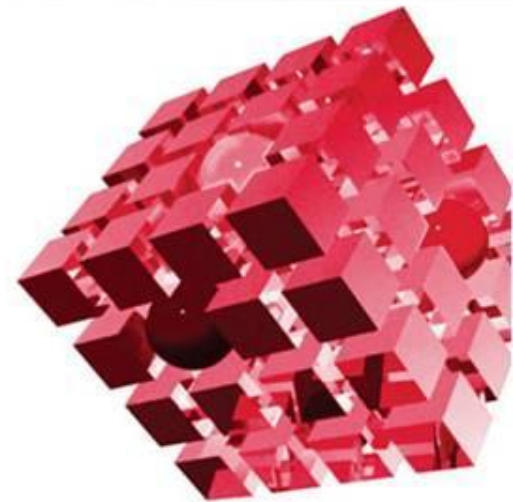


**CUBRID**  
Driving the DBMS **REVOLUTION**

# 개발자 과정

기술컨설팅팀

**CUBRID**



v20081201



1. CUBRID 살펴보기
2. CUBRID 설치하기
3. 데이터베이스 설계
4. 데이터베이스 생성
5. 스키마 관리
6. 데이터 검색 및 조작
7. 연산자와 함수
8. 트리거
9. 분할
10. 자바 저장 프로시저

# 1. CUBRID 살펴보기



CUBRID



- 2008.11 오픈 소스 CUBRID 2008 발표
- 2008.10. NHN 서치솔루션 CUBRID 인수
- 2007.10. CUBRID 7.3 출시
- 2007.04. CUBRID 7.0 출시
- 2006.07. XDBMS 공동개발 계약(NHN)
- 2006.06. CUBRID 6.5 출시 및 무료선언
- 2006.03. 신NEIS 시스템 개통 (CUBRID)
- 2006.03. CUBRID 6.4 출시
- 2006.02. CUBRID 설립. UniSQL ⇨ CUBRID 명칭 변경
- 2003. 01. UniSQL 6.0 출시
- 2001. 10. UniSQL 5.0, UniCAS 4.5 출시
- 2000. 12. 통합 애플리케이션 서버 UniCAS 4.0 출시
- 1999. 07. UniSQL 4.0K 출시
- 1997. 01. 3계층 클라이언트/서버 미들웨어 Vision3 개발
- 1996. 09. 웹 애플리케이션 서버 UniWEB 개발

1988. 02. 한국컴퓨터통신(주) 설립



- RDBMS 계승
  - 성능과 안정성
  - 보편성 및 확장성
  - SQL-2 (ANSI SQL 92)
  - Default, not null, unique, primary/foreign key
  - View
  - Trigger
- 대용량
  - 멀티볼륨 DB
  - DB 개수/크기, 테이블 개수/크기 무제한
  - 컬럼개수:6,400, 컬럼크기:2GB, 색인개수:6,400
- 확장성
  - 멀티 볼륨 DB – 볼륨 추가
  - 멀티 스레드 서버 – 멀티 CPU 사용 최적화
  - 복제(Replication) – 서버 추가
  - 분할(Partition) – 데이터 확장 대처



- 고성능 서버 구조
  - 멀티 쓰레드, 멀티 서버
  - Cost Based Optimizer (CBO)
  - 질의 플랜 캐쉬
  - 복제(Replication)
    - 질의 분산
  - 분할(Partition)
    - 질의 분할 최적화
- 고성능 클라이언트 구조
  - Broker 미들웨어 포함
    - 쓰레드 풀 관리
    - 자동 부하 최적화 기능
  - 객체 메모리 캐쉬



## ● 트랜잭션

- 완벽한 트랜잭션 ACID 보장: commit, rollback, savepoint
- 시스템/DB 장애시 트랜잭션 일치성 보장
- 복제간 트랜잭션 일치성 보장
- 다중 단위 잠금: DB, 테이블, 객체(레코드)
- 교착상태(deadlock) 자동 해결
- 분산 트랜잭션 지원: Java, C-API

## ● 백업/복구

- 트랜잭션 일치성 보장
- On-line/off-line 백업 지원
- 3단계 백업 레벨 지원: full, incremental-1, incremental-2
- 장애발생시점 또는 특정시점에서의 복구 지원
- 병렬 백업/복구 지원
- 실시간 압축 백업 지원
- 다양한 3rd-party 백업 시스템과의 연동



- 보안 및 권한 관리
  - 사용자 권한 관리: 사용자, 그룹, 개체, 권한
  - 트랜잭션 별로 사용 권한 설정 가능
- 풍부한 개발 환경
  - 표준 API: JDBC, ODBC, OLEDB, PHP, Embedded-SQL 등
  - Native API: CCI-API, C-API
  - Java Stored Procedure
- 큐브리드 매니저
  - 플랫폼과 무관하게 사용(Java로 개발)
  - 관리, 질의, 진단, 튜닝 통합도구

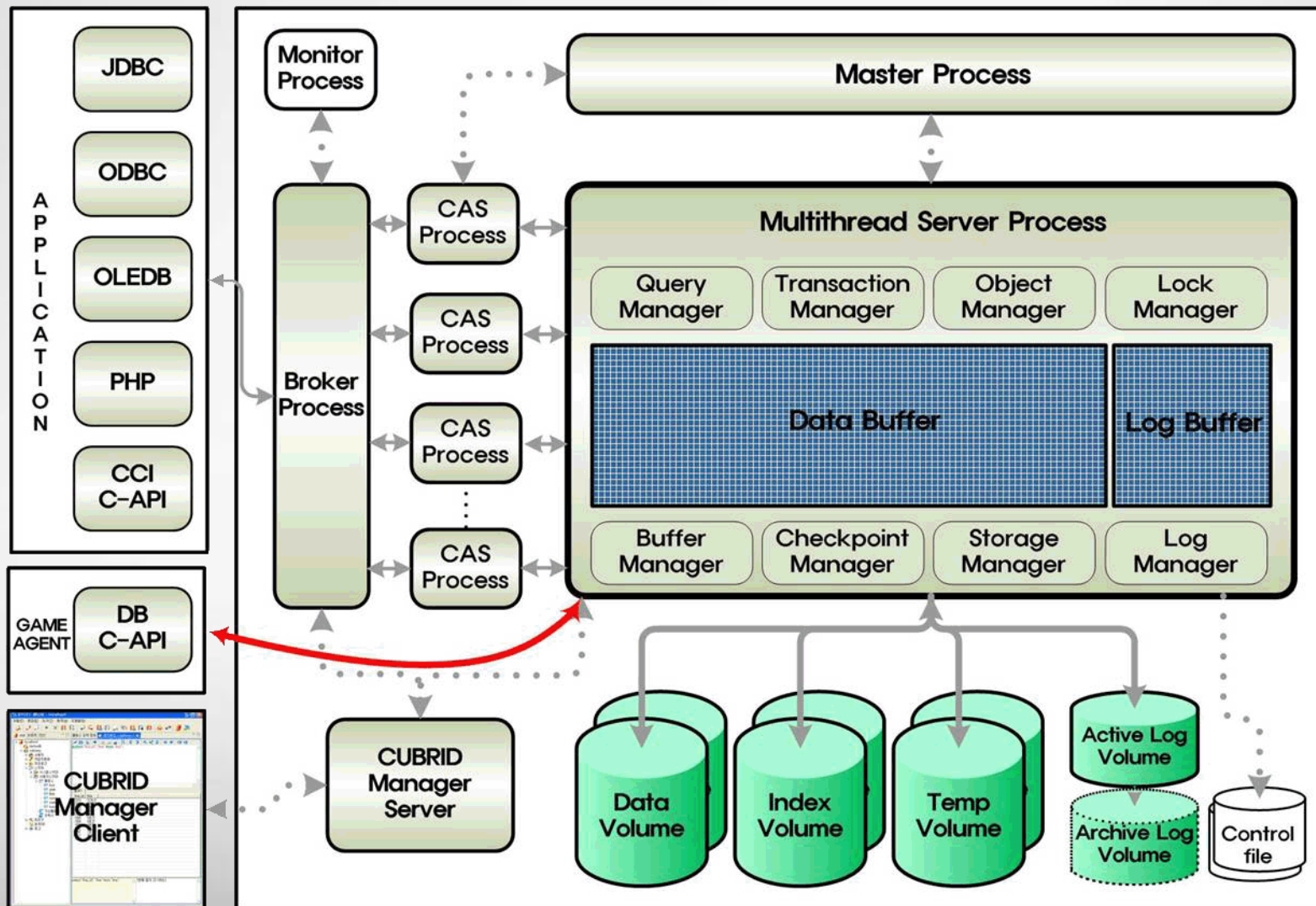




- Object 확장
  - 사용자 정의 데이터 타입(User Defined Data Type)
  - 집합형 데이터 타입(Collection Data Type)
  - 메소드(Method)
  - 상속(Inheritance)
- OID (Object Identifier)
  - Database 내 class 또는 record 에 대한 Identifier
    - 물리적 위치 정보(Volumn ID, Page ID, Slot ID)
      - 검색을 하지 않으므로 데이터에 대한 빠른 접근 보장
    - 재구성되지 않는 한 변경되지 않음
    - 사용자 정의 데이터 타입 사용시 해당 컬럼에 저장되어 지는 값



관계형 데이터베이스	CUBRID
table	class
view	virtual class
row	instance
column	attribute
type	domain
inline view	derived table



## 2. CUBRID 설치하기



CUBRID



OS	OS version	CPU
Microsoft Windows	Windows NT / 2000 / XP / 2003 / vista 32	x86
LINUX	glibc-2.3.2 기반 <ul style="list-style-type: none"> <li>- Asianux 1.0</li> <li>- Redhat 9</li> <li>- Redhat Enterprise ES/AS 3.0</li> <li>- Fedora core 1</li> <li>- CentOS 3.x</li> </ul>	x86 / Intel EM64T / AMD64
	glibc-2.3.4 기반 <ul style="list-style-type: none"> <li>- Asianux 2.0</li> <li>- Fedora core 4 이상</li> <li>- Redhat Enterprise ES/AS 4.0</li> <li>- CentOS 4.x, 5.x</li> <li>- SUSE 9.0 이상</li> <li>- Ubuntu 6.10 이상</li> </ul>	x86 / Intel EM64T / AMD64
SUN	solaris 7 ~ 10 x86 solaris 10	sparc x86
HP	HP-UX 11i HP-UX IA64	PA-RISC intel Itanium 64
IBM	AIX 5.3	PowerPC

# 설치전 확인/준비 사항

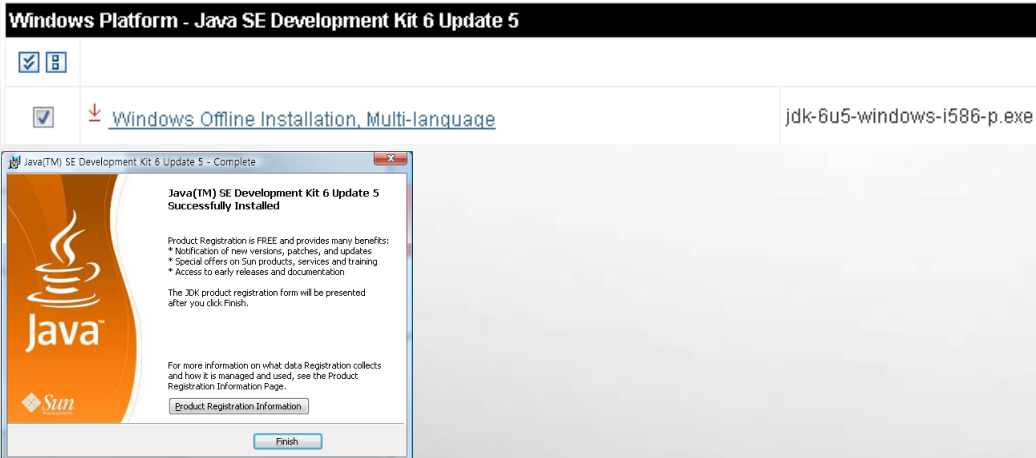


**CUBRID**  
Driving the DBMS REVOLUTION

- 필요한 시스템 자원 확인한다.
  - Memory : 최소 128M 이상 여유
  - Hard disk : 최소 150M 이상 여유
- 설치 대상 서버의 지원 여부를 CUBRID 홈페이지에서 확인한다.
- Linux / Unix 서버의 경우 설치 전 다음의 시스템 튜닝이 필요하다.
  - Linux : kernel 2.4.2 이상, glibc 2.3.2 / 2.3.4 이상
    - `uname -r`
    - `rpm -qa | grep glibc`
  - HP11i : memory 및 thread 관련 kernel parameter값 변경 (운영자과정 참고)
  - IBM-AIX5.3 : SCOPE mode (운영자과정 참고)
- Linux / Unix의 경우 설치 전 CUBRID DB를 관리할 사용자(User)를 생성한다.
- 자바가 설치되어 있는지를 확인한다.(JAVA 1.4이상, JAVA 1.6 권장)
  - JAVA Stored Procedure 를 사용하는 경우
    - LINUX 의 경우 glib234 이상에서만 지원
  - CUBRID manager client 를 사용하는 경우
  - 운영만 하는 경우 JRE 를 설치



- JAVA Stored Procedure 사용
- CUBRID Manager client 사용
- JAVA 다운로드
  - <http://java.sun.com/javase/downloads/index.jsp>
  - 최신버전의 JAVA 다운로드
    - JDK6 Update5 또는 JRE6 Update10(운영환경의 경우), 2008년11월현재
  - 사용 플랫폼(OS) 에 맞는 제품 다운로드(self-extracting file 선택)
    - Windows 의 경우 Offline Installation 선택





- JAVA 환경 변수 설정

- Windows (jvm.dll 이 있는 디렉토리를 PATH로 지정)

```
set JAVA_HOME= C:\ Program Files\ Java\ jdk1.6.0_10  
set PATH=%PATH%;%JAVA_HOME%\ jre\ bin\ client
```

- Linux

- bash shell

```
set JAVA_HOME=/usr/local/jdk1.6.10; export JAVA_HOME  
set LANG=korean; export LANG  
set PATH=$PATH:$JAVA_HOME/bin; export PATH  
set  
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/i386:$JAVA_HOME/jre/lib/i386/cl  
ient; export LD_LIBRARY_PATH
```

- c shell

```
setenv JAVA_HOME=/usr/local/jdk1.6.10  
setenv LANG=korean  
set path=($path $JAVA_HOME/bin)  
setenv LD_LIBRARY_PATH  
$LD_LIBRARY_PATH:$JAVA_HOME/jre/lib/i386:$JAVA_HOME/jre/lib/i386/client
```

- UNIX

- 위 LINUX 환경변수중 LD\_LIBRARY\_PATH 를 libjvm.so 가 있는 곳으로 수정
- 예) SUN : \$JAVA\_HOME/jre/lib/**sparc**:\$JAVA\_HOME/jre/lib/**sparc**/client





- JAVA 환경 변수 설정

- class path 설정

- JAVA stored procedure 수행시 사용되는 CUBRID JDBC driver 의 path 설정
    - Windows

```
set CLASSPATH=%CUBRID%\jdbc\ cubrid_jdbc.jar
```

- Linux/Unix
      - bash shell

```
set CLASSPATH=$CUBRID/jdbc/cubrid_jdbc.jar; export CLASSPATH
```

