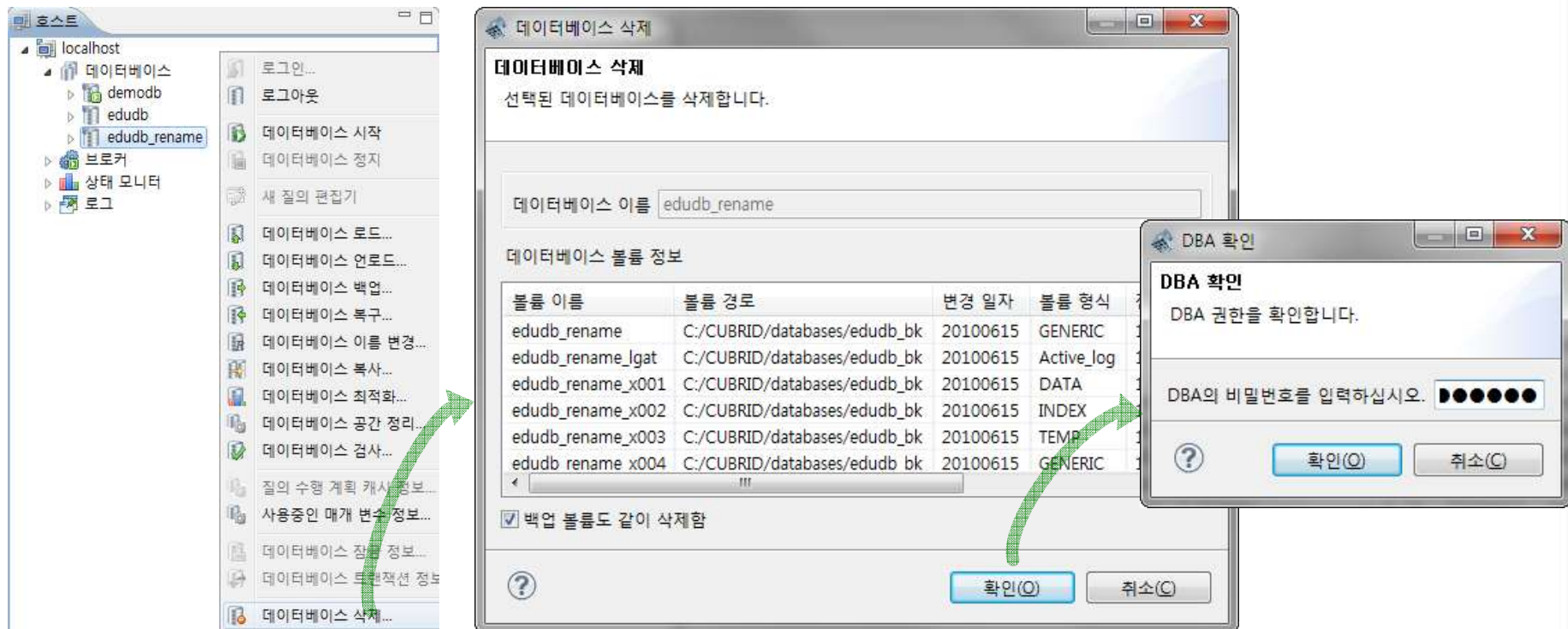
```
% cubrid renamedb edudb_bk edudb_rename
```

➔ 원본 edudb를 경로변경 없이 edudb\_rename으로 이름 변경

옵션	인자	설명	기본값
-E	voext_path	target database 확장 볼륨 경로 지정	DB home directory
-F	vol_tofrom_path	각 볼륨에 대해 복사할 경로 파일로 지정	
-d	vol_tofrom_path	백업 볼륨과 백업 정보 파일 제거	제거 안함

# 데이터베이스 삭제

- stand-alone 모드에서 동작
- 해당 데이터베이스와 관련된 모든 파일을 삭제
  - 백업볼륨은 옵션을 부여하여 삭제



- 명령어 사용 : deletedb

```
% cubrid deletedb [OPTION] database-name
```

```
% cubrid deletedb -d edudb_rename
```

➔ 백업볼륨을 포함하여 삭제

---

## 7. 데이터베이스 환경 설정



- cubrid.conf

- CUBRID 시스템 파라미터의 설정 값을 저장하는 파일이다.
- 위치는 \$CUBRID/conf 아래 존재하며, 데이터베이스 별로 다를 수 있는 내용들은 데이터베이스 별로 별도로 지정하는 것이 좋다.

```
[commom]
max_clients=50
[demodb]
max_clients=100
```

- 서버 파라미터와 클라이언트 파라미터로 나뉘며 파라미터변경을 할 경우 해당하는 프로세스를 재 구동하여야 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 만, 해당 문자열에 공백 문자가 포함된 경우에는 따옴표를 사용한다.





- cubrid.conf의 설정 우선순위 적용

- 환경변수에 설정할 때는 파라미터 앞에 CUBRID\_를 붙여 설정

```
set CUBRID_SORT_BUFFER_PAGE=512
```

- SQL문을 이용한 설정

- ◆ 클라이언트 파라미터만 설정가능
- ◆ 복수 설정을 할 경우 “;”를 이용

```
SET SYSTEM PARAMETERS 'parameter_name=value [{; name=value}...]'
```

```
SET SYSTEM PARAMETERS 'csql_history_num=70'
```

```
SET SYSTEM PARAMETERS 'csql_history_num=70; index_scan_in_oid_order=1'
```

- CSQL 인터프리터를 이용한 설정변경

- ◆ 클라이언트 파라미터만 설정가능

```
csql> ;se block_ddl_statement=1
```

```
=== Set Param Input ===
```

```
block_ddl_statement=1
```



- data\_buffer\_pages
  - 데이터베이스 서버가 메모리 내에 캐시하고 있는 데이터 페이지 개수
  - $\text{num\_data\_buffers} * \text{database page size}$  (데이터베이스 초기화할 때 지정한 페이지 크기로 default 4KB) 만큼 메모리 필요 (default 25,000 설정 시 100MB)
  - 실제 데이터베이스의 크기 및 실제 메모리의 크기, 기타 다른 프로세스의 개수 및 크기를 고려해서 할당해야 한다.
  - 이 값이 클수록 메모리에 많은 데이터가 캐시되므로 disk I/O는 줄일 수 있지만, 너무 크면 페이지 버퍼 폴 스와핑 발생 한다.
- sort\_buffer\_pages
  - 정렬을 필요로 하는 질의를 처리할 때 사용되는 버퍼 페이지의 개수로 generic 볼륨이나 temp 볼륨을 이용한다.
  - Active client request 마다 하나의 sort buffer가 할당된다.
  - 정렬이 끝나면 할당되었던 메모리는 해제된다.
  - 16 ~ 500 page 범위의 값을 권장.
- temp\_file\_memory\_size\_in\_pages
  - 질의에 관한 임시 결과를 캐시하는 버퍼 페이지 개수를 설정
  - 디폴트 값은 4이며, 최대값은 20까지 허용된다.