- c shell

```
setenv CLASSPATH $CUBRID/jdbc/cubrid_jdbc.jar
```

- JAVA compiler path 설정

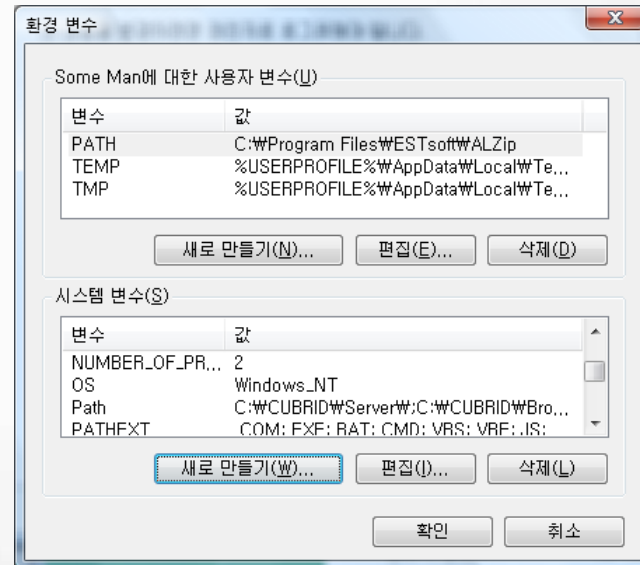
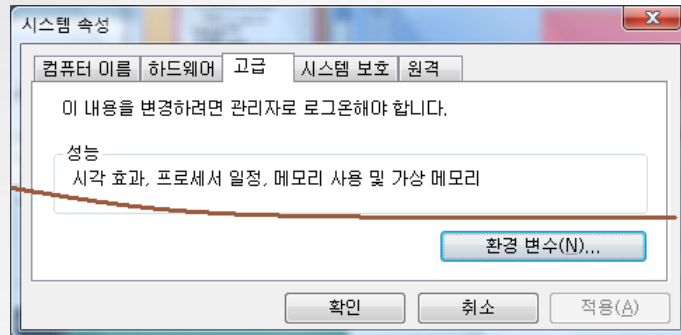
- javac (javac.exe) 에 대한 path 추가
    - windows
      - 기존 path 에 %JAVA\_HOME%\bin 추가
    - Linux/Unix
      - 기존 path 에 \$JAVA\_HOME/bin 추가

# 실습 (JAVA 설치)



CUBRID  
Driving the DBMS REVOLUTION

- JAVA 설치 (실습을 위해 JDK 를 설치)
  - JDK 다운로드 후 설치
    - 환경변수 설정
      - JAVA stored procedure 작성 위해 javac path 추가



- java, javac 동작 여부 확인
- Java 버전 확인
  - java -version
- 환경 변수상의 JAVA 설정 확인
  - JAVA\_HOME, PATH, LD\_LIBRARY\_PATH, CLASSPATH

# CUBRID 설치(Windows)



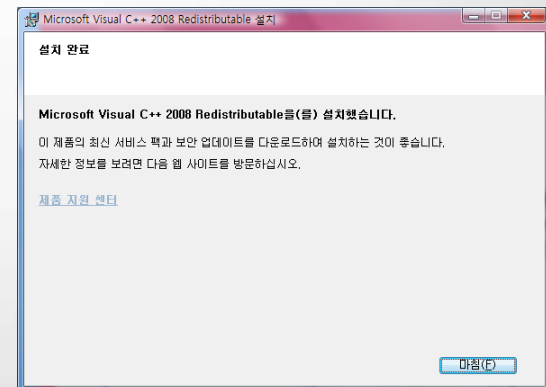
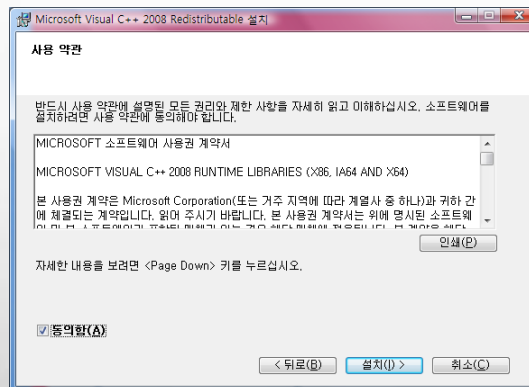
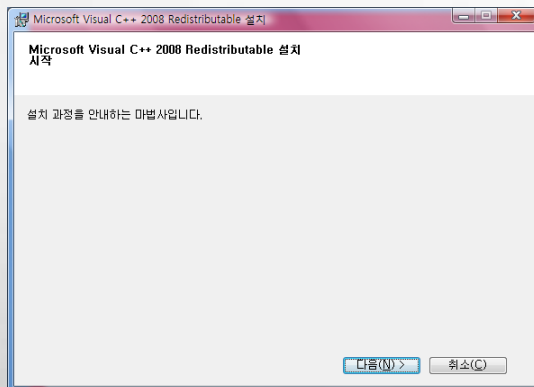
CUBRID  
Driving the DBMS REVOLUTION

- CUBRID 홈페이지에서 windows 제품 다운로드
- 다운로드 받은 화일 실행
  - 이전 버전의 제품이 있는 경우 설치되지 않으므로 제거후 설치
- DMBS 설치시 server component와 client component 동시 설치
  - ODBC 나 CUBRID manager client 만을 설치시 client component 설치
- 제품설치전 Microsoft Visual C++ 2008 재배포 패키지 를 설치
  - CUBRID 홈페이지에서 다운로드

Visual C++ 2008 재배포 패키지 다운로드

CUBRID 2008 Windows 용 제품을 설치하기 전에 반드시 설치해 주시기 바랍니다.

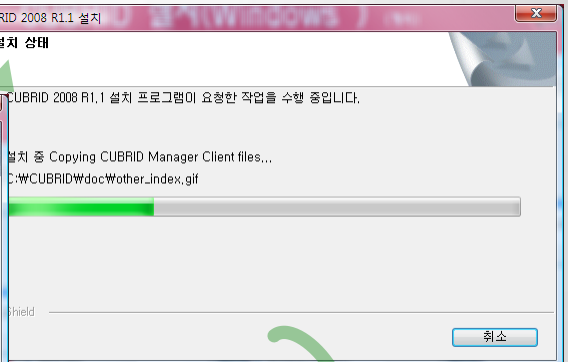
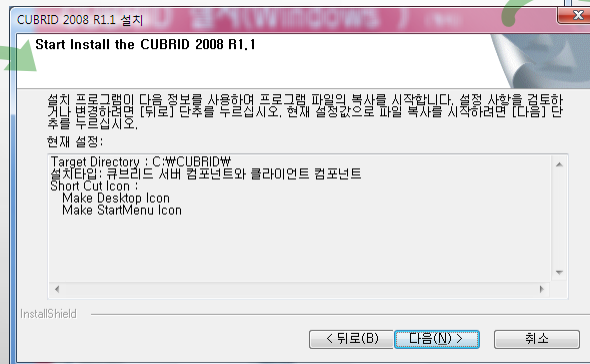
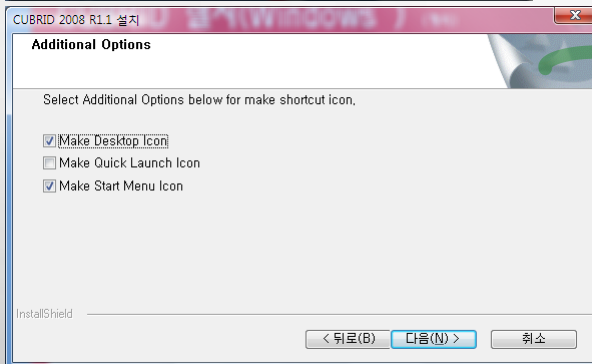
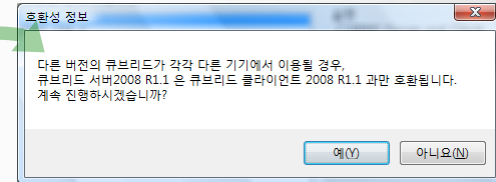
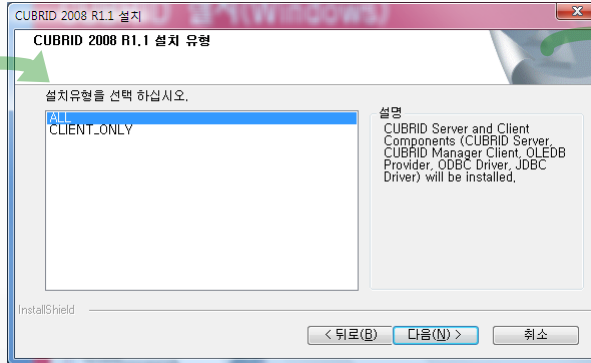
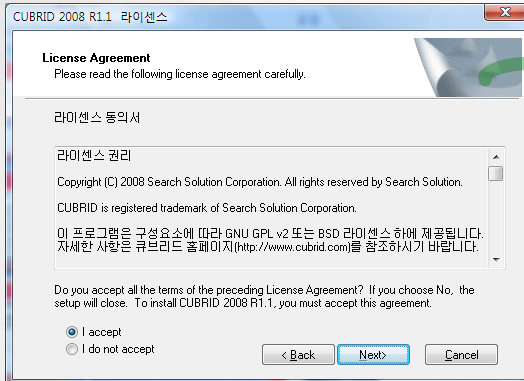
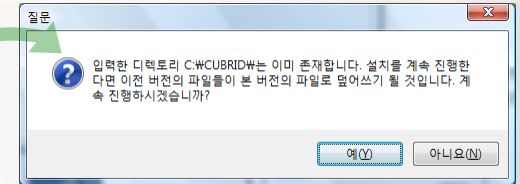
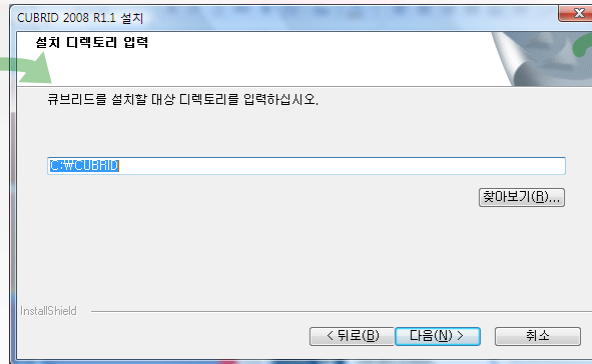
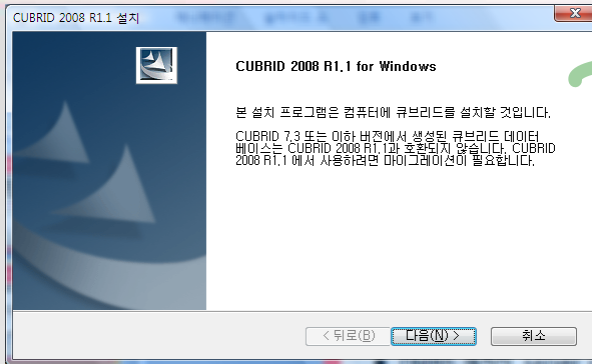
- <http://www.microsoft.com/downloads/details.aspx?displaylang=ko&FamilyID=9b2da534-3e03-4391-8a4d-074b9f2bc1bf>



# CUBRID 설치(Windows) (계속)



CUBRID  
Driving the DBMS REVOLUTION

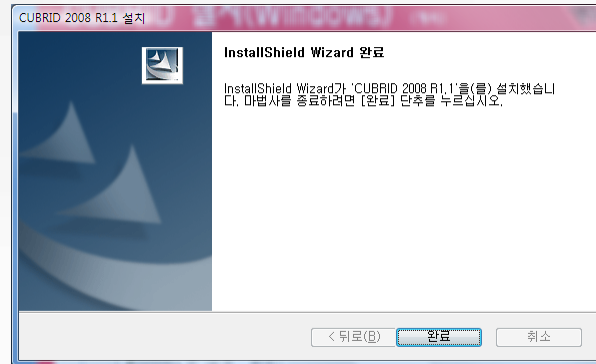
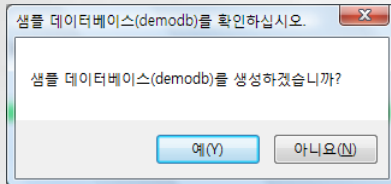


# CUBRID 설치(Windows) (계속)



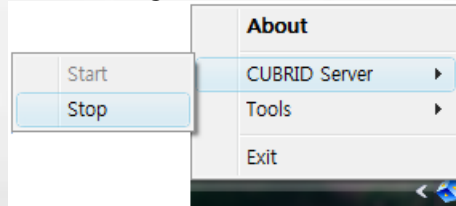
CUBRID  
Driving the DBMS REVOLUTION

- 실습을 위해 샘플 데이터베이스(demodb) 생성



- 설치가 완료되면 큐브리드 서비스가 자동 구동됨

- tray 상에 CUBRID icon 이 보이고, 파란색(구동중 의미)으로 표시됨.



- 서비스 등록 확인 (제어판→관리도구→서비스)



# CUBRID 설치 (Linux/UNIX)



**CUBRID**  
Driving the DBMS REVOLUTION

- JAVA Stored Procedure/CUBRID manager client 사용 환경 설정
  - JAVA Runtime 환경 필요 (JRE1.4 이상, JRE1.6 권장)
- CUBRID 홈페이지에서 Linux/UNIX 제품 다운로드
  - OS 종류, glibc 버전(Linux의 경우), CPU 종류 등 확인
  - CUBRID manager client 를 windows에서 사용할 경우 앞의 windows 용 설치 참고하여 client component 만 설치
  - 다운로드한 제품 upload(binary mode)
- CUBRID 관리자로 로그인후 제품 설치
  - 실습을 위해 demodb 데이터베이스 생성

# CUBRID 설치 (Linux/UNIX) (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

```
$ sh CUBRID_8.1.1.1032_linux.sh
```

라이선스 권리

Copyright (C) 2008 Search Solution Corporation. All rights reserved by Search Solution.  
CUBRID is registered trademark of Search Solution Corporation.

이 프로그램은 구성요소에 따라 GNU GPL v2 또는 BSD 라이선스 하에 제공됩니다. 자세한 사항은 큐브리드 홈페이지(<http://www.cubrid.com>)를 참조하시기 바랍니다.

...

The precise terms and conditions for copying, distribution and modification follow.

Do you agree to the above license terms? [yes or no] : yes

Input the CUBRID install directory. [Default: /home/web/CUBRID] :

Install CUBRID to '/home/web/CUBRID' ...

In case a different version of the CUBRID product is being used in other machines, please note that the CUBRID 2008 R1.1 servers are only compatible with the CUBRID 2008 R1.1 clients and vice versa.

Do you want to continue? (y/n) [Default: y] : yes

Copying old .cubrid.sh to .cubrid.sh.bak ...

CUBRID has been successfully installed.

demodb has been successfully created.

If you want to use CUBRID, run the following commands

```
% . /home/cubrid/.cubrid.sh
```

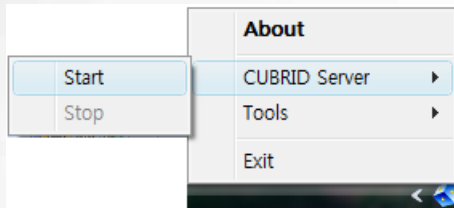
```
% cubrid service start
```



- CUBRID 서비스 구동

- Windows

- 자동으로 구동됨
    - 구동되지 않은 경우 tray CUBRID 서비스 메뉴를 이용하여 구동



- 명령어 이용

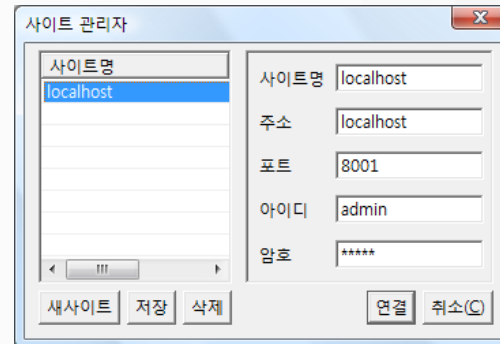
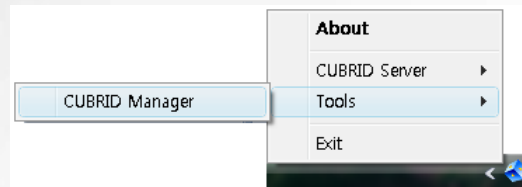
- cubrid service 명령어를 이용하여 구동

```
-bash-3.1$ 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
-bash-3.1$
```

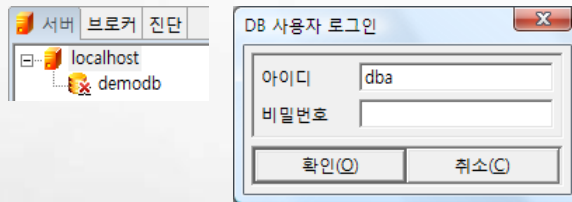




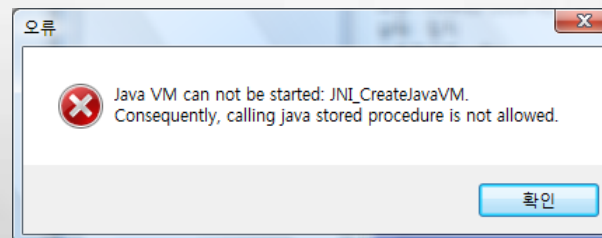
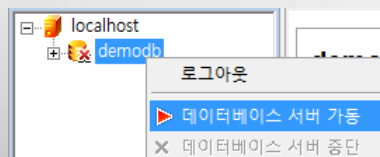
- 데이터베이스 구동
  - CUBRID manager client 에서 구동
    - CUBRID manager client 구동 및 로그인



- 데이터베이스 선택 및 데이터베이스 사용자 로그인



- 데이터베이스 구동





## – 명령어 이용

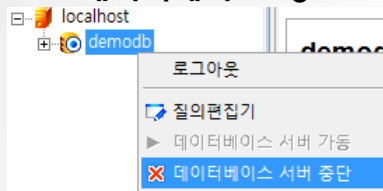
```
$ cubrid server start <데이터베이스 이름>
```

```
-bash-3.1$ cubrid server start demodb  
@ cubrid server start: demodb  
  
CUBRID 2008 R1.0  
  
++ cubrid server start: success  
-bash-3.1$
```



- 데이터베이스 종료

- CUBRID manager client 에서 종료
  - CUBRID manager client 구동 및 로그인
  - 데이터베이스 선택 및 데이터베이스 사용자 로그인
  - 데이터베이스 종료



- 명령어 이용

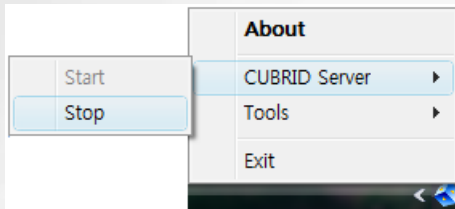
```
-bash-3.1$ cubrid server stop demodb
@ cubrid server stop: demodb

Server demodb notified of shutdown.
This may take several minutes. Please wait.
++ cubrid server stop: success
-bash-3.1$
```



- CUBRID 서비스 종료

- Windows



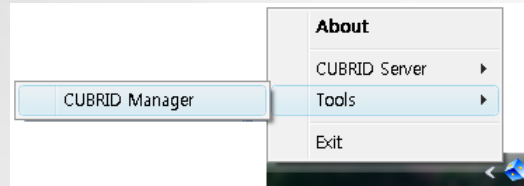
- 명령어 이용

```
-bash-3.1$ 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
-bash-3.1$
```

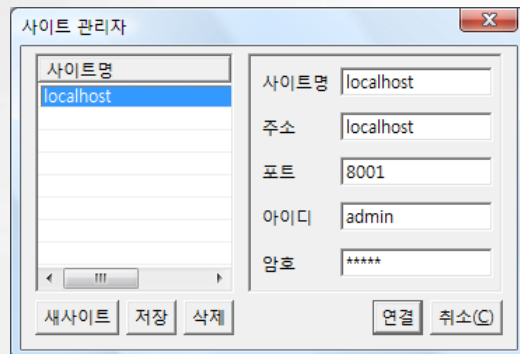
# CUBRID manager client



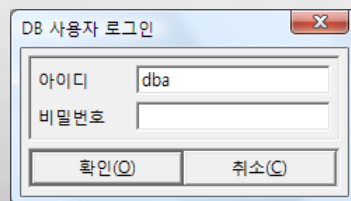
- 구동



- 로그인




- 데이터베이스 로그인

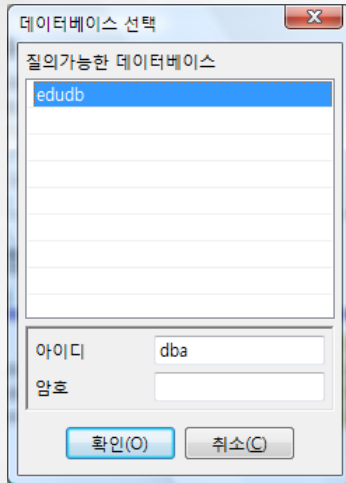


# CUBRID manager client (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

- 질의 편집기 사용
  - 데이터베이스 서버가 구동되어있어야 함
  - 상단의  를 클릭하여 질의편집기 로그인



# CUBRID manager client (계속)

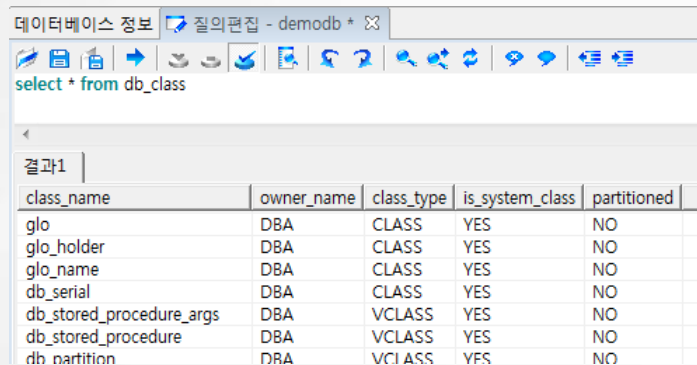


CUBRID  
Driving the DBMS REVOLUTION

## ● 질의 편집기 사용

### — 단축기

- 질의수행 : F5 / ctrl-Enter (실행아이콘: ➡)
- 편집취소 : ctrl-Z
- 편집복구 : ctrl-Y
- 질의수행계획보기 : ctrl-L



The screenshot shows the CUBRID Manager Client interface. The title bar indicates the active window is '질의편집 - demodb \*'. The menu bar includes options like '데이터베이스 정보', '질의편집', and '실행'. The toolbar contains icons for file operations, query execution, and editing. The query editor shows the SQL statement 'select \* from db\_class'. Below the editor, the results are displayed in a table format.

결과1

class_name	owner_name	class_type	is_system_class	partitioned
glo	DBA	CLASS	YES	NO
glo_holder	DBA	CLASS	YES	NO
glo_name	DBA	CLASS	YES	NO
db_serial	DBA	CLASS	YES	NO
db_stored_procedure_args	DBA	VCLASS	YES	NO
db_stored_procedure	DBA	VCLASS	YES	NO
db_partition	DBA	VCLASS	YES	NO



## ● CUBRID SQL interpreter

### – 실행

- csql [-u <데이터베이스 사용자>] [-S] <데이터베이스 이름>
  - 데이터베이스 사용자를 입력하지 않으면 PUBLIC 사용자로 로그인
  - -S : 서버가 구동되지 있지 않은 경우 사용

### – 입력한 SQL 문은 버퍼에 저장됨. 버퍼의 내용을 실행시키거나, 삭제후 새로 입력후 실행

### – 명령어

- ;RUn : 버퍼의 내용을 수행
- ;Xrun : 버퍼의 내용을 수행후 버퍼의 내용 지움
- ;edit : 버퍼의 내용을 편집 (windows 의 경우 notepad 사용)
- ;list : 버퍼의 내용 보기
- ;CLean : 버퍼의 내용 지우기
- ;au on/off : 자동 커밋 ON/OFF (default ON)
- ;help : 명령어 도움말 보기
- ;EXit : 실행종료





```
관리자: 명령 프롬프트
C:\Users\Administrator>csql demodb

      CUBRID SQL Interpreter

Type `;help' for help messages.

csql> select class_name from db_class where rownum = 10
csql> ;x

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

  class_name
=====
  'db_meth_file'

1 rows selected.

Current transaction has been committed.

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

C:\Users\Administrator>
```



- CUBRID 설치 (windows 용)
  - 제품 다운로드 후 설치
    - demodb 생성
    - CUBRID service tray 구동 확인
    - CUBRID service 구동 확인
      - 서비스, 프로세스
  - CUBRID manager client
    - 데이터베이스 생성 확인
    - 데이터베이스 서버 구동
      - 구동시 JAVA 관련 에러메세지 표시 여부 확인
    - 질의 편집기 사용
      - 간단한 질의 수행 : `select * from db_class`
    - 데이터베이스 서버 종료
    - 큐브리드 서비스 종료
      - 프로세스 확인
    - 큐브리드 서비스 구동

### 3. 데이터베이스 설계



CUBRID



## 고객 명함 관리

### 고객회사 코드

고객회사 코드 CHAR(5)  
고객회사 이름 VARCHAR(200)

고객회사 코드 CHAR(5)  
고객 이름 VARCHAR(20)  
직위 VARCHAR(10)  
이메일 VARCHAR(100)  
전화번호 VARCHAR(20)  
주소 VARCHAR(200)

### 고객 취미 관리

고객회사 코드 CHAR(5)  
고객 이름 VARCHAR(20)  
고객 취미 VARCHAR(100)

```
create table company (
    company_id      char(5),           // 고객회사 코드
    company_name    varchar(200),      // 고객회사 이름
    primary key (company_id)
);

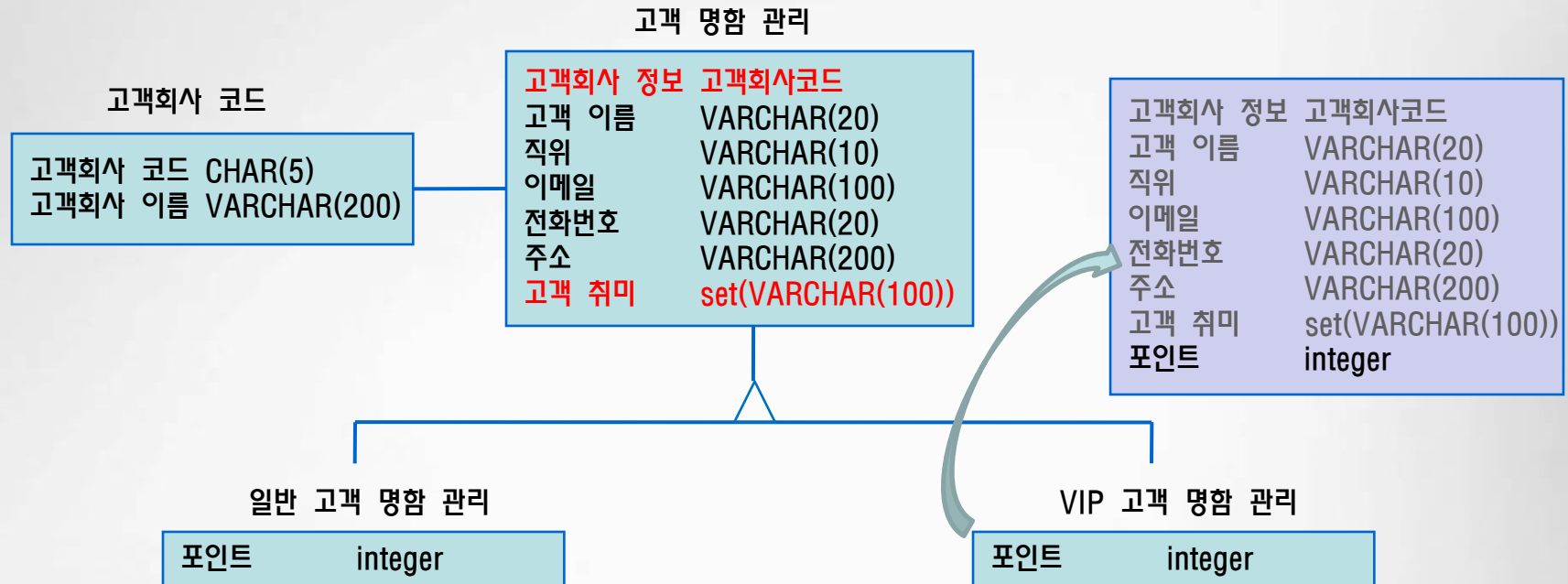
create table client (
    company_id      char(5),           // 고객회사 코드
    client_name     varchar(20),       // 고객 이름
    title           varchar(10),       // 직위
    email           varchar(100),      // 이메일
    phone           varchar(20),       // 전화번호
    address         varchar(200),      // 주소
    foreign key (company_id) references coompany(company_id)
);

create table client_hobby (
    company_id      char(5),           // 고객회사 코드
    client_name     varchar(20),       // 고객 이름
    hobby           varchar(100)       // 취미
);
```

# 객체 관계형 모델



CUBRID  
Driving the DBMS REVOLUTION



- 각 테이블간의 데이터를 조회하기 위해서는 조인이 필요하나, 객체 관계형 모델에서는 조인없이 참조하는 데이터를 조회할 수 있음
- 보다 유연한 데이터 모델링 제공
- 상속은 스키마 레벨의 상속이며, 데이터는 각 테이블에 별도 저장되며 상속되지 않음.
  - 인덱스도 상속되지 않음
  - parent table 의 unique 는 상속관계 전체에서 unique 보장

# 객체 관계형 모델



CUBRID  
Driving the DBMS REVOLUTION

```
create table company (  
    company_id    char(5),           // 고객회사 코드  
    company_name  varchar(200)      // 고객회사 이름  
);  
  
create table client (  
    company_oid   company,           // 고객회사 정보  
    client_name   varchar(20),       // 고객 이름  
    title         varchar(10),       // 직위  
    email         varchar(100),      // 이메일  
    phone        varchar(20),       // 전화번호  
    address       varchar(200),      // 주소  
    client_hobby  set(varchar(100)) // 고객 취미  
);  
  
create table normal_client under client (  
    point         integer           // 포인트  
);  
  
create table vip_client under client (  
    point         integer           // 포인트  
);
```



- 기본 데이터 타입
  - CHAR(n)
  - VARCHAR(n)
  - BIT(n)
  - BIT VARYING(n)
  - NUMERIC(n,m) or DECIMAL
  - INTEGER
  - FLOAT or REAL
  - DOUBLE
  - DATE
  - TIME
  - TIMESTAMP
  - SET(domainIdomain list)
  - MULTISSET(domainIdomain list)
  - SEQUENCE or LIST(domainIdomain list)



## ● 문자열 타입

자료형	형식예제	데이터	비고
CHAR	'pacesetter' as CHAR(12) 'pacesetter ' as CHAR(10) 'pacesetter' as CHAR(4) 'p ' as CHAR	' pacesetter ' 'pacesetter' Error condition 'p'	
VARCHAR	'pacesetter' as VARCHAR(12) 'pacesetter ' as VARCHAR(12) 'pacesetter ' as VARCHAR(10) 'pacesetter' as VARCHAR(4) 'p ' as VARCHAR	' pacesetter' 'pacesetter ' 'pacesetter' Error condition 'p'	





## ● 숫자 타입

자료형	형식예제	데이터	비고
NUMERIC or DECIMAL	55555.3333 as NUMERIC 555.33 as NUMERIC(5) 55555.3333 as NUMERIC(9,4) 5.33 as NUMERIC(3,4) .533333 as NUMERIC(4,4)	55555. 555. 55555.3333 Error condition .5333	NUMERIC(p,[s])
INTEGER	8934 as INTEGER 7823467 as INTEGER 89.8 as INTEGER 3458901122 as INTEGER	8934 7823467 89 Error condition	INTEGER
SMALLINT	8934 as SMALLINT 89.8 as SMALLINT 23467 as SMALLINT 89354 as SMALLINT	8934 89 23467 Error condition	SMALLINT
FLOAT	8934 as FLOAT 34 as FLOAT(0) -234.67 as FLOAT(5)	8.934000e+03 3.400000e+01 -2.346700e+02	FLOAT(p)
DOUBLE PRECISION	89.998765431132 as DOUBLE -18.256743237573 as DOUBLE	8.999876543113200e+01 -1.825674323757300e+01	DOUBLE PRECISION