# 동시성/잠금 관련 설정



- lock\_timeout\_in\_secs
  - 잠금 대기 시간을 지정
  - 시간 이내에 잠금이 허용되지 않으면 해당 트랜잭션이 취소되고 오류가 반환
  - 디폴트 값인 -1로 설정하면 대기 시간이 무제한이고, 0이면 잠금대기가 없다
- isolation\_level
  - 트랜잭션의 고립수준을 1에서 6까지의 정수값 또는 문자 스트링으로 설정
  - SERIALIZABLE: 트랜잭션이 끝날 때 까지 접근 불가
  - REPEATABLE: SELECT에서 S\_LOCK이 트랜잭션 종료될 때 까지 계속 유지
  - READ UNCOMMITTED: 완료되지 않은 트랜잭션에 Read를 허용

설정값	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



- block\_ddl\_statement
  - 데이터 정의문(Data Definition Language, DDL)을 제한
  - default값은 no로 설정하지 않음
- block\_nowhere\_statement
  - UPDATE/DELETE문에 조건(Where)이 없는 경우 질의를 수행하지 않음
  - default값은 no로 설정하지 않음
- intl\_mbs\_support
  - 스키마의 멀티바이트 문자 세트의 지원 여부를 지정(default no 설정하지 않음)
  - 연산 비용이 크므로, 테이블 이름이나 칼럼 이름을 영어로 사용할 것을 권장
- oracle\_style\_empty\_string
  - yes로 설정할 경우 빈 문자열(empty string)을 NULL로 처리(default no)
- single\_byte\_compare
  - 문자열 비교 시 1byte 씩 처리하도록 하며, 유니코드 사용 시(default no)



- 통신 서비스 관련

- cubrid\_port\_id

- ◆ Master Process Port
    - ◆ default 값은 1523
    - ◆ 1523이 이미 사용 중일 경우 이 파라미터를 다른 번호로 변경해야 한다.

- 클라이언트/서버 요청 관련

- max\_clients

- ◆ 서버에 동시 연결 가능한 클라이언트의 최대 개수로써 동시에 수행중인 트랜잭션 전체의 개수를 의미한다.
    - ◆ 따라서, 클라이언트와 연결을 담당하는 server thread의 개수와 같은 값이다.
    - ◆ default 값은 50
    - ◆ 실제 동시 사용자 수 고려

- 서버 재 구동 설정

- auto\_restart\_server

- ◆ 장애로 인하여 서버가 종료된 경우 자동으로 재 구동
    - ◆ default는 yes로 자동 재 구동된다.

---

## 8. CUBRID HA





- 하드웨어, 소프트웨어, 네트워크 등에 장애 발생시에도 지속적인 서비스 제공
- 하루 24시간 1년 내내 서비스를 제공해야 하는 네트워킹 컴퓨팅 부분에서 필수적인 요소
- 두 대 이상의 서버 시스템으로 구성하여 시스템 구성 요소 중의 한 요소에 장애 발생시에도 중단 없는 서비스 제공
- High Availability 기능을 CUBRID에 적용 → CUBRID HA
- CUBRID HA 기능은 여러 서버 시스템에서 데이터베이스를 항상 동기화된 상태로 유지하여 서비스를 제공
- 서비스 수행중인 시스템에 예상치 못한 장애가 발생하였을 경우 자동으로 다른 시스템이 서비스를 수행하도록 하여 서비스 중단 시간 최소화



- shared-nothing 구조
  - 마스터 데이터베이스 서버로부터 슬레이브 데이터베이스 서버로의 데이터 동기화를 위해 다음 두 단계를 수행
    - ◆ 트랜잭션 로그 다중화 단계 : 마스터 데이터베이스 서버에서 생성되는 트랜잭션 로그를 실시간으로 다른 node에 복제
    - ◆ 트랜잭션 로그 반영 단계 : 실시간으로 복제되는 트랜잭션 로그를 분석하여 슬레이브 데이터베이스 서버로 데이터를 반영
- Heartbeat : 시스템과 CUBRID의 상태를 실시간으로 감시하고 장애 발생 시 자동으로 절체(failover)를 수행하기 위해 사용됨





- 마스터 데이터베이스
  - 복제의 대상이 되는 원본 데이터베이스, 읽기/쓰기 등 모든 데이터베이스 연산 허용
- 슬레이브 데이터베이스
  - 마스터 데이터베이스와 동일한 내용의 복제된 데이터베이스(replica)이며, 마스터 데이터베이스의 변경이 자동적으로 반영
  - 마스터 데이터베이스와 달리 슬레이브 데이터베이스는 읽기 연산만 가능
- Active 서버
  - 사용자에게 서비스를 제공하는 서버
  - 마스터 데이터베이스를 사용한 읽기, 쓰기 등 모든 서비스를 제공
- Standby 서버
  - active 서버가 장애로 인해 서비스 불가 시, active 서버를 대신하여 서비스 제공
  - 슬레이브 데이터베이스를 사용한 읽기 서비스 제공