## ● 날짜형 타입

자료형	형식예제	데이터	비고
DATE	DATE '11/11/1994' DATE '11/11' DATE '2008-10-23' DATE '10-23'	DATE '11/11/1994' DATE '11/11/1994' DATE '2008-10-23' DATE '10-23'	DATE 'mm/dd[/yyyy]' DATE '[yyyy-]mm-dd'
TIME	TIME '1:15:45 pm' TIME '16:08:33 am' TIME '16:08:33 pm' TIME '1:15'	'01:15:45 PM' Invalid time error '04:08:33 PM' '01:15:00 AM'	TIME 'hh:mm [:ss] [am   pm]'
TIMESTAMP	TIMESTAMP '01/31/1994 8:15:00 pm' TIMESTAMP '16:08:33 am 1/1/1946'	TIMESTAMP '8:15:00 PM 01/31/1994' Error	TIMESTAMP 'hh:mm [:ss] [am pm] mm/dd [/yyyy]' TIMESTAMP 'mm/dd [/yyyy] hh:mm [:ss] [am pm]'



## ● 집합형 타입

자료형	형식예제	데이터	비고
SET	SET (VARCHAR(20),INTEGER) SET (cabin,suite) SET CHAR(5) SET () SET VARCHAR(10)	{'golf', 'handicap', 10} {cabin, suite} {'aaa ', 'bbbb ', 'cccc' } {11, 22, 33.8, 'hello'} {'golf', 'scuba'}	Input :{'golf', 'scuba', 'golf'} Stored :{'golf', 'scuba'}
LIST or SEQUENCE	SEQUENCE INTEGER SEQUENCE (cabin, suite) LIST CHAR(5)	{20, 40, 60, 80} {cabin, cabin, suite} {'aaa ', 'bbbb ', 'cccc' }	
MULTISET	MULTISET (FLOAT, INTEGER) MULTISET (cabin, suite) MULTISET CHAR(5)	{10, 20, 10, 80, 50.5} {cabin, cabin, suite} {'aaa ', 'bbbb ', 'cccc' }	



- 객체형 타입 (사용자정의 데이터타입)
  - 데이터의 타입으로 이미 선언된 테이블을 타입으로 선언

고객 명함 관리

고객회사 코드

고객회사 코드 CHAR(5)  
고객회사 이름 VARCHAR(200)

고객회사 정보	고객회사코드
고객 이름	VARCHAR(20)
직위	VARCHAR(10)
이메일	VARCHAR(100)
전화번호	VARCHAR(20)
주소	VARCHAR(200)
고객 취미	set(VARCHAR(100))

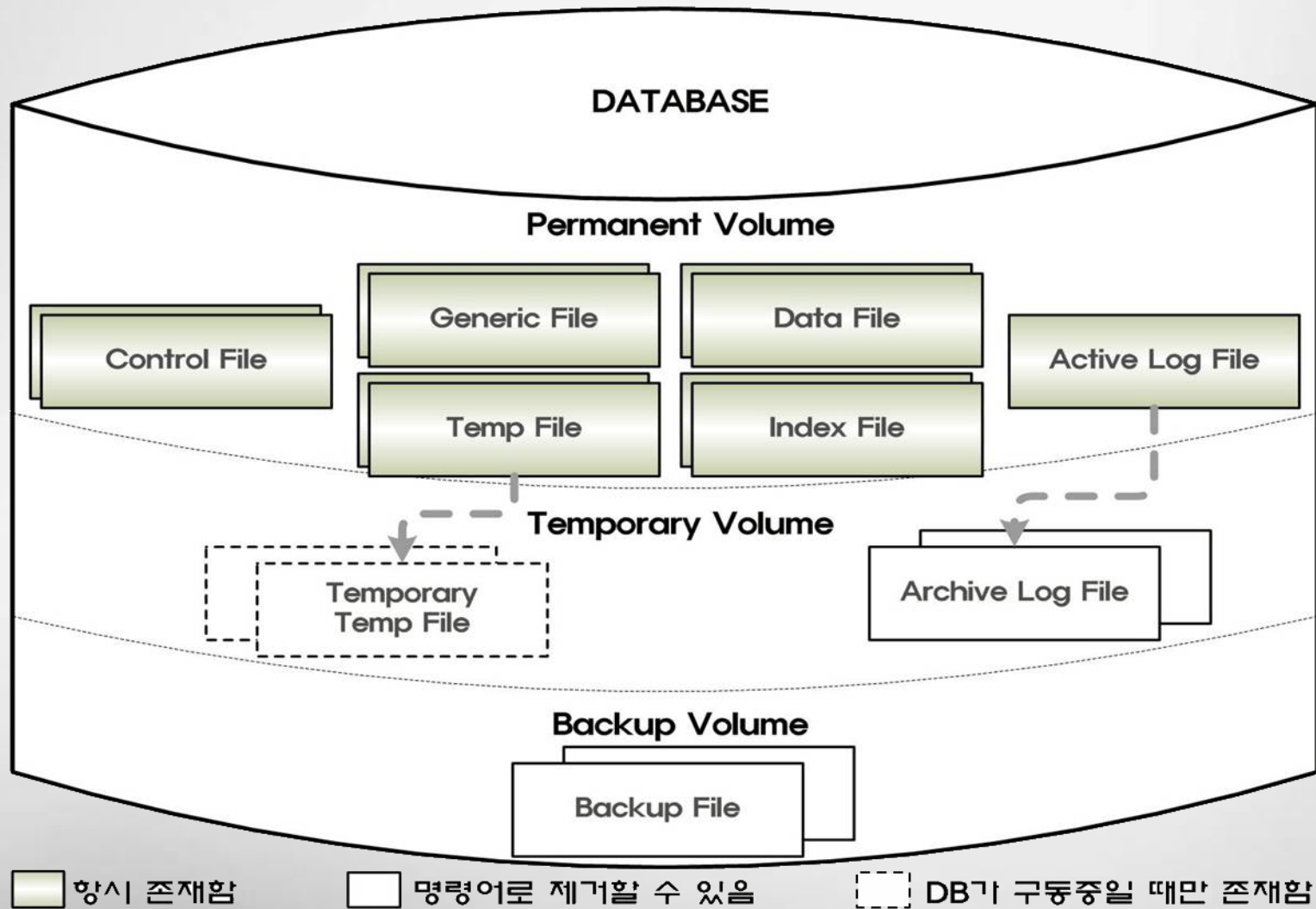
```
create table company (
    company_id    char(5),           // 고객회사 코드
    company_name  varchar(200)      // 고객회사 이름
);

create table client (
    company_oid   company,           // 고객회사 정보
    client_name   varchar(20),       // 고객 이름
    title         varchar(10),       // 직위
    email         varchar(100),      // 이메일
    phone         varchar(20),       // 전화번호
    address       varchar(200),      // 주소
    client_hobby  set(varchar(100)) // 고객 취미
);
```

## 4. 데이터베이스 생성

CUBRID







## ● Permanent Volumes

- 데이터베이스 초기화(생성)시에 생성되는 화일들
- 공간 부족 등의 이유로 추가되어지는 화일들
  - 삭제되어지지도, 삭제할 수도 없음
- 다음 3가지로 분류됨
  - control volumes
  - information volumes
  - log volumes
- information volumes 은 실제 데이터등이 저장되는 공간이며, 추가 가능
- 각 화일들의 크기는 페이지로 표현
  - 데이터베이스 입출력시 기본 입출력 단위
  - 모든 화일들은 동일한 크기의 페이지로 구성됨
  - 데이터베이스 생성시 지정하며, 변경 불가능
  - 1K ~ 16K 까지 지정가능하며, 기본값은 4K



## ● Control Volumes

- text 파일이며, 데이터베이스를 구성하는 화일들에 대한 정보를 가짐
- volume information file
  - 데이터베이스를 구성하는 각 화일들에 대한 정보를 가짐
  - 화일명 : <db name>\_vinf
- log information file
  - 데이터베이스 로그 파일에 대한 정보를 가짐
  - 화일명 : <db name>\_lginf
- backup information file
  - 데이터베이스 백업 파일에 대한 정보를 가짐
  - 화일명 : <db name>\_bkvinf





## ● Information Volumes

- 데이터, 인덱스 등이 저장되는 파일
- 생성시 용도를 구분
  - data volume
    - 실제 데이터 및 스키마 정보 저장
  - index volume
    - 검색을 위한 인덱스 정보 저장
  - temp volume
    - 검색(order by, group by, join, subquery 등) 시 사용
    - permanent temp 라 칭하며, 사용자가 만들어 놓은 공간이 부족하면 temporary temp volume 을 자동 생성하여 사용
  - generic volume
    - data, index, temp 세가지 용도로 모두 사용
    - 데이터베이스 생성시 초기 볼륨



## ● Log Volumes

- 데이터베이스에 대한 갱신 정보를 기록
- committed/aborted/active 트랜잭션 상태 기록
- 데이터베이스당 한 개의 log active volume 을 가짐
- 데이터베이스 손상시 데이터베이스 복구를 위해 사용
- 할당된 모든 공간이 사용되면, 로그에 대한 백업(log archive)을 만들어 보관
  - 로그 백업 생성시 성능 저하
  - 운영중 모니터링을 통해 업무시간중 로그 백업이 발생하지 않도록 로그 크기 조정 필요
  - 데이터베이스 백업을 통해 로그 정리 및 필요없어진 로그 백업 삭제 가능(백업 옵션)
- 화일명
  - <db name>\_lgat (<db name>l), <db name>\_lgarxxx(<db name>a.xxx)



## ● Temporary Volumes

- permanent volumes 중 temp volume의 공간이 부족한 경우, 서버가 자동 생성
- 생성된 볼륨은 서버의 재구동시 삭제됨
- 자동 생성되는 경우 성능 저하
  - 볼륨 생성시 I/O 증가
  - temp volume 이 필요한 일부 SQL 은 수행 중지(볼륨 생성완료시 수행 재개)
  - 운영중 모니터링을 통해 증가된 크기 만큼의 permanent temp volume 추가 필요



## ● Backup Volumes

- 데이터베이스 백업을 통하여 생성
- 온라인 백업시 “fussy” snapshot 백업
  - 현재 상태 그대로 백업
  - 완료되지 않은 트랜잭션 포함
  - 완료되지 않은 트랜잭션에 대한 정보는 log archive 에 기록
  - 온라인 백업본으로 복구시 log archive 필요 (없을 경우 부분 복구)
- 이전 백업 존재시 overwrite 여부 확인
- 데이터베이스 삭제시 백업 파일 보존(옵션으로 삭제 가능)
- 데이터베이스 시점 복구시 로그 파일과 같이 사용



- **databases.txt**

- 데이터베이스가 존재하는 서버, 위치 정보를 가짐
  - 데이터베이스이름
  - 볼륨정보화일의 위치
  - 서버이름
    - 서버이름 변경시 반드시 수정되어야 함.
  - 로그화일의 위치
  - 예) demodb /database/demodb db\_server /database/demodb/log
- 데이터베이스 생성 및 볼륨 추가시 정보 기록
  - 데이터베이스 관리자는 이 파일에 대한 쓰기/접근 권한이 있어야 함
- CUBRID\_DATABASES 환경변수가 지정하는 위치에 존재
  - 환경변수가 없는 경우 현재 디렉토리 참조

- **cubrid.conf**

- CUBRID 시스템 설정 정보 저장 화일
- 메모리사용, 파일위치, 파일크기, 락레벨 등 데이터베이스 중요 설정 정보
- \$CUBRID/conf 아래 존재
  - 데이터베이스별로 별도 지정 가능



- 초기 범용 볼륨 및 로그 볼륨 생성
  - 초기 범용 볼륨
    - 스키마 정보
    - 카다로그 정보
    - 적정 크기 : 5,000 pages
  - 로그 볼륨
    - 적정 크기 : 100,000 pages
- Information(데이터, 인덱스, 템프) 볼륨 추가
  - 산정된 용량에 따라 용도별 볼륨 추가
  - 볼륨의 최대 크기 : 2G (4K page size 일 경우 500,000pages)



## ● 데이터베이스 생성 예

### – 구성

- 위치 : /database/edudb (모든 볼륨 동일)
  - 단 windows 의 경우 C:\CUBRID\Databases\edudb
- 초기 범용 볼륨 : 5,000 pages
- 로그 볼륨 : 100,000 pages
- 데이터 볼륨 : 500,000 pages
- 인덱스 볼륨 : 250,000 pages
- 템프 볼륨 : 250,000 pages

### – 명령어 이용(Linux/UNIX)

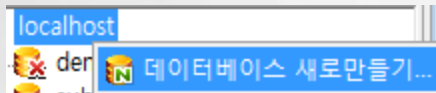
```
% cubrid createdb -p 5000 -F /database/edudb -l 100000 -L /database/edudb edudb
% cubrid addvoldb -S -p data -F /database/edudb edudb 500000
% cubrid addvoldb -S -p index -F /database/edudb edudb 250000
% cubrid addvoldb -S -p temp -F /database/edudb edudb 250000
```

# 데이터베이스 생성 (계속)



CUBRID  
Driving the DBMS REVOLUTION

## – CUBRID manager client 이용



새 데이터베이스 생성

기본 설정(페이지 1)

데이터베이스 이름  
생성할 데이터베이스 이름: edudb

범용 볼륨 정보  
페이지 수: 5000  
페이지 크기(Bytes): 4096  
범용 볼륨 경로: C:\CUBRID\databases\wedudb

로그 볼륨 정보  
페이지 수: 100000  
로그 볼륨 경로: C:\CUBRID\databases\wedudb\log

< 뒤로(B) 다음(N) > 확인(F) 취소(C)

새 데이터베이스 생성

추가 볼륨 설정(페이지 2)

새로 추가될 볼륨 정보  
볼륨 이름: edudb\_temp\_x00  
볼륨 경로: C:\CUBRID\databases\wedudb  
볼륨 타입: temp 페이지 수: 250000  
볼륨 추가 볼륨 삭제

추가될 볼륨 리스트

볼륨 이름	타입	페이지 수	경로
edudb_data_x000	data	500000	C:\CUBRID\databases\wedudb
edudb_index_x000	index	250000	C:\CUBRID\databases\wedudb
edudb_temp_x000	temp	250000	C:\CUBRID\databases\wedudb

< 뒤로(B) 다음(N) > 확인(F) 취소(C)



# 데이터베이스 생성 (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

**새 데이터베이스 생성**

**데이터베이스 생성 정보(페이지 3)**

새 데이터베이스를 생성하기 위해 필요한 과정을 모두 마쳤습니다.  
생성될 데이터베이스의 설정 내용은 아래와 같습니다.

데이터베이스 이름: edudb

페이지의 크기: 4096 bytes

범용 볼륨의 페이지 수: 5000 pages

로그 볼륨의 페이지 수: 100000 pages

아래 추가 볼륨들이 생성됩니다.

볼륨 이름: edudb\_data\_x000  
볼륨 타입: data volume  
볼륨의 페이지 크기: 500000 pages  
볼륨의 디렉토리 경로: C:\CUBRID\databases\edudb

< 뒤로(B)   다음(N) >   **확인(E)**   취소(C)

**새 디렉토리 생성**

아래 디렉토리가 존재하지 않습니다  
생성할까요?

디렉토리

C:\CUBRID\databases\edudb  
C:\CUBRID\databases\edudb\log

**확인(O)**   **취소(C)**

데이터베이스를 생성합니다

잠시만 기다려주세요...

서버 브로커 진단

localhost  
demodb  
edudb

**데이터베이스 정보**

**edudb**

버전 : CUBRID 2008 R1.1 (8.1.1.1032)  
상태 : 가동중  
사용자 권한 : dba  
페이지 크기 : 4096 byte  
총 용량 : 1005000 pages(3.93G)  
남은 용량 : 1004741 pages(3.93G)

**볼륨 정보**

볼륨 이름	사용 용량/남은 용량(페이지)	총 용량	타입
edudb	55/4945	5000	GENERIC
edudb_data_x000	141/499859	500000	DATA
edudb_index_x000	53/249947	250000	INDEX
edudb_temp_x000	10/249990	250000	TEMP



- 아래 조건에 맞는 데이터베이스를 생성
  - 각 볼륨별 생성 위치 및 크기
    - 페이지크기 : 4Kb
    - 첫번째 볼륨 : 5,000p, C:\CUBRID\databases\<데이터베이스이름>
    - 로그 볼륨 : 100,000p, C:\CUBRID\databases\<데이터베이스이름>\log
    - 데이터 볼륨 : 500,000p, C:\CUBRID\databases\<데이터베이스이름>
    - 인덱스 볼륨 : 250,000p, C:\CUBRID\databases\<데이터베이스이름>
    - 템프 볼륨 : 250,000p, C:\CUBRID\databases\<데이터베이스이름>
- 생성된 볼륨 확인
  - databases.txt 내용 확인
  - volume information file 의 내용을 보고 각 디렉토리의 파일 확인
    - control volumes
    - information volumes
    - log volumes

## 5. 스키마 관리



CUBRID



- class (table)
  - 개수 무제한
  - 이름에 한글, 영문자, 숫자, \_, #, % 사용 가능, 첫글자는 문자(한글, 영문)여야 함.
  - 이름의 최대 길이는 255자
- attribute (column)
  - class 당 최대 6,400 개 생성 가능
  - 이름에 한글, 영문자, 숫자, \_, #, % 사용 가능, 첫글자는 문자(한글, 영문)여야 함.
  - 이름의 최대 길이는 255자
- index
  - class 당 최대 6,400 개 생성 가능
  - 이름지정 가능
- 제약조건
  - NULL
    - NULL 값을 허용
  - NOT NULL
    - NULL 값을 허용하지 않음
  - unique
    - 중복된 값 허용하지 않음
    - default 와 같이 선언 불가
  - primary key
  - foreign key



## ● class 생성 구문

```
CREATE CLASS 클래스이름(하위 클래스)
[ UNDER | AS SUBCLASS OF 상위 클래스 이름 {, 상위 클래스 이름} ... ]
[ (애트리뷰트 정의절 [{, 애트리뷰트 정의절 }])];
```

애트리뷰트 정의절 :

    애트리뷰트\_이름 데이터\_타입 [DEFAULT 값] [제약 사항]

제약 사항 :

    PRIMARY KEY  
    FOREIGN KEY  
    NOT NULL  
    UNIQUE  
    NULL

```
create class my_company (
    comp_id int not null unique,
    comp_name varchar(100) default ''
)
```



## ● CUBRID manager client 이용

**클래스 추가**

클래스 타입: ☒ 클래스 ☐ 가상클래스  
 소유자: dba  
 클래스 이름: my\_client  
 확인 버튼을 누르면 클래스 편집 화면으로 이동하여 속성, 인덱스 등을 작성할 수 있습니다.  
 확인(O) 취소(C)

**속성 추가**

이름: client\_id  
 도메인:   
 타입: INTEGER  
☐ 클래스속성  
☐ SHARED  
☐ NOT NULL  
☐ UNIQUE  
 DEFAULT:   
 주의: 도메인이 문자열일 경우 작은 따옴표로 둘러싸야 합니다.  
 ex) 'abcdef', 'abc'  
 확인(O) 취소(C)

**속성 추가**

이름: comp\_id  
 도메인:   
 타입: INTEGER  
☐ 클래스속성  
☐ SHARED  
☐ NOT NULL  
☐ UNIQUE  
 DEFAULT:   
 주의: 도메인이 문자열일 경우 작은 따옴표로 둘러싸야 합니다.  
 ex) 'abcdef', 'abc'  
 확인(O) 취소(C)

**클래스 상세 조회 및 편집**

일반 수퍼클래스 메소드

이름: my\_client  
 구분: 사용자스키마  
 소유자: DBA  
 타입: 클래스  
 변경

속성:

이름	도메인	NOT NULL	SHARED	UNIQUE	DEFAULT	수퍼클래스	클래스속성

인덱스

이름	인덱스 종류	속성	규칙

추가 수정 삭제

닫기(C)

속성:

이름	도메인	NOT NULL	SHARED	UNIQUE	DEFAULT
client_id	integer				
comp_id	integer				

# primary key



- class의 각 레코드를 대표(식별)하는 키, class 당 1개만 지정
- 추가 및 삭제는 alter 명령으로 가능
  - foreign key 로 참조되는 경우 삭제 불가
- 이름 지정가능하며, 미 지정시 pk\_<class이름>\_<필드명>

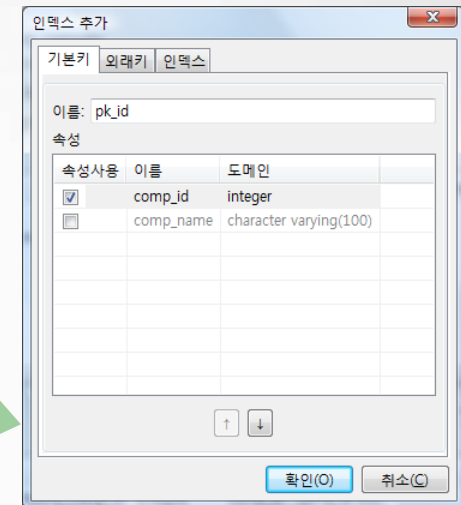
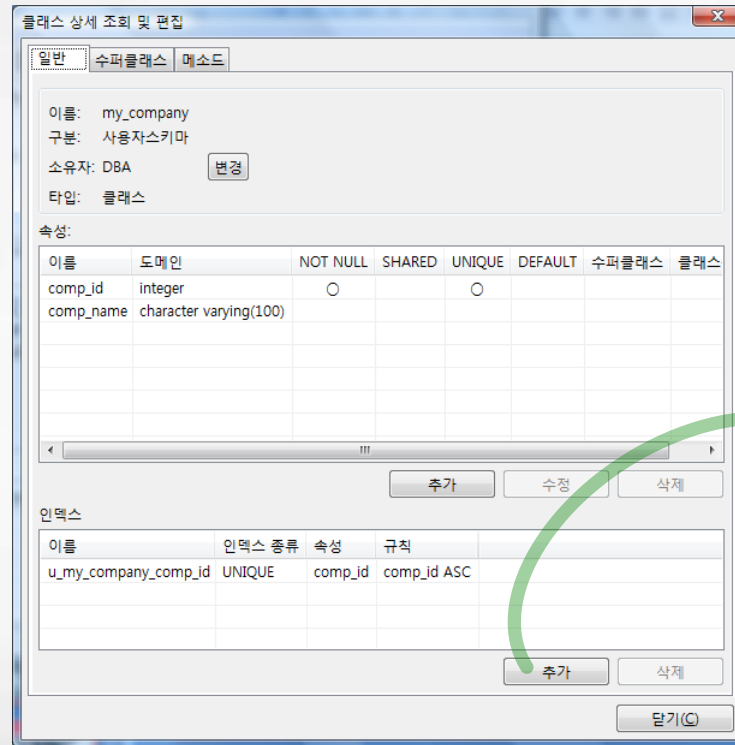
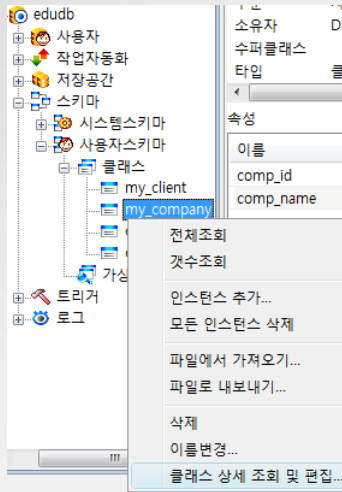
```
create class country (  
    id int,  
    primary key (id)  
)  
create class company (  
    comp_id int,  
    comp_name varchar(100) not null unique,  
    constraint pk_id primary key(comp_id)  
)  
alter class country drop constraint pk_country_id  
alter class country add primary key (id)  
  
alter class company drop constraint pk_id  
alter class company add constraint pk_id primary key (comp_id)
```

# primary key (계속)



CUBRID  
Driving the DBMS REVOLUTION

## ● CUBRID manager client 이용



인덱스

이름	인덱스 종류	속성	규칙
u my company comp id	UNIQUE	comp id	comp id ASC
pk_id	PRIMARY KEY	comp_id	



# foreign key



- class 간 참조무결성을 유지하기 위해 사용
- 추가 및 삭제는 alter 명령으로 가능
- 입력/수정시 기본키의 존재 여부 확인
  - 존재하지 않는 경우 에러
- 이름은 지정가능하며, 미 지정시 fk\_<class이름>\_<필드명>

```
create class company (  
    comp_id int,  
    comp_name varchar(100),  
    primary key (comp_id)  
 ) // 앞서 생성한 테이블  
create class client (  
    client_id int,  
    client_name varchar(20),  
    comp_id int,  
    foreign key (comp_id) references company(comp_id)  
 )  
  
alter class client drop constraint fk_client_comp_id  
alter class client add constraint fk_id foreign key (comp_id) references company(comp_id)  
  
alter class client drop constraint fk_id  
alter class client add foreign key (comp_id) references company(comp_id)
```



- **정의시 참조하는 개체에 대한 참조무결성 유지위한 옵션 제공**
  - **on delete** : 참조되어지는 기본키 개체 삭제시 연산 규칙
    - no action : 기본키는 삭제, 추가적인 연산은 없음
    - restrict : 참조하는 외래키가 있는 경우 삭제 금지 (default)
    - cascade : 기본키는 삭제, 참조하는 외래키가 있는 경우 삭제
  - **on update** : 참조되어지는 기본키 개체 수정시 연산 규칙
    - no action : 기본키는 수정, 추가적인 연산은 없음
    - restrict : 참조하는 외래키가 있는 경우 수정 금지 (default)
  - **on cache object** : 외래키 설정시 객체 참조 관계 정의
    - 기본키를 가지는 클래스를 도메인으로 하는 컬럼 추가
    - 레코드 입력시 해당 컬럼에 자동으로 OID 입력
    - 큐브리드에서만 제공
    - 외래키값이 입력되지 않는 경우 관련 필드에 값이 NULL 로 들어감
    - on cache object 필드명과 일반필드명이 같은 경우 에러 발생

# foreign key (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

```
create class company (  
    comp_id int,  
    comp_name varchar(100),  
    primary key (comp_id)  
 ) // 앞서 생성한 테이블  
create class client2 (  
    client_id int,  
    client_name varchar(100),  
    comp_id int,  
    foreign key (comp_id) references company(comp_id) on update no action on delete  
restrict  
 )  
  
alter class client2 drop constraint fk_client2_comp_id  
alter class client2 add foreign key (comp_id) references company(comp_id) on update restrict  
on delete cascade on cache object comp_oid  
  
select co.comp_name from client2 cl, company co where cl.comp_id = co.comp_id  
select comp_oid.comp_name from client2
```

# Virtual class(view)



CUBRID  
Driving the DBMS REVOLUTION

- 사용 빈도 높은 복잡한 질의에 대한 단순화
- 특정 사용자에게 대한 접근 제한
- 인덱스 추가 / 지정(using index) 불가
- 가상 class 생성 구문

```
CREATE VCLASS 클래스이름(하위 클래스)
[ UNDER | AS SUBCLASS OF 상위 클래스 이름 {, 상위 클래스 이름} ... ]
[ (애틀리뷰트 정의절 [{, 애틀리뷰트 정의절 }])];
```

애틀리뷰트 정의절 :

애틀리뷰트\_이름 데이터\_타입 [DEFAULT 값] [NOT NULL]

```
create vclass v_company_a (
    id      int,
    name varchar(100)
)
as select comp_id, comp_name from company where comp_name like 'a%'

create view v_company_b
```

# Virtual class(view) (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

## ● CUBRID manager client 이용

### — 필드 추가

The screenshot illustrates the steps to add a virtual class and its fields in the CUBRID Manager Client.

**가상클래스 추가...** (Add Virtual Class...): This dialog box is used to create a new virtual class. It shows the class type set to '가상클래스' (Virtual Class), the owner as 'dba', and the class name as 'v\_company\_c'.

**가상클래스 상세 조회 및 편집** (View and Edit Virtual Class Details): This window displays the details of the 'v\_company\_c' virtual class. It shows the class name, owner, and type. The '속성' (Properties) tab is active, showing a table with columns for Name, Domain, NOT NULL, SHARED, UNIQUE, DEFAULT, Superclass, and Class Property. The '인덱스' (Index) tab is also visible.

**속성 추가** (Add Property): This dialog box is used to add a new property to the virtual class. It shows the property name, domain, type, and size. The '속성' (Properties) tab is active, showing the property name, domain, type, and size.

**속성:** (Properties): This table lists the properties of the virtual class.

이름	도메인
id	integer
name	character varying(100)



# Virtual class(view) (계속)



CUBRID  
Driving the DBMS REVOLUTION

- 갱신 가능한 가상 class
  - 질의부의 형식에 따라서 virtual class의 갱신 여부가 결정된다.
  - 변경 가능한 virtual class를 생성시 아래조건을 만족해야 함
    - FROM 절에 하나의 클래스 또는 갱신 가능한 가상 클래스를 참조
    - DISTINCT나 UNIQUE 키워드 사용불가
    - GROUP BY...HAVING 절 사용불가
    - 집합함수는 SELECT 절에서 사용불가
    - UNION ALL 구문은 UNION 양쪽에 업데이트 가능한 entities가 사용되어야 함
    - path expression 사용불가
    - 복잡한 산술연산자가 포함된 숫자 형식 사용불가



- class 변경
  - class 이름 변경
  - 필드 추가
    - unique 속성을 가지는 경우 unique 확인 및 중복시 에러
    - not null 속성을 가지는 경우 기존값 null 처리 필요함
  - 필드명 변경
  - 필드 초기값 변경/제거
  - not null 변경
    - CUBRID manager client 에서만 가능
  - 필드 타입 변경
  - 필드 삭제
  - 가상 class 질의 스펙 변경
- class 제거



# class 변경 (계속)

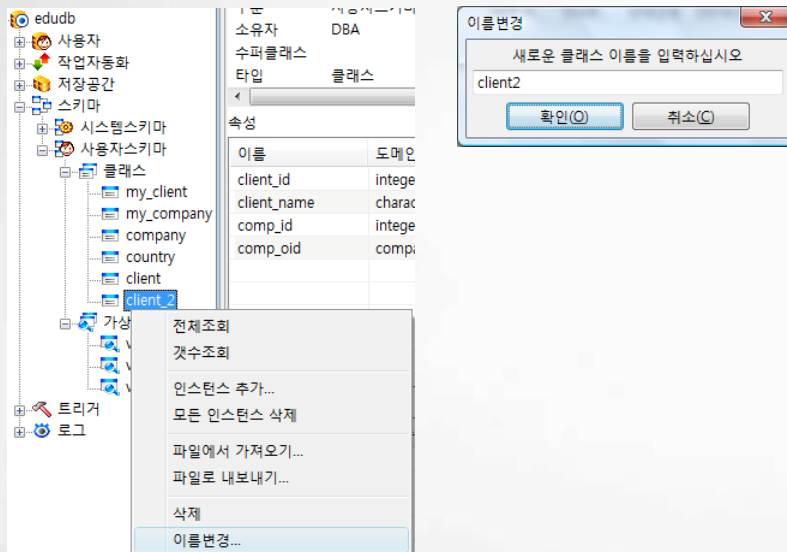


CUBRID  
Driving the DBMS REVOLUTION

## ● class 이름 변경

```
rename class client2 as client_2
```

### — CUBRID manager client 이용



# class 변경 (계속)

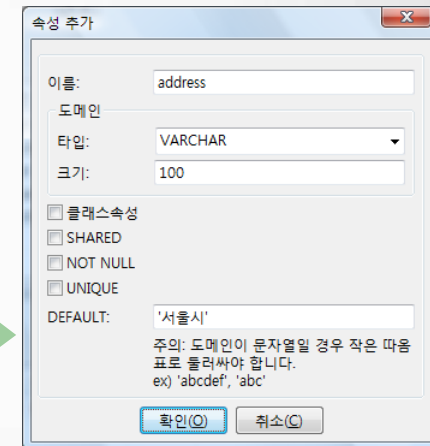
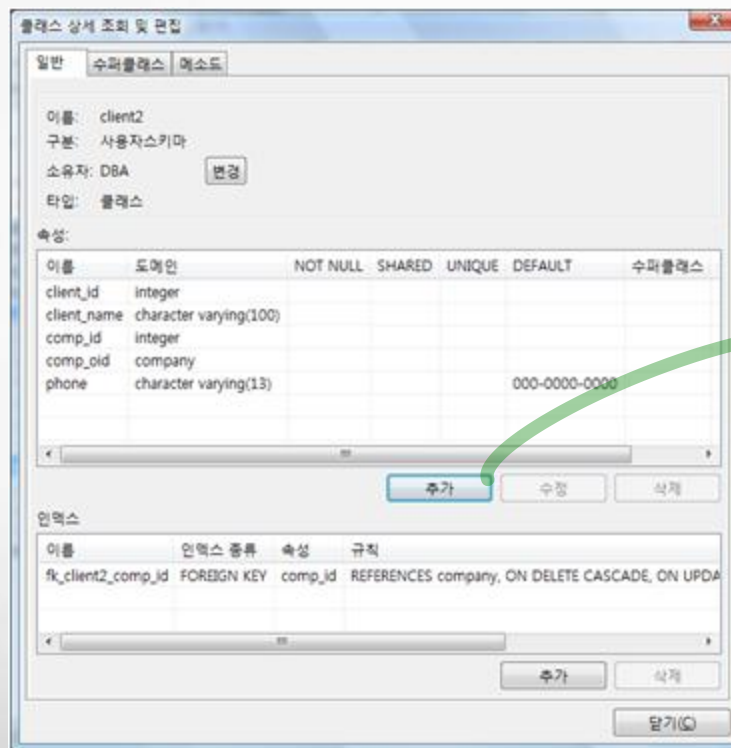
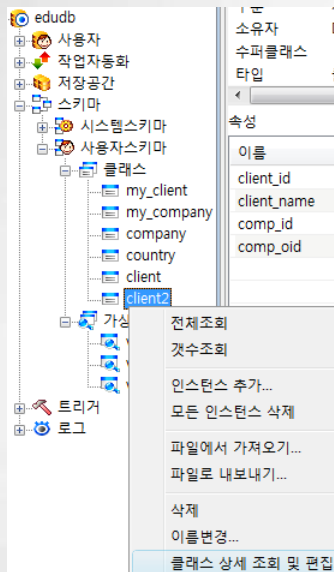