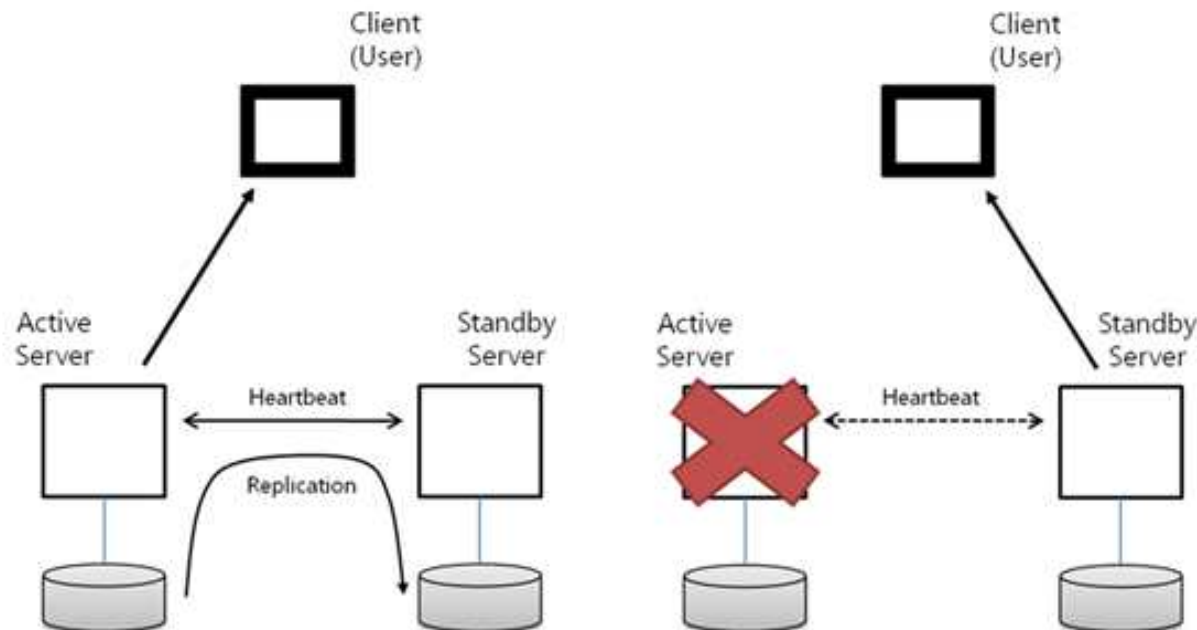


- Failover
  - active 서버 또는 active 서버가 존재하는 시스템에 장애가 발생하여 서비스를 제공할 수 없는 상태가 감출되면, 자동으로 standby 서버를 active 서버로 절체하여 서비스를 계속 제공하는 기능
- Failback
  - failover 후 active 서버가 장애 이전 상태로 복구되면, 복구된 active 서버를 통해 서비스가 수행되도록 원래 상태로 다시 절체되는 기능
- Role change
  - 이전 active 서버에서 장애가 복구되어도 현재의 상태로 서비스를 계속 제공하는 기능



- Heartbeat
  - HA 기능을 제공하기 위한 핵심 구성 요소
  - CUBRID Heartbeat 기능은 cub\_master 프로세스 내에 포함
    - ◆ 다른 노드의 cub\_master 프로세스와 heartbeat 메시지를 주고 받으며 노드 장애 감지
    - ◆ 장애 감지시 standby 서버로 failover 수행
  - HA 관련 프로세스(cub\_server, copylogdb, applylogdb)의 가용 상태를 주기적으로 모니터링

- 서버 이중화(Duplex, Duplicate)
  - HA기능 제공 위해 서버 HW를 중복으로 구성하여 시스템을 구축
  - Active-Standby : 장애 발생시 stand-by 서버가 active 서버의 기능을 수행
  - Active-Active : 장애 발생 서버의 역할을 추가로 대행하면서 서비스 기능을 수행하도록 이중 체계를 구축





- 데이터베이스 서버 다중화 구조
  - 하나의 데이터베이스에 장애가 발생하여도 서비스를 계속 제공할 수 있도록 여러 대의 데이터베이스 서버를 구성한 구조
  - 서비스를 제공 중이던 active 데이터베이스 서버에 장애 발생시, 동일한 데이터를 가지고 있는 standby 데이터베이스 서버가 서비스를 제공
- 트랜잭션 로그 다중화(Transaction Log Multiplication)
  - active 서버에서 생성되는 트랜잭션 로그가 하나 이상의 standby 서버에 실시간으로 전송하여 모든 서버에 동일한 로그가 기록되도록 하는 기능



- 브로커 다중화 구조

- 브로커를 다중화 하여 특정 브로커에 장애가 발생하여도 다른 브로커를 통한 서비스를 계속 제공할 수 있도록 구성한 구조
- 각 브로커별로 다른 특성을 가질 수 있음
  - ◆ Read-only broker : read only 브로커는 읽기 연산만을 수행하는 브로커
    - ▶ standby 서버와의 연결을 통해 서비스를 제공
    - ▶ standby 서버와 연결 실패시 active 서버로 읽기 요청을 보낼 수 있음
  - ◆ Slave-only broker : read-only broker와 달리 standby 서버로만 연결
    - ▶ standby 서버가 존재하지 않는 경우 active 서버로 연결을 시도하지 않음



- CUBRID 데이터베이스 서버의 HA 모드
  - **active** : 일반적인 읽기 및 쓰기 요청에 대한 서비스 제공
    - ◆ 복제에 필요한 트랜잭션 로그를 생성하는 상태
  - **to-be active** : standby 상태에서 active 모드 넘어가기 전의 상태
    - ◆ 유입되는 요청을 중지(suspend)하고 반영되지 않은 복제 로그를 반영한 후 active 상태로 변경됨
  - **standby** : 읽기 요청에 대한 서비스만 제공하는 상태
    - ◆ 쓰기 요청은 거부(deny)
  - **to-be standby** : active 상태에서 standby로 모드로 넘어가기 전의 상태
    - ◆ 유입되는 요청을 거부(deny)하고 수행 중인 트랜잭션이 완료되면 standby 상태로 변경됨
- failover 시 데이터베이스 서버의 상태 변화
  - active 서버: active → dead
  - standby 서버: standby → to-be-active → active

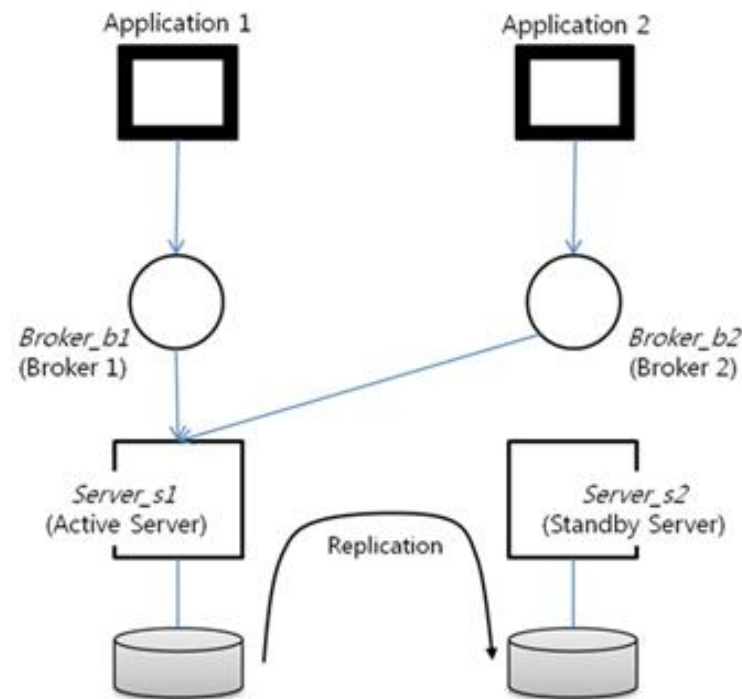
# HA 제약 사항

- Linux 계열에서만 CUBRID HA 기능을 사용 가능
- active 서버와 standby 서버는 반드시 동일 플랫폼으로 구성
- 기본 키(primary key)가 설정된 테이블에 대해서만 복제 가능
- 다음의 경우 HA 구성에서 마스터 데이터베이스와 슬레이브 데이터베이스 간 데이터 불일치 발생 가능
  - 테이블에 trigger 또는 Java Stored Procedure가 설정된 경우 : 슬레이브 데이터베이스에서 중복 수행
  - 테이블에 method가 사용된 경우 : 메소드 사용 시 복제 로그가 생성되지 않아 복제되지 않음
  - CUBRID 매니저를 이용하여 컬럼에 NOT NULL 옵션을 설정하는 경우 : 복제 로그가 생성되지 않음
  - 독립 실행 모드(standalone)에서 작업이 수행된 경우 : 작업 관련 로그가 슬레이브 데이터베이스로 반영되지 않음