CUBRID  
Driving the DBMS REVOLUTION

## ● 필드 추가

```
alter class client2 add attribute phone varchar(13) default '000-0000-0000'
```

### — CUBRID manager client 이용



속성:

이름	도메인	NOT NULL	SHARED	UNIQUE	DEFAULT
client_id	integer				
client_name	character varying(100)	○			
comp_id	integer				
comp_old	company				
phone	character varying(13)				000-0000-0000
address	character varying(100)				서울시

# class 변경 (계속)

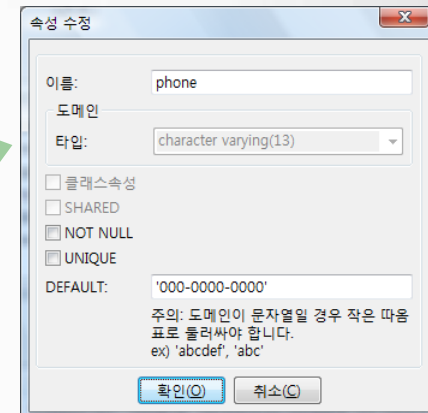
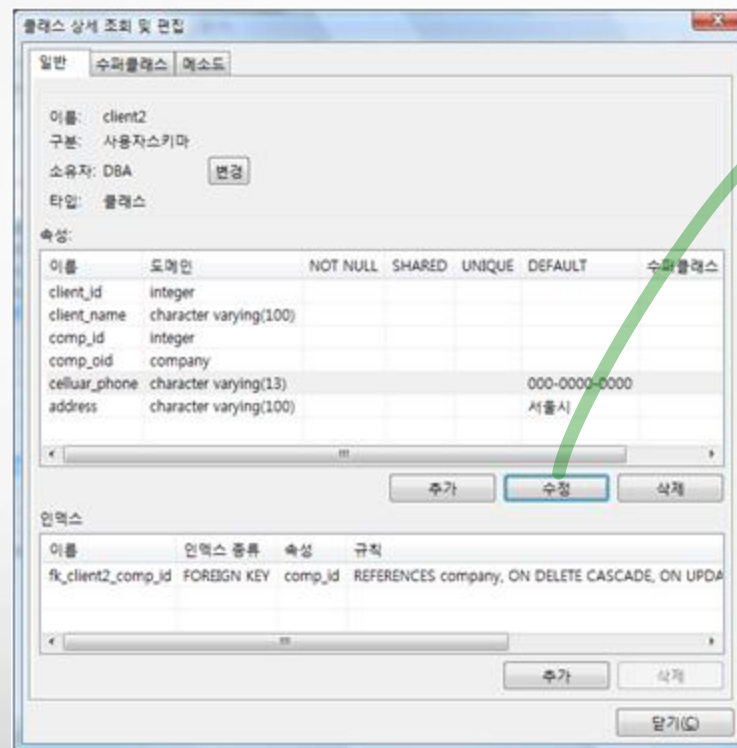
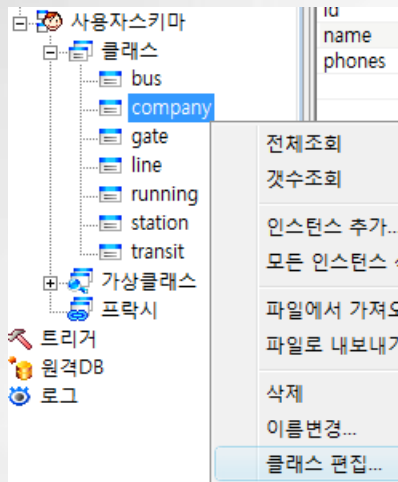


CUBRID  
Driving the DBMS REVOLUTION

## ● 필드명 변경

```
alter class client2 rename attribute phone as cellular_phone
```

### — CUBRID manager client 이용

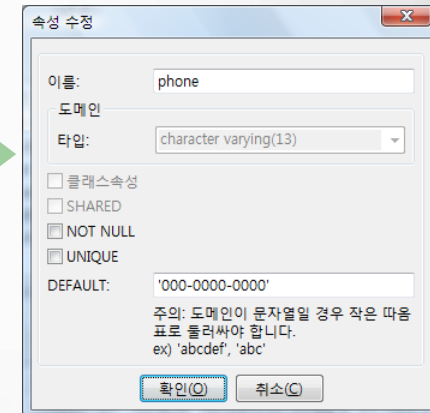
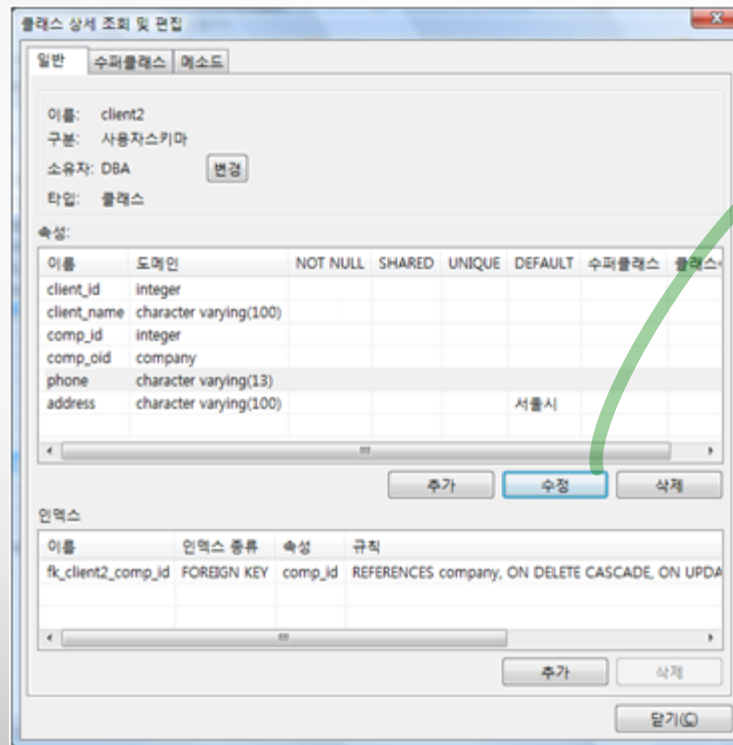
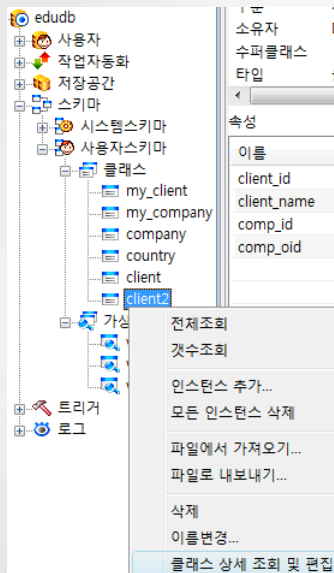




## ● 필드 초기값 변경

```
alter class client2 change phone default '0000-0000'  
alter class client2 change phone default NULL -- default 값 없음
```

- CUBRID manager client 이용
  - default 값 없애는 것 허용되지 않음

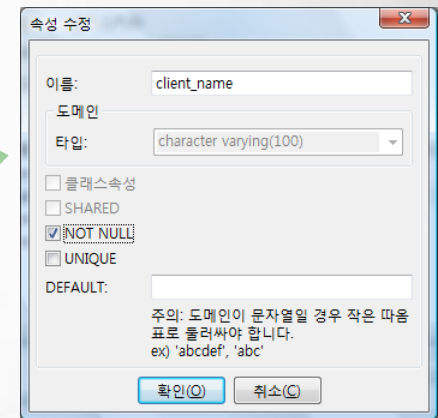
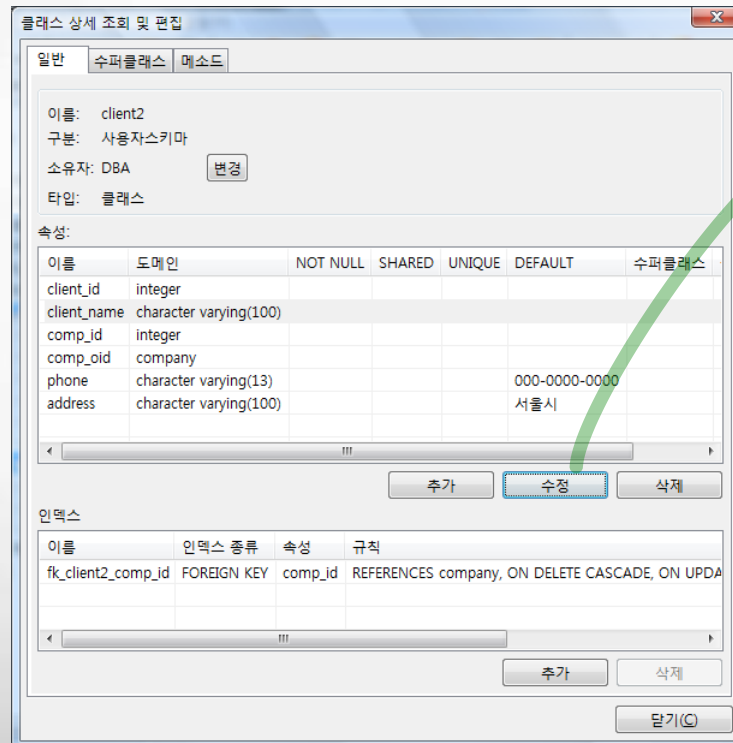
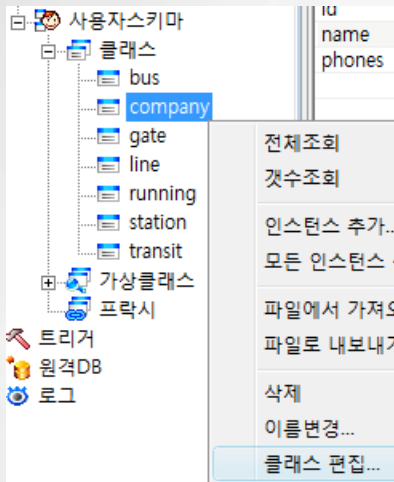


# class 변경 (계속)



CUBRID  
Driving the DBMS REVOLUTION

- not null 변경
  - CUBRID manager 에서만 가능
  - null → not null 로 변경하는 경우, 기존값에 null 이 있는지 확인하지 않음
    - null 처리후 변경하여야 함.



속성:

이름	도메인	NOT NULL
client_id	integer	
client_name	character varying(100)	○
comp_id	integer	
comp_oid	company	



- 필드 타입 변경

- 명령어를 통해 지원되지 않음
- 변경하고자 하는 타입의 필드를 추가하여 업데이트 하는 형식으로 변경
  - 업데이트시 적절한 형변환 필요
- 클래스내의 필드 순서가 달라지므로 주의

```
alter class client2 rename attribute phone as old_phone  
alter class client2 add attribute phone varchar(20)  
update client2 set phone = old_phone  
alter class client2 drop attribute old_phone
```

- 필드순서가 변하지 않도록 하는 방법

```
rename class client2 as old_client2  
create class client2 ( ... // 원하는 타입으로 client2 table 생성  
insert into client2(...) select ... from old_client2
```

# class 변경 (계속)

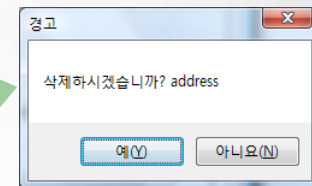
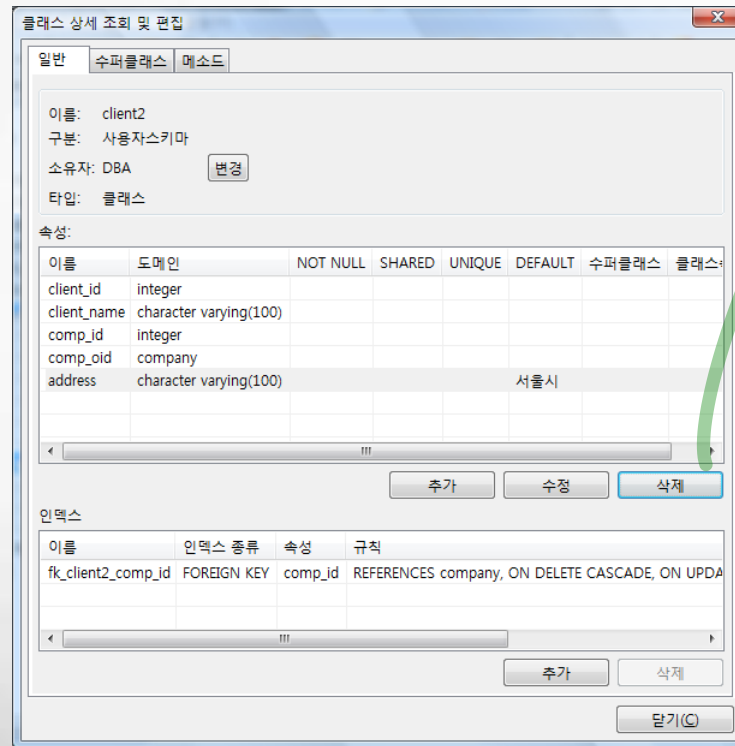
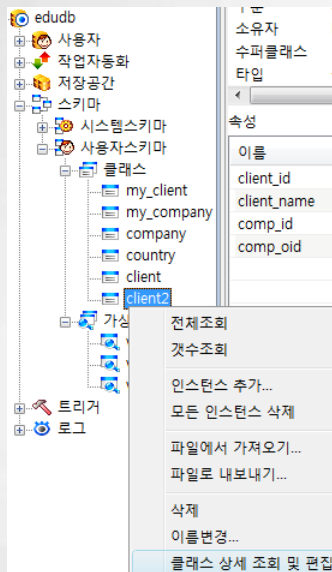