# HA 구성

- HA 구성을 위해 active 서버 및 standby 서버 모두 CUBRID heartbeat 기능 활성화
  - 환경 변수 설정 참고
- 브로커와 데이터베이스 서버를 하나의 장비 또는 분리된 시스템에 구성 가능
- HA 기능을 사용하기 위해서 최소한 물리적으로 분리된 2개 이상의 서버 필요





- 노드 그룹 식별자 생성
  - active 및 standby 서버에 동일한 사용자 계정(예: ha\_user1) 생성
    - ◆ 해당 계정에서 CUBRID 실행
    - ◆ 생성된 UNIX 사용자 계정이 HA 모드로 구동될 서버 노드 그룹 식별자
- 데이터베이스 복사
  - active 서버에 생성한 데이터베이스를 standby 서버로 복사
    - ◆ 데이터베이스 이름은 동일해야 함
    - ◆ 데이터 볼륨 복사, 백업/복구, unloaddb/loaddb 중 하나의 방법 이용
    - ◆ 구조가 같을 필요는 없으나, 볼륨 복사의 방법이 간편
      - ▶ HA 모드 구동 전에 진행하여야 함
      - ▶ 볼륨 복사시 데이터베이스 서버 중단 후 `csql -S <dbname>` 접속/종료 후 복사

- 서버 설정(cubrid.conf)

- cubrid.conf 설정 파일에 HA 관련 파라미터를 추가

- ◆ HA 관련 파라미터는 반드시 [common] 섹션에 설정
    - ◆ HA 모드로 구동하지 않을 데이터베이스에 대해서만 [@<database>] 섹션에서 ha\_mode의 값을 no로 설정
    - ◆ 만약, [common] 섹션에 ha\_mode의 값이 no이고, [@<database>] 섹션에 ha\_mode의 값이 yes인 경우 에러



- ◆ **ha\_mode** : HA 기능을 설정, 디폴트 값은 off
  - ▶ HA 사용을 위해 on으로 설정
- ◆ **ha\_port\_id** : 장애 감지를 위해 heartbeat 메시지 통신 포트(UDP)
- ◆ **ha\_node\_list** : HA 모드로 구동되는 노드 그룹 식별자 및 그룹에 속하는 멤버 노드의 호스트명을 명시
  - ▶ 명시된 멤버 노드의 호스트명 및 현재 노드의 호스트명은 /etc/hosts에 등록되어 있어야 함.
  - ▶ 아래 : ha\_user1이라는 노드 그룹의 멤버로 active 서버의 호스트명(server\_s1)과 standby 서버의 호스트명(server\_s2)을 명시

```
#cubrid.conf
[common]
ha_mode = on
ha_port_id = 41523
ha_node_list = ha_user1@server_s1:server_s2
```

- HA 구동 스크립트(cubrid-ha) 설정
  - active 및 standby 데이터베이스 서버 모두 설정
  - 위치 : \$CUBRID/share/init.d
    - ◆ CUBRID\_USER : 노드 그룹 식별자, 즉 CUBRID를 설치 및 실행한 UNIX 사용자 계정
    - ◆ DB\_LIST : HA 모드로 구동할 데이터베이스 이름을 명시
      - ▶ 한 개 이상의 데이터베이스 이름을 명시하는 경우, 공백으로 구분

```
#cubrid-ha
```

```
CUBRID_USER = ha_user1
```

```
DB_LIST = 'tdb01'
```

```
# DB_LIST = 'tdb01 tdb02 tdb03'
```

- 브로커 설정

- 기본 동작 모드 : 읽기 및 쓰기 연산을 요청

- 동작 모드 변경

- ◆ cubrid\_broker.conf의 ACCESS\_MODE 변경

```
#cubrid_broker.conf
```

```
ACCESS_MODE = RW |RO |SO
```

```
RW := Read-Write broker (기본값)
```

```
RO := Read-Only broker
```

```
SO := Slave-Only broker
```

- 브로커의 동작 모드 확인

- ◆ cubrid broker status -f : ACCESS\_MODE항목으로 확인

- 데이터베이스 호스트 정보 설정

- databases.txt

- ◆ 브로커 및 데이터베이스 서버 모두 설정

- ▶ HA 구성에서 사용할 Active 서버 및 Standby 서버의 호스트 정보 추가

- CUBRID 응용 사용시 호스트 이름 지정

- ◆ csq1 -u dba demodb@server\_s1

- ◆ backupdb -u dba demodb@server\_s2

```
#databases.txt
```

```
#db-name      vol-path      db-host      log-path
```

```
tdb01         /home/db/db2  server_s1:server_s2  /home/db/db2
```

- cubrid-ha start
  - HA 모드로 데이터베이스 서버를 구동
    - ◆ cub\_master, cub\_server, copylogdb, applylogdb 프로세스 순서대로 구동
    - ◆ active server 에서 먼저 구동하여야 함. cubrid-ha start 가 먼저 수행된 서버가 active 서버로 설정됨.

```
[ha_user1@server_s1 ~]# $CUBRID/share/init.d/cubrid-ha start
Starting cubrid-ha: [ OK ]
```

```
[ha_user2@server_s2 ~]# $CUBRID/share/init.d/cubrid-ha start
Starting cubrid-ha: [ OK ]
```





- cubrid-ha stop

- HA 모드로 구동 중인 데이터베이스 서버 종료

- ◆ applylogdb, copylogdb, cub\_server, cub\_master 프로세스 순서대로 종료
    - ◆ standby 서버 부터 중단.
    - ◆ active에서 먼저 수행할 경우 failover 발생

```
[ha_user2@server_s2 ~]# $CUBRID/share/init.d/cubrid-ha stop
Stopping cubrid-ha: [ OK ]
```

```
[ha_user1@server_s1 ~]# $CUBRID/share/init.d/cubrid-ha stop
Stopping cubrid-ha: [ OK ]
```

- cubrid-ha status

- HA 모드로 구동 중인 데이터베이스 서버의 상태 확인
- HA 노드의 상태 정보 및 HA 관련 프로세스의 상태 정보 출력

```
[ha_user2@server_s2 ~]# service cubrid-ha status
```

```
HA-Node Info (current server_s2, state slave)
```

```
Node server_s2 (priority 2, state slave)
```

```
Node server_s1 (priority 1, state master)
```

```
HA-Process Info (master 30519, state slave)
```

```
Applylogdb tdb01@localhost:/home1/cubrid1/DB/tdb01_server_s1 (pid 30796, state registered)
```

```
Copylogdb tdb01@server_s1:/home1/cubrid1/DB/tdb01_server_s1 (pid 30788, state registered)
```

```
Server tdb01 (pid 30551, state registered)
```

```
++ cubrid heartbeat list: success
```

```
Status cubrid-ha: [ OK ]
```

## ● 복제 반영 상태 모니터링

- applylogdb 유틸리티가 복제 로그를 반영할 때마다 그 진행 상태가 db\_ha\_apply\_info 라는 시스템 테이블에 저장된다. 이 테이블은 applylogdb 유틸리티가 커밋하는 시점마다 갱신되며, \*\_counter 컬럼에는 수행 연산의 누적 카운트 값이 저장된다. 각 컬럼의 의미는 다음과 같다.

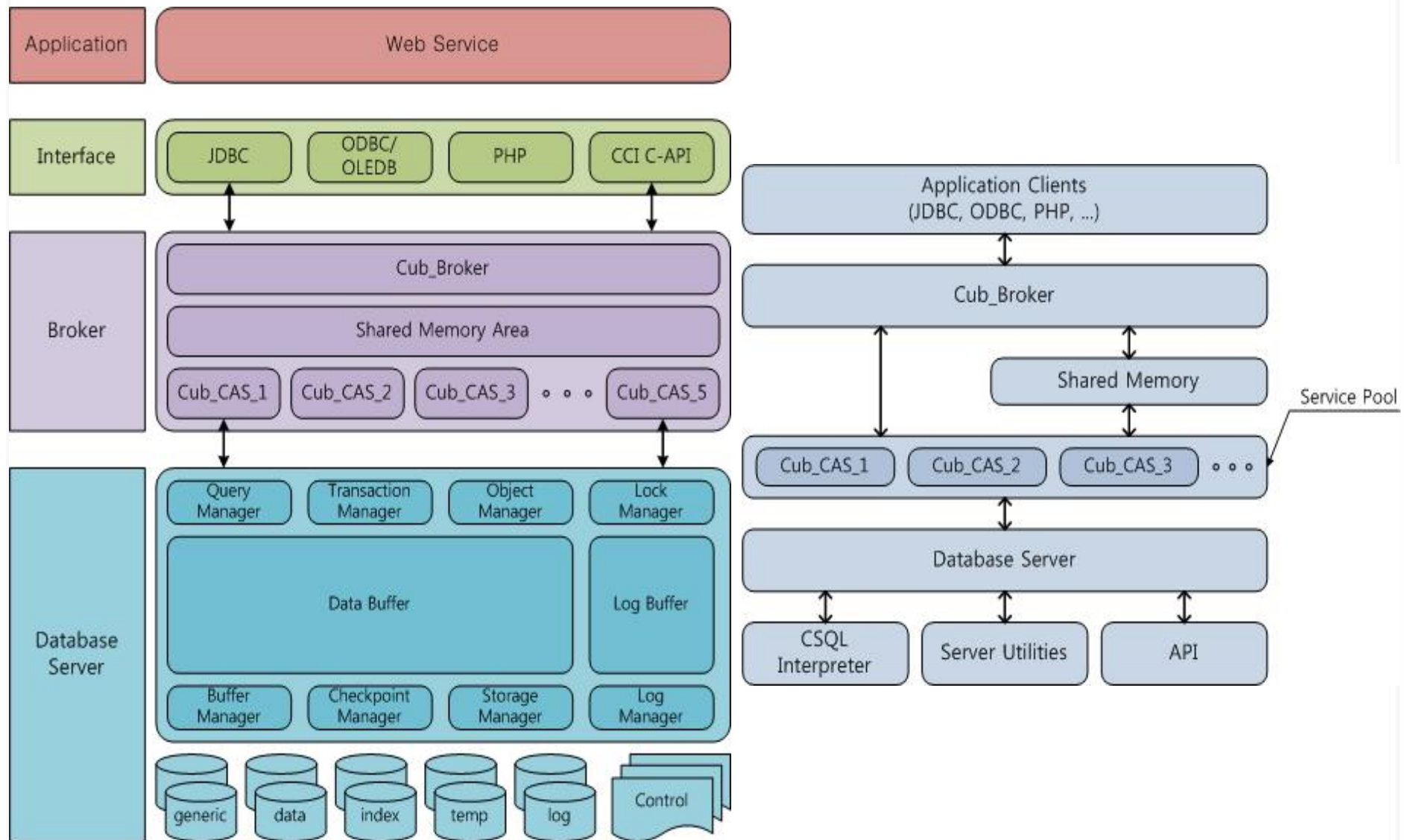
컬럼명	컬럼 타입	
db_name	VARCHAR(255)	로그에 저장된 DB 이름
db_creation_time	DATETIME	반영하는 로그에 대한 원본 DB의 생성 시각
copied_log_path	VARCHAR(4096)	반영하는 로그 파일의 경로
page_id	INTEGER	슬레이브 DB에 커밋된 복제 로그의 page
offset	INTEGER	슬레이브 DB에 커밋된 복제 로그의 offset
log_record_time	DATETIME	슬레이브 DB에 커밋된 복제 로그에 포함된 timestamp, 즉 해당 로그 레코드 생성 시간
last_access_time	DATETIME	applylogdb가 슬레이브 DB에 commit 한 시각
insert_counter	BIGINT	applylogdb가 insert한 횟수
update_counter	BIGINT	applylogdb가 update한 횟수
delete_counter	BIGINT	applylogdb가 delete한 횟수
schema_counter	BIGINT	applylogdb가 schema를 변경한 횟수
commit_counter	BIGINT	applylogdb가 commit한 횟수
fail_counter	BIGINT	applylogdb가 insert/update/delete/commit/schema 변경 중 실패 횟수
required_page_id	INTEGER	applylogdb가 읽을 수 있는 최소 pageid
start_time	DATETIME	applylogdb 프로세스가 슬레이브 DB에 접속한 시간
status	INTEGER	반영 진행 상태 (0: IDLE, 1: BUSY)

---

## 9. CUBRID BROKER



# 구조





- Broker

- 클라이언트의 요청을 응용서버로 전달하고, 응용서버가 처리한 결과를 클라이언트로 보냄
- ODBC, JDBC등 데이터베이스 연결 시 Broker를 통해 관리
- Broker
  - ◆ 응용서버 CAS 를 관리
- Shared memory
  - ◆ 응용서버와 브로커의 정보(현재상태, 요청처리건수 등)을 관리