CUBRID  
Driving the DBMS REVOLUTION

## ● 필드 삭제

```
alter class client2 drop attribute phone
```

### — CUBRID manager client 이용

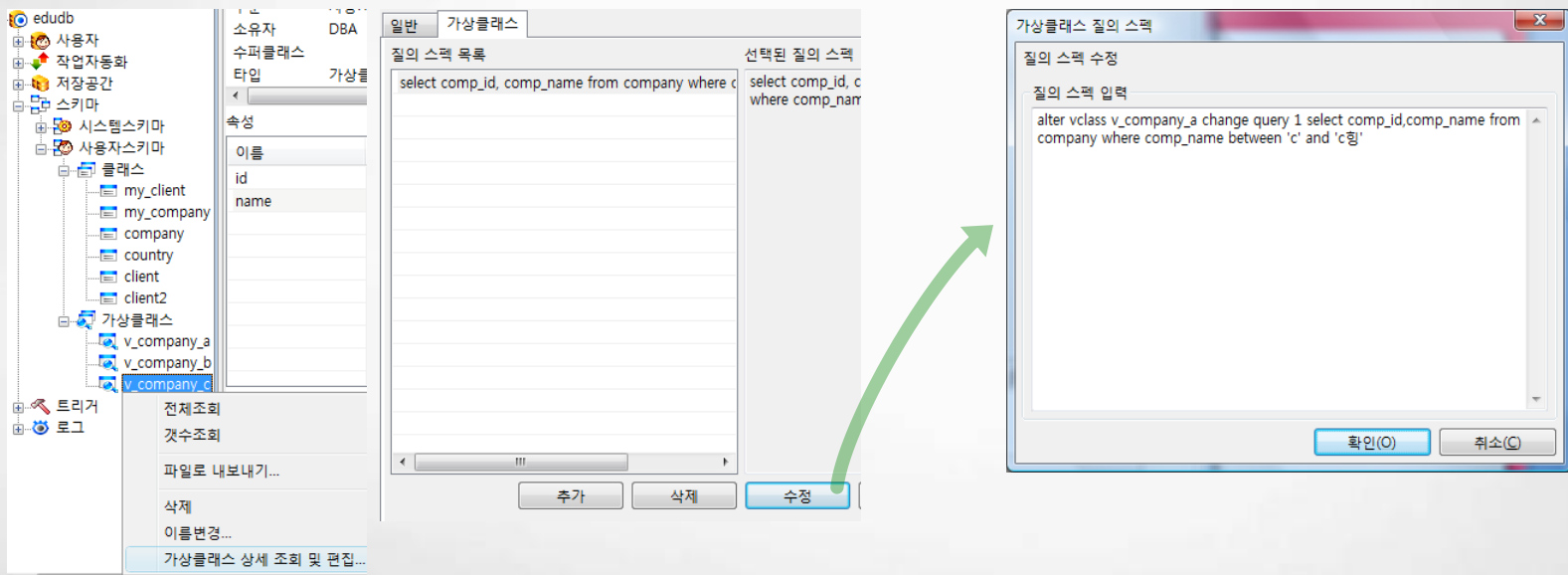




## ● 가상클래스의 질의 명세 변경

```
alter vclass v_company_a change query 1 select comp_id,comp_name from company where comp_name between 'a' and 'a형'
```

### – CUBRID manager client 이용







## ● class 제거 구문

DROP [CLASS | VCLASS] 클래스이름 [ { , 클래스이름 }... ]

DROP ONLY 클래스이름

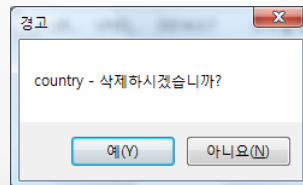
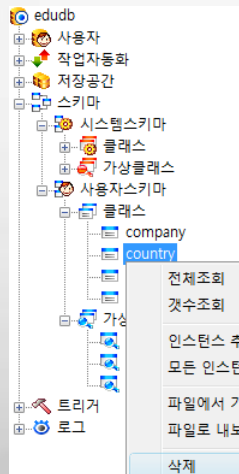
DROP ALL 클래스이름 [ EXCEPT 클래스이름 ]

drop my\_company

drop class my\_client

drop vclass v\_company\_a

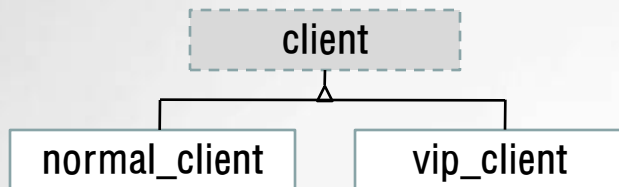
- foreign key로 참조되어 지는 경우 에러
- CUBRID manager client 이용



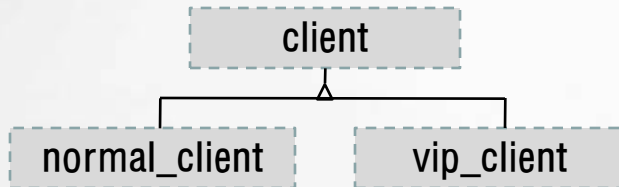


- 상속관계에서 class 제거

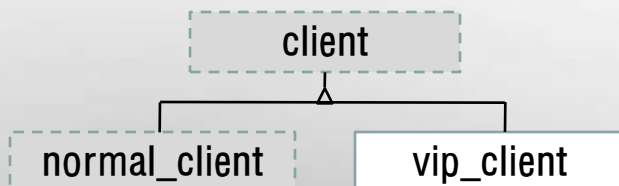
drop only client



drop all client



drop all client except vip\_client





- class의 레코드에 대한 신속한 검색을 위하여 검색 대상 필드의 값(들)을 모아 구성한 구조체
- 각 필드에 대하여 오름차순(default), 내림차순 지정 가능
- unique, primary key, foreign key 도 index 를 통한 관리
- reverse index : 모든 필드에 대하여 내림차순으로 정렬된 인덱스
  - 오름차순 지정은 무시됨
- index 이름 지정가능하며, 미지정시 i\_<class이름>\_<필드명>
  - unique index 의 경우 u\_<class이름>\_<필드명>

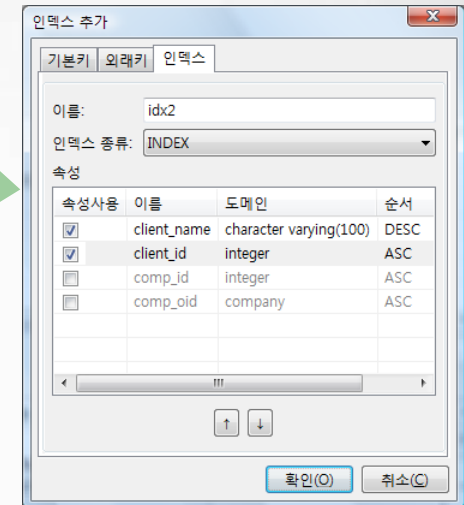
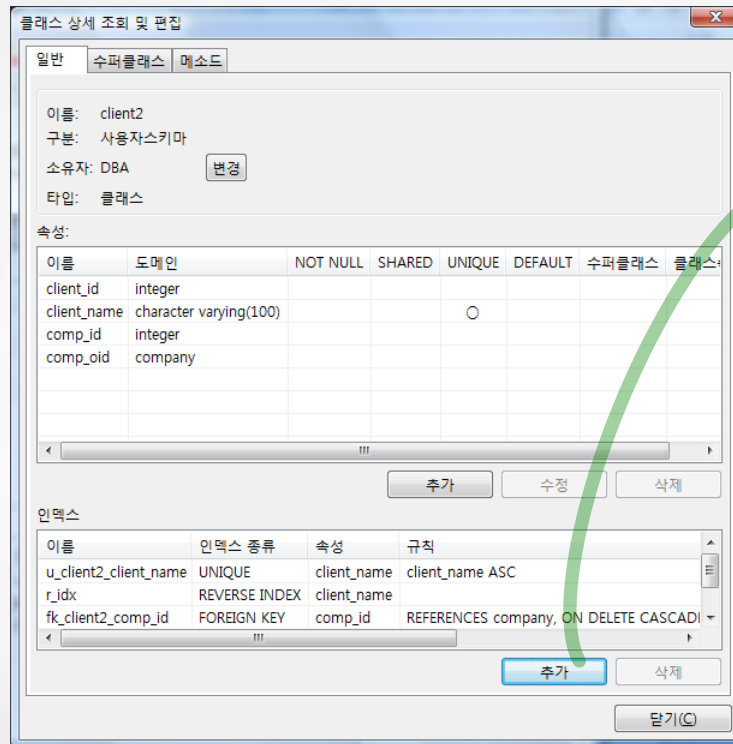
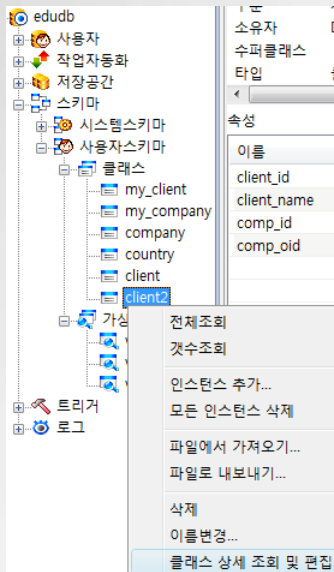
```
-- 일반 index
create index on client2(client_name)
create index idx1 on client2(client_id, client_name desc)

drop index on client2(client_name)
drop index idx1

-- unique index
create unique index on client2(client_name)

-- reverse index
create reverse index r_idx on client2(client_name)
```

## ● CUBRID manager client 이용

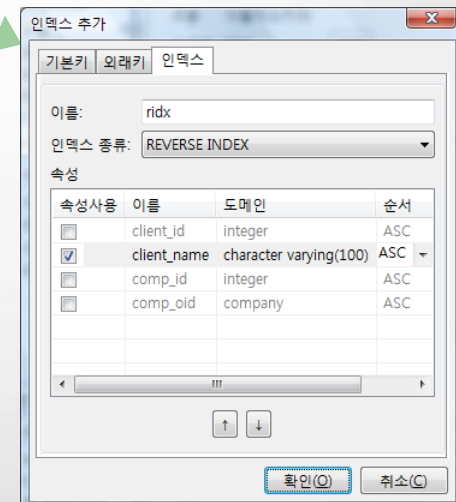
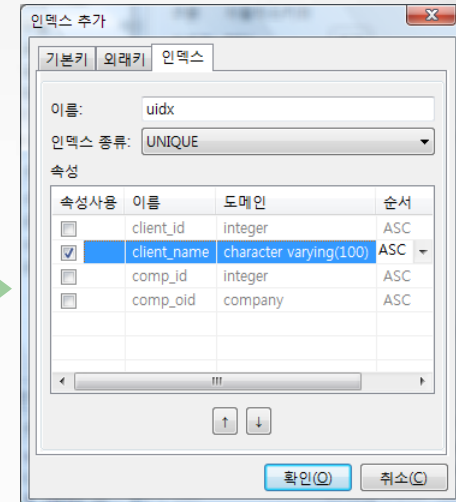
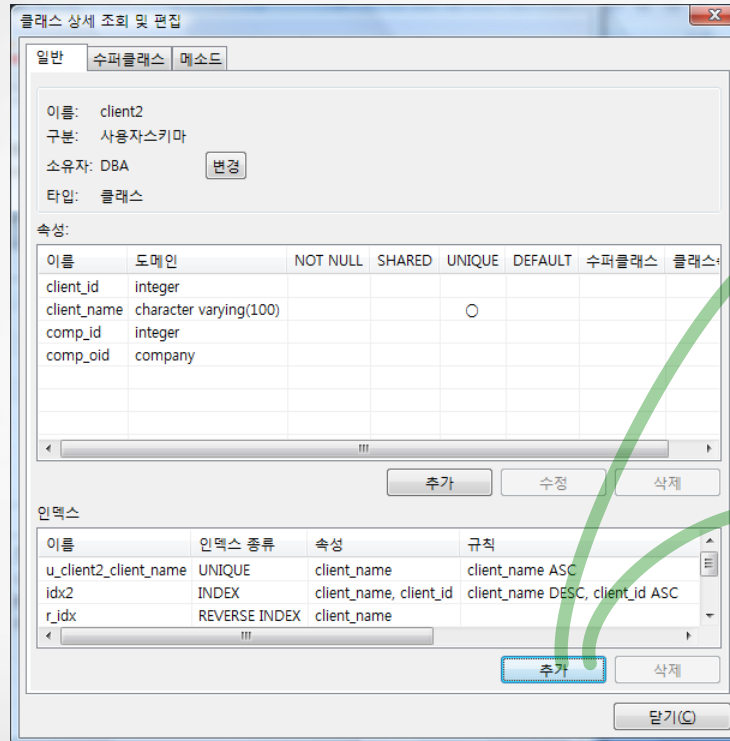
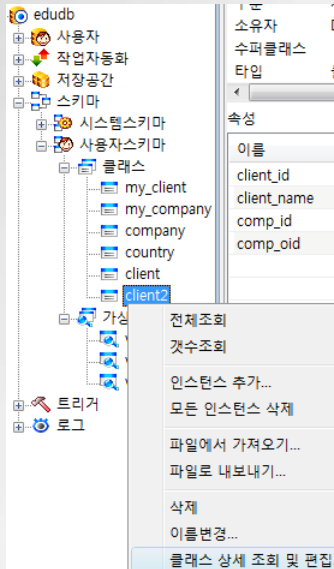


인덱스

이름	인덱스 종류	속성	규칙
u_client2_client_name	UNIQUE	client_name	client_name ASC
idx2	INDEX	client_name, client_id	client_name DESC, client_id ASC
r_idx	REVERSE INDEX	client_name	

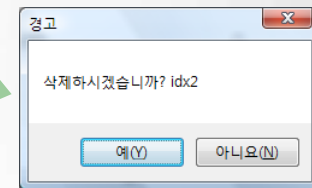
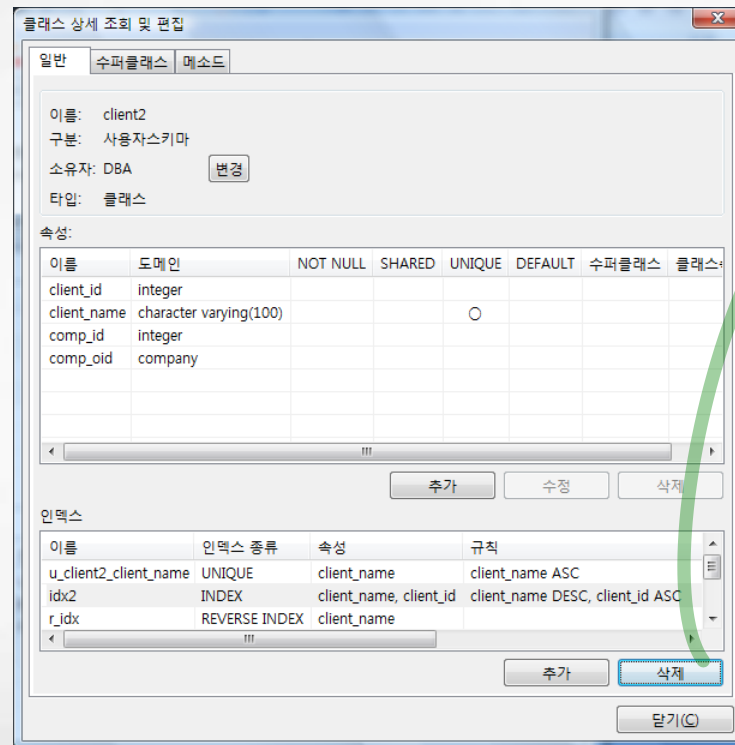
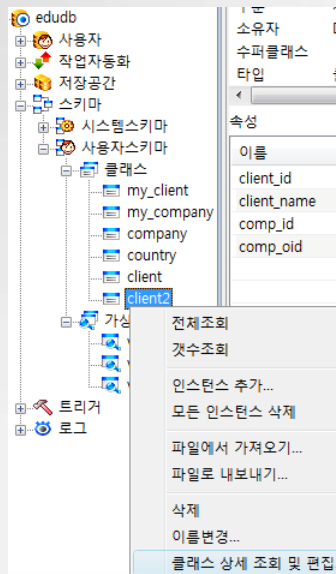