- Service pool

- 클라이언트의 요청을 처리하는 응용서버의 그룹들
- Broker로부터 요청을 받으면 데이터베이스로 질의를 하거나, 스크립트의 번역을 수행

- 구동

- CUBRID가 설치되어 있는 호스트의 브로커 구동
- 큐브리드 service 구동 시 자동으로 구동

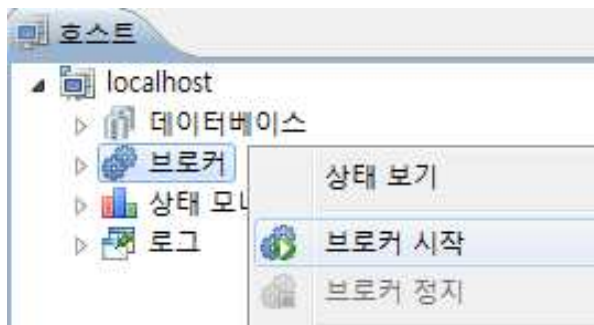
```
% cubrid broker start
```

```
@ 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
- 큐브리드 service 구동 시 자동으로 종료

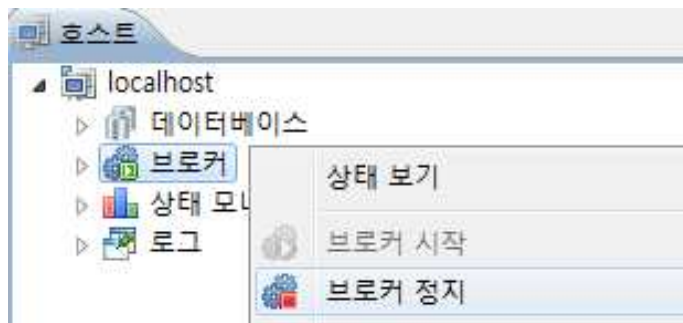
```
% cubrid broker stop
```

```
@ cubrid broker stop
```

```
++ cubrid broker stop: success
```

➔ 이미 구동되어 있을 경우

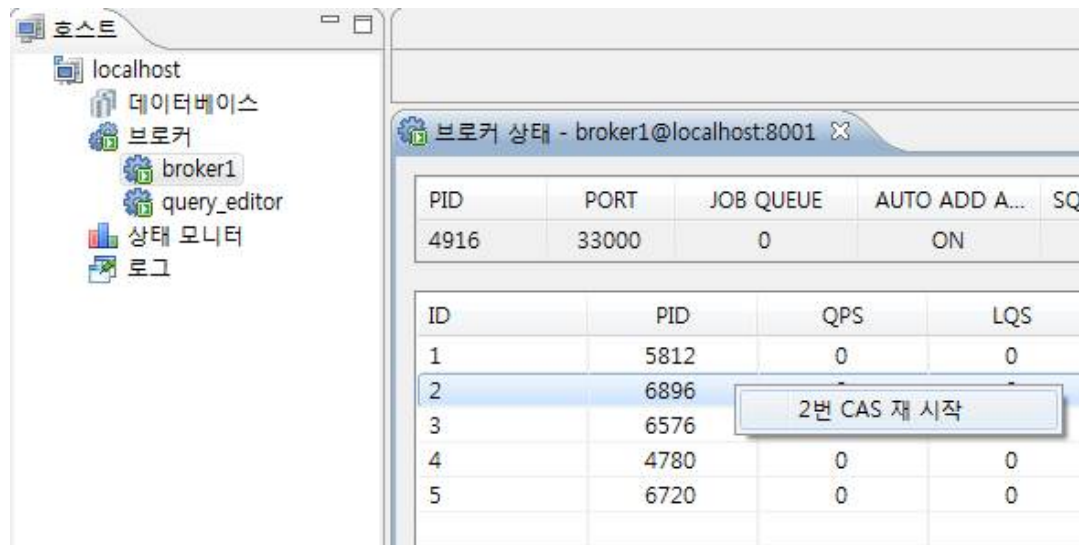
```
++ cubrid broker is not running.
```



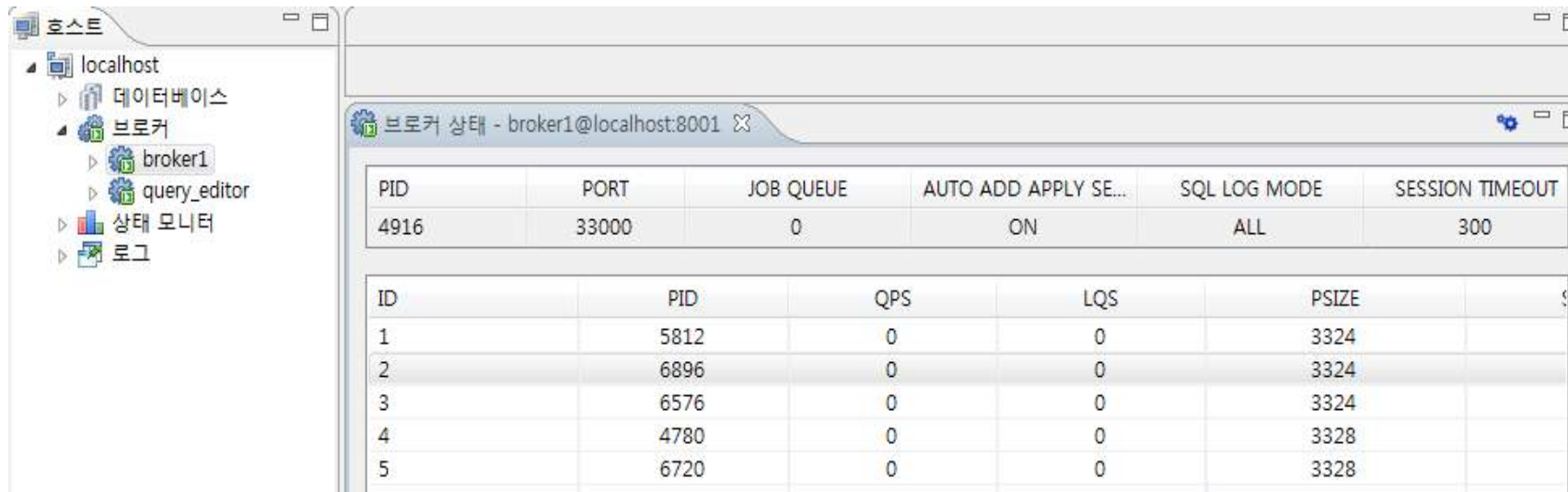


- 응용서버(CAS) 재 시작

- 서비스 중이거나 개발 중 일때 어느 응용서버가 장시간 BUSY 인 상태로 있을 수 있음
- 이 경우 다른 서비스에 영향을 줄 수도 있으므로 해당 서비스만을 강제로 종료시킨 후 재 구동 시킬 필요가 있음
  - ◆ 재 시작할 응용서버의 번호(응용서버 ID)를 선택 후, 확인을 click



- monitoring
  - CUBRID BROKER의 현재 요청 처리상태를 보여줌



PID	PORT	JOB QUEUE	AUTO ADD APPLY SE...	SQL LOG MODE	SESSION TIMEOUT
4916	33000	0	ON	ALL	300

ID	PID	QPS	LQS	PSIZE
1	5812	0	0	3324
2	6896	0	0	3324
3	6576	0	0	3324
4	4780	0	0	3328
5	6720	0	0	3328



## ● 명령어 이용 : broker status

```
% cubrid broker status options [broker_name]  
options : [ -b | -q | -s secs | -t | -f ]
```

- -b : 브로커 정보만 출력
- -q : job queue에 대기중인 job의 정보 출력
- -s <sec> : 지정한 간격으로 모니터링 정보를 refresh 함
- 종료는 'q'
- -t : 출력내용을 파일로 저장

```
C:\Documents and Settings\Administrator>cubrid broker status  
@ cubrid broker status  
% query_editor -cub_cas [3288,30000] C:\WCUBRID\log/broker/query_editor.access C:\WCUBRID/  
JOB_QUEUE:0, AUTO_ADD_APPL_SERVER:ON, SQL_LOG_MODE:ALL:100000  
LONG_TRANSACTION_TIME:60, LONG_QUERY_TIME:60, SESSION_TIMEOUT:300  
KEEP_CONNECTION:AUTO, ACCESS_MODE:RW
```

ID	PID	QPS	LQS	PORT	PSIZE	STATUS	CPU
1	4028	0	0	30001	2500	IDLE	0.00
2	2712	0	0	30002	2500	IDLE	0.00
3	2128	0	0	30003	2500	IDLE	0.00
4	1392	0	0	30004	2496	IDLE	0.00
5	3992	0	0	30005	2500	IDLE	0.00



## ● Broker Status

값	설 명
ID	응용 서버 ID
PID	응용서버의 process ID
QPS	초당 처리한 질의의 수
LQS	초당 처리되는 장기수행 질의의 수
PSIZE	프로세스 크기
STATUS	응용서버의 현재 상태 •IDLE : 요청을 받기 위한 대기 상태 •BUSY : 요청을 처리하고 있음 •CLIENT WAIT : 처리중은 요청은 없으나, 트랜잭션이 끝나지 않은 상태로 클라이언트의 요청을 기다리는 상태 •CLOSE WAIT : 트랜잭션이 끝나고 클라이언트의 연결 종료를 기다리는 상태
DB	접속중인 데이터베이스 이름
HOST	접속중인 서버 이름
LAST ACCESS TIME	가장 최근의 요청 처리 시간
LAST CONNECTION TIME	가장 최근에 데이터베이스와 연결을 맺은 시간



- broker 상태 정보 갱신

- 브로커의 상태를 주어진 시간(초) 간격으로 갱신하여 확인
- 명령어 이용

% cubrid broker status -s 1 broker1

➔ broker1에 대한 정보 1초 간격으로 갱신

% cubrid broker status -s 1

➔ broker이름을 명시하지 않을 경우 모든 broker의 정보를 1초 간격으로 갱신



- 응용서버그룹(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 별도의 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
APPL_SERVER_MAX_SIZE    =20
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
```

```
# 추가할 내용
[% MY_BROKER]
SERVICE                =ON
BROKER_PORT              =40000
MIN_NUM_APPL_SERVER     =5
MAX_NUM_APPL_SERVER     =10
APPL_SERVER_SHM_ID      =40000
APPL_SERVER_MAX_SIZE    =20
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
```

- 환경 설정 변경

- 설정 파일 : \$CUBRID/conf/cubrid\_broker.conf
- 편집기를 통하여 수정가능하며, Broker 재 구동될 경우 설정 적용
- 재 구동 없이 설정 변경할 경우 명령어를 이용하여 변경

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

- 설정 가능 환경변수

- ◆ APPL\_SERVER\_MAX\_SIZE, SQL\_LOG, TIME\_TO\_KILL
- ◆ SESSION\_TIMEOUT, KEEP\_CONNECTION
- 환경변수 및 설정 값이 잘못되어 있을 경우 재 구동 시 오류가 발생하며 구동되지 않는다.



Parameter 이름	설 명	값
MASTER_SHM_ID	Broker의 전체적인 운영을 위해 필요한 공유 메모리 번호를 지정하는 파라미터 %% 문자 뒤에 지정해야 하며 시스템에서 고유한 값이어야 한다.	30001 (int)
ADMIN_LOG_FILE	admin log file(CUBRID broker start/stop/restart 등을 기록)의 위치와 이름을 지정	log/admin.log (char)
<Broker 이름> (Broker name)	응용서버그룹(Broker)의 이름을 지정하며, 고유한 값으로 지정해야 한다. % 뒤에 지정하며, 대소문자를 구분하지 않음	BROKER1 (char)
SERVICE	CUBRID Broker의 구동시 응용서버그룹의 구동 여부를 지정한다	ON / OFF
BROKER_PORT (Broker Port)	Broker의 port 번호를 지정한다. 시스템에서 유일한 값이어야 하며, 이 port를 이용하여 응용프로그램(JDBC, ODBC 등)과 통신한다. 방화벽이 구성되어 있을 경우 설정된 포트를 open하여야 한다. Windows에서 Broker의 CAS포트는 "Broker포트+1" 순서로 생성된다.	30000 (int)





Parameter 이름	설 명	값
APPL_SERVER_SHM_ID	Broker 와 해당 그룹의 응용서버들이 사용하는 공유메모리의 key 값	30000 (int)
MIN_NUM_APPL_SERVER (AS Minimum)	응용서버의 초기 개수(처음 구동시 구동되는 개수)를 지정	5 (int)
MAX_NUM_APPL_SERVER (AS Maximum)	응용서버가 증가할 수 있는 최대 개수를 지정	40 (int)
APPL_SERVER_MAX_SIZE	응용서버의 허용가능한 프로세스 크기를 MByte 단위로 지정	20 (int)
LOG_DIR	access log 가 기록되는 디렉토리 지정	log/broker (char)
ERROR_LOG_DIR	error log 가 기록되는 디렉토리 지정	log/broker/error_log (char)
SQL_LOG	응용서버가 처리하는 transaction 에 대하여 기록을 남길지 여부를 지정	ON / OFF



Parameter 이름	설 명	값
TIME_TO_KILL	응용서버의 초기 개수를 초과하여 증가된 응용서버가 해당 시간(초)동안 요청을 받지않는 경우 자동 종료시킨다	60 (int)
SESSION_TIMEOUT	transaction 처리를 위하여 요청을 받게되는 경우 transaction 을 시작하여 transaction 의 끝(commit/rollback) 까지 하나의 응용서버에서 계속 처리하게 되는데 이때 해당 시간(초) 동안 요청이 없는 경우 강제로 transaction 을 종료(rollback) 시킨다 -1 일 경우 강제 종료시키지 않고 무한정 기다린다.	300 (int)
KEEP_CONNECTION	CAS와 클라이언트간의 연결방식을 지정 ON은 클라이언트 접속단위로 연결(connection pool) OFF일 경우 클라이언트의 트랜잭션 단위로 연결(로드밸런싱) AUTO일 경우 CAS수가 클라이언트 수보다 많을 경우 ON과 같이 동작하고, 클라이언트가 많을 경우 OFF와 같이 동작한다.	ON/OFF/ AUTO
STATEMENT_POOLING	아파치 DBCP에서 statement pooling사용에 따른 CAS지원 여부를 결정. ON설정 시 commit 시점에 CAS의 모든 핸들을 재사용	ON / OFF



Parameter 이름	설 명	값
MAX_STRING_LENGTH	문자형 데이터 타입에 대한 최대 허용 문자길이를 설정한다. 100으로 설정되어 있는 상태에서 입력길이가 1,000일 경우 첫 100byte만 허용 하고 이하 버림을 수행한다.	-1 (int)
SELECT_AUTO_COMMIT	PHP(CCI포함)에서 SELECT문에 대한 자동 commit 모드를 수행	ON/OFF
LONG_QUERY_TIME	장기 실행 질의(Long-duration query)로 판단될 질의 실행 시간을 설정 값이 0일 경우 수행하지 않음	60000 (msec)
LONG_TRANSACTION_TIME	장기 실행 트랜잭션(Long-duration transaction)으로 판단될 트랜 잭션의 실행 시간을 설정 값이 0일 경우 수행하지 않음	60000 (msec)

## ● 접속로그 확인

- Broker 별로 각 응용서버가 요청을 받아 처리한 시간에 대한 기록이  
며, r.conf 의 Broker 파라미터 ACCESS\_LOG의 값이 ON인 상태  
이어야 한다.
- <broker 이름>.access 라는 이름으로 만들어진다.

```
1 192.168.100.201 --1158198049.151 1158198049.246 2010/06/14 10:40:49 ~2010/06/14 10:40:49 29438 --1
2 192.168.100.201 --1158198049.401 1158198049.406 2010/06/14 10:40:49 ~2010/06/14 10:40:49 29438 --1
```

값	설 명
1	응용서버 ID
192.168.1.201	요청을 보낸 client 의 IP address
1158198049.151 1158198049.246	요청을 받은 시간과 요청을 끝낸 시간에 대한 timestamp
2010/06/14 10:40:49 ~2010/06/14 10:40:49	요청을 받은 시간과 요청을 끝낸 시간
29438	응용서버의 process ID
--1 or ERR 1025	--1 인 경우 에러 없이 처리 에러발생 시 ERR 하며 에러로그의 offset명시



- 에러 로그 확인

- 응용 클라이언트의 요청을 처리하는 도중에 발생한 에러에 관한 정보를 broker\_name\_app\_server\_num.err의 파일에 기록

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

값	설 명
Time: 02/04/10 13:45:17.687	에러 발생 시점
SYNTAX ERROR	에러의 종류
ERROR CODE = -493	에러 코드
Tran = 1	트랜잭션 ID
EID = 38	에러 ID SQL 문 처리 중 에러가 발생한 경우, 서버나 클라이언트 에러 로그와 관련이 있는 SQL 로그를 찾을 때 사용함
Syntax:...	에러 메시지



## ● SQL 로그

- SQL 로그 파일은 응용 클라이언트가 요청하는 SQL을 기록하며, broker\_name\_app\_server\_num.sql.log라는 이름으로 저장된다.

```
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 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) auto_rollback 0
*** 0.000
```

```
02/04 13:45:17.687 (39) prepare 0 select * from unique_tbl
02/04 13:45:17.687 (39) prepare srv_h_id 1 (PC)
02/04 13:45:17.687 (39) execute srv_h_id 1 select * from unique_tbl
02/04 13:45:17.687 (39) execute 0 tuple 1 time 0.000
02/04 13:45:17.687 (0) auto_commit
02/04 13:45:17.687 (0) auto_commit 0
*** 0.000
```

- 응용 클라이언트의 요청 시각
- (39) : SQL 문 그룹의 시퀀스 번호  
prepared statement pooling일 경우
- prepare 0 : prepared statement인지 여부
- (PC) : 플랜 캐시에 저장되어 있는 내용을 사용
- SELECT... : 실행 SQL 문.  
-Statement pooling한 경우,  
WHERE 절의 binding 변수가 ?로 표시된다.
- Execute 0 tuple 1 time 0.000  
-1개의 row가 실행되고, 소요 시간은 0.000초
- auto\_commit/auto\_rollback  
-자동으로 커밋 되거나, 롤백 되는 것을 의미  
-두 번째 auto\_commit/auto\_rollback은  
에러 코드이며, 0은 에러가 없이 트랜잭션이 완료

- SQL 로그 분석

- Broker\_log\_top

- ◆ SQL log를 분석하여, 수행된 질의별 또는 트랜잭션별 수행 속도별 정렬

```
% broker_log_top [-t] [-F MM/DD] [-T MM/DD] <log file> ...
```

```
# 로그를 분석하여 트랜잭션 수행 속도별로 정렬
```

```
% broker_log_top -t broker_1.sql.log → log_top.t 생성
```

```
# 로그를 분석하여 질의 수행 속도별로 정렬
```

```
% broker_log_top broker_1.sql.log → log_top.res, log_top.q 생성
```

- Log\_top.t

```
broker_www_40.sql.log:458
02/09 21:48:44.139 (0) connect 127.0.0.1
02/09 21:48:44.139 (0) connect zbxz zbxz
02/09 21:48:44.139 (0) isolation level : 3, lock timeout : -1, auto_commit : false
02/09 21:48:44.140 (0) get_version
02/09 21:48:44.141 (82) prepare 0 select site_srl from xe_sites
02/09 21:48:44.141 (82) prepare srv_h_id 1
...
02/09 22:18:28.562 (0) SESSION_TIMEOUT
02/09 22:18:28.563 (0) disconnect
*** 1784.424
```

- ◆ log\_top.res : 질의별 수행시간이 가장 오래 걸린 순서로 최대, 최소, 평균을 보여줌

	max	min	avg	cnt(err)
[Q1]	2:44.360	1:10.360	1:44.360	12 (0)
[Q2]	1:01.174	33.174	42.174	33 (0)
[Q3]	56.926	32.926	41.926	21 (1)
[Q4]	6.027	4.027	5.027	21 (0)

- ◆ Log\_top.q : 질의 목록

```
[Q1]-----
broker_www_40.sql.log:495
02/09 21:48:44.185 (88) execute srv_h_id 1 update "xe_documents" as documents set
    "readed_count" = readed_count+1 where ("document_srl" = 32186)
02/09 22:13:28.545 (88) execute 0 tuple 1 time 1484.360

[Q2]-----
broker_www_40.sql.log:876
02/09 22:23:29.570 (155) execute srv_h_id 1 update "xe_documents" as documents set
    "readed_count" = readed_count+1 where ("document_srl" = 32189)
02/09 22:33:30.744 (155) execute 0 tuple 1 time 601.174
```



감 사 합 니 다.

보다 나은 교육을 위하여 설문 작성을 부탁드립니다.  
[www.cubrid.com/edu\\_survey.php](http://www.cubrid.com/edu_survey.php)