## — unique index, reverse index





## — index 삭제



# 스키마 조작 제한

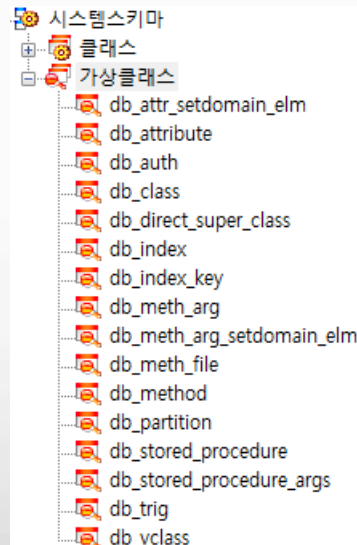


**CUBRID**  
Driving the DBMS REVOLUTION

- 권한이 있는 사용자에게 의하여만 조작
- 환경설정값(cubrid.conf)을 통하여 스키마 조작 제한
  - block\_ddl\_statement 가 1일 경우 스키마 조작 불가 (에러 발생)
  - client parameter



- class 정보
  - db\_class
    - 주요 필드 : class\_name, owner\_name
- 필드 정보
  - db\_attribute
    - 주요 필드 : class\_name, attr\_name, attr\_type
- 기타
  - db\_vclass
  - db\_index
  - db\_index\_key
  - db\_trig
  - db\_partition
  - db\_stored\_procedure
  - db\_auth







- 다음 조건의 테이블 생성
  - 회사 테이블(company)
    - 회사ID(정수):primary key, 회사이름(문자열)
  - 고객 테이블(client)
    - 고객ID(정수):중복되지 않음
    - 고객이름,직위,이메일,전화번호,주소 : 문자열
    - 회사ID : foreign key (update 무시, delete 제한)
  - 일반 고객 테이블, VIP 고객 테이블
    - 고객 테이블과 동일한 내용
    - 추가적으로 포인트(정수)를 가짐
  - CUBRID manager client 에서 테이블 정보 보기



- 아래 조건에 맞게 테이블 수정
  - primary key, foreign key 삭제후 재 생성
    - update 제한, delete 시 같이 삭제, on cache object:comp\_oid
  - type 변경
    - 직위 : char→varchar 또는 varchar → char
  - 초기값 추가/변경
    - 직위 : 초기값 ‘사원’ 으로 지정하였다가 제거
- 인덱스
  - client
    - 고객이름이 유일하고 u\_name 이라는 이름을 가지는 인덱스 추가
    - 직위는 역순, 고객이름 은 순방향 정렬을 위한 idx1 이라는 이름을 가지는 인덱스 추가
- 카다로그를 통한 테이블 정보 검색
  - 생성한 테이블 정보 확인
    - 테이블 이름, 컬럼 정보, 인덱스 정보

## 6. 데이터 검색 및 조작



CUBRID



## ● insert 구문

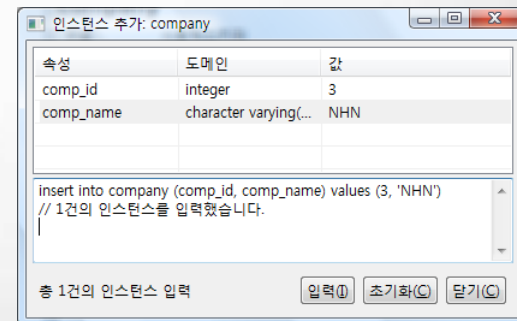
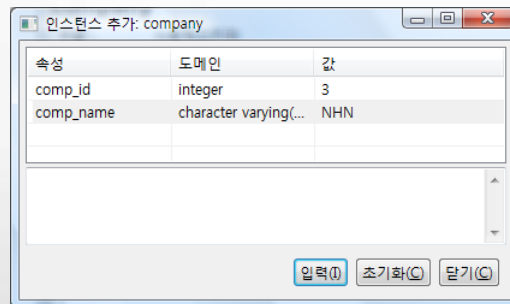
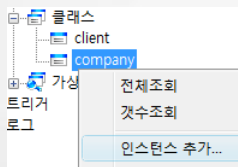
```
INSERT INTO class_name VALUES(value_list)
INSERT INTO class_name(attribute_list) VALUES(value_list);

-- default 값이 선언된 필드는 default 값, 나머지는 null 인 레코드 입력
INSERT INTO class_name DEFAULT [VALUES];
```

## ● 예제

```
insert into company values(1, '큐브리드')
insert into company(comp_id, comp_name) values(2, 'CUBRID')
```

## ● CUBRID manager client 이용





- sub-query 결과를 이용한 입력

```
insert into client(client_id, comp_id, client_name)
values(1, (select comp_id from company where comp_id = 1), '홍길동')
```

- sub-query 를 이용한 입력

```
insert into client(client_id, comp_id, client_name)
select 2, comp_id, '홍길순' from company where comp_id = 1
```



- update 구문

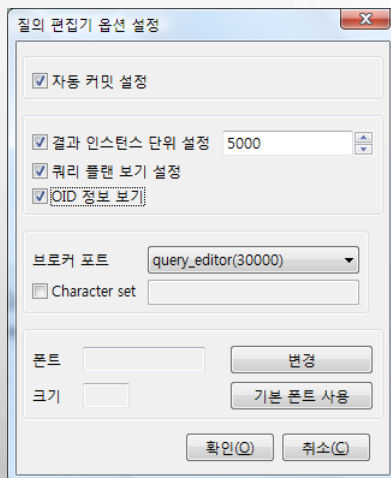
UPDATE 클래스이름 set 변경사항 WHERE 검색조건

- 예제

```
update client set client_name = '홍길자' where client_id = 2  
update client set (comp_id, client_name) = (select comp_id, '홍길순' from company where  
comp_id= 1) where client_id = 2
```

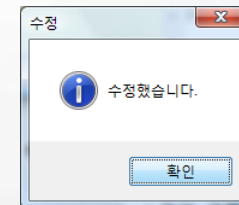
- CUBRID manager client 이용

- 질의편집기 옵션설정 → “OID정보보기 선택” 시 가능
- select 질의 수행후 검색된 결과에 대하여 수정



select \* from client

결과1							
OID	client_id	comp_id	client_name	email	phone	address	coi
@370 1 1	1	1	홍길동	kildong@cubrid.com	NULL	NULL	@:
@370 2 1	2	1	홍길순	NULL	NULL	NULL	@:





- delete 구문

DELETE FROM 클래스이름 WHERE 검색조건

- 예제

delete from company where comp\_id = 3  
delete from company

- where 절 없는 update, delete 수행 제한

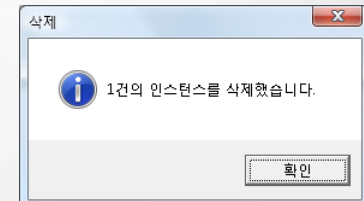
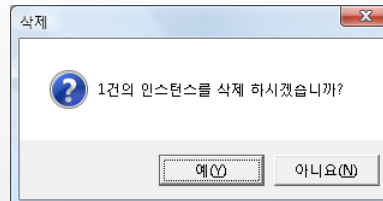
- 환경설정값인 block\_nowhere\_statement 가 1일 경우 수행 불가 (에러 발생)
- client parameter

- CUBRID manager client 이용

결과1

OID	client_id	comp_id	client_name	email	phone	address	coi
@370111	1	1	홍길동	kildong@cubrid.com	NULL	NULL	@:
@370121	1	1	박기승	NULL	NULL	NULL	@:

클립보드에 복사  
삭제





- select 구문

```
SELECT 검색필드명 FROM 클래스이름 WHERE 검색조건
```

- 예제

```
select * from nation  
select name, capital from nation where code = 'KOR'  
  
select athlete_code, medal, score from record
```





- select 절

DISTINCT( or UNIQUE), \*, alias.\*, class\_name.\*

- 예제

```
select distinct event_code from record where host_year = 2004
```

```
select sports, record.* from record, event where host_year = 2004 and code = event_code
```



- from 절

상속관계를 위해 ONLY 또는 ALL, EXCEPT 등의 키워드 사용

- 예제

```
select * from client
select * from only client
select * from all client

select * from all client (except vip_client)
```



## ● where 절

WHERE 절에서는 검색하고자 하는 인스턴스의 조건을 명시한다. 검색 조건으로 올 수 있는 술어(predicate)의 종류는 다음과 같다.

비교 연산자:

NOT, AND, OR, = <> < > >= <=

Predicates:

BETWEEN

IN

LIKE

NULL

```
select * from nation  
select name, capital from nation where code = 'KOR'
```

```
select athlete_code, medal, score from record where host_year = 2004
```



- between

```
select (select name from athlete where code = athlete_code), medal, score from record where  
host_year between 2000 and 2004
```

- in

```
select (select name from athlete where code = athlete_code), medal, score  
from record where host_year = 2004 and event_code in (select code from event where sports  
= 'Swimming')
```

```
select (select name from athlete where code = athlete_code), medal, score  
from record where (host_year, event_code) in (select 2004, code from event where sports =  
'Swimming')
```

- like

```
select name, gender, players from event where sports = 'Wrestling' and name like 'Freestyle%'  
select name, gender, players from event where sports = 'Wrestling' and name like '%48kg'  
select name, gender, players from event where sports = 'Wrestling' and name like '%48%'
```



- sub-query

- select 절이나 where 절에 ()를 이용해 다른 select 문장의 수행

```
select (select name from athlete where code = athlete_code), medal, score from record where  
host_year = 2004
```

```
select distinct event_code, (select sports from event where code = event_code)  
from record where host_year = 2004
```

```
select (select name from event where code = event_code), medal, score, unit  
from record  
where host_year = 1988  
and (event_code, athlete_code) in (select event_code, athlete_code from game where host_year  
= 1988 and nation_code = 'KOR')
```



- NULL

- NULL 과 ‘ ’ 은 서로 같지 않음
- not null default ‘ ’ 활용

```
insert into event values(9998, '수영', '100M', '', 1)
insert into event values(9999, '수영', '200M', NULL, 2)
```

```
select * from event where gender is null
select * from event where gender = ''
select * from event where gender = null
```



- order by 절

```
ORDER BY sort_spec [{, sort_spec} ...]  
    sort_spec:  
        integer_literal  [ASC | DESC]  
        expression  [ASC | DESC]
```

ASC : 오름차순 (default)  
DESC : 내림차순  
숫자는 SELECT 절에서 해당 필드 각각의 위치를 나타낸다

- 예제

```
select name, capital from nation order by name  
  
select name, capital from nation order by continent desc, name
```



- group by 절

GROUP BY expression [HAVING search\_condition]

- > expression은 SELECT절에 반드시 명시해야 한다
- > having은 그룹화된 결과에 대한 조건

- 예제

select sports from event where gender = 'F' group by sports

select sports from event where gender = 'M' group by sports having count(\*) > 1





## ● 조인

- 다른 테이블의 정보를 참조하고 있는 경우 참조하고 있는 다른 테이블의 정보를 가져오는 방법
- 조인조건 : 다른 테이블을 참조하고 있을때 참조하는 필드(FK)와 참조되어지는 필드(PK)의 값을 비교
- inner join : 조인 조건을 모두 만족하는 레코드 검색
  - 88올림픽에 참가한 나라 이름

```
select A.name from nation A, participant B where A.code=B.nation_code and B.host_year=1988
```

```
-- 또는
```

```
-- select A.name from nation A inner join participant B on A.code=B.nation_code where B.host_year=1988
```

결과1

OID	name
NONE	Zimbabwe
NONE	Zambia
NONE	Zaire
NONE	Yemen Democratic Republic
NONE	Yemen Arabian Republic
NONE	Samoa
NONE	Vietnam
NONE	Venezuela
NONE	Vanuatu
NONE	Uruguay
NONE	United Arab Emirates



- outer join : 어느 한쪽만 조인 조건을 만족하는 레코드 검색
  - left outer join : 조인 관계에서 왼쪽 테이블에 대하여만 조인 조건을 만족
  - right outer join : 조인 관계에서 오른쪽 테이블에 대하여만 조인 조건을 만족
  - 만족되지 않는 테이블의 값에 대하여는 그 값을 NULL 로 함
  - 88올림픽에 참가한 나라들의 메달 획득 내용

```
select (select name from nation where code = a.nation_code),
(select name from event where code = b.event_code), medal
from participant a left outer join game b
on a.nation_code = b.nation_code and a.host_year = b.host_year
where a.host_year = 1988
```

결과1			
OID	(select name from natio...	(select name fr...	medal
NONE	Afghanistan	NULL	NULL
NONE	Netherlands Antilles	Division II	S
NONE	Andorra	NULL	NULL
NONE	Angola	NULL	NULL
NONE	Antigua & Barbuda	NULL	NULL
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B
NONE	Argentina	Indoor	B



- derived table (inline view)
  - 질의 수행결과를 가상의 테이블로 만들

```
select sports from (select sports from event group by sports) t
```

```
select name from (select sports from event group by sports) t(name)
```



## ● 질의에서 사용할 인덱스 지정

```
using index {NONE | index_spec [{, index_spec} ...]  
index_spec:  
    [class_name].index_name[(+)]
```

- 지정된 인덱스만 평가, full scan 가능
  - full scan 을 원하는 경우 : NONE
  - 잘못된 인덱스 지정시 에러
- 비용기반의 인덱스 선정
  - 비용을 0으로 강제 설정 : (+) 사용
- 테이블간 인덱스 이름 중복시 <테이블이름>.<인덱스이름>
- join 시 모든 테이블의 인덱스 지정
  - sub 질의는 상관 없음
- index\_scan\_in\_oid\_order=0(cubrid.conf) : 인덱스 순서대로 결과 생성
  - client parameter



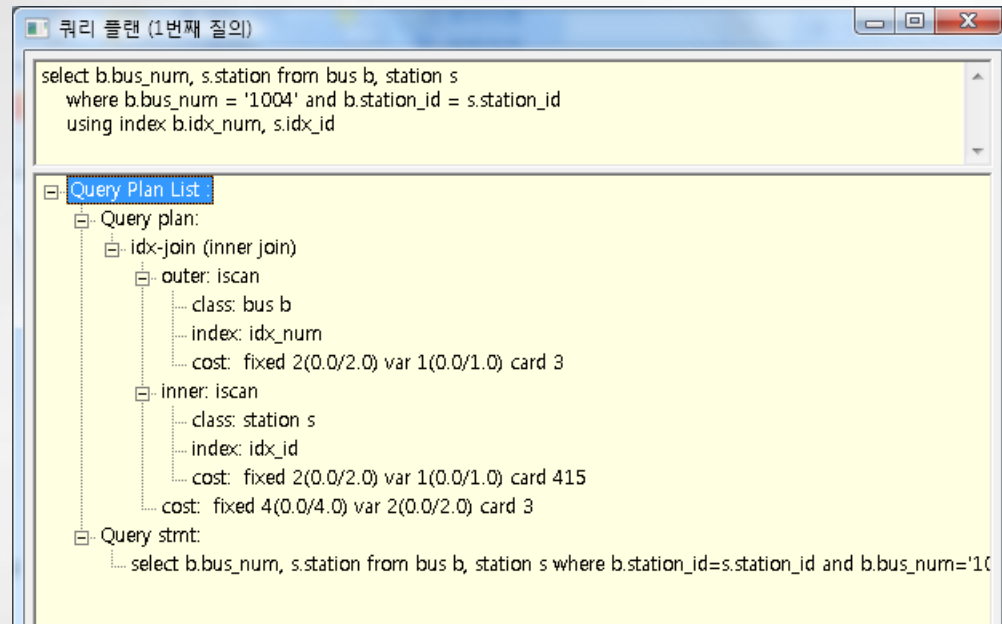
```
create index idx1 on athlete(name)
create index idx1 on record(athlete_code)
```

```
select * from athlete where name = 'Lee Eun-Chul' using index none
select * from athlete where name = 'Lee Eun-Chul' using index idx1
select * from athlete where name = 'Lee Eun-Chul' using index idx1(+)
select * from athlete using index idx1(+)
```

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



- 인덱스 선택 여부 확인
  - CUBRID manager client 이용





## ● 질의 수행 계획 캐쉬

- 질의 수행시 생성되는 질의 수행 계획 캐쉬
- 동일한 질의 수행시 캐싱된 질의 수행 계획 사용으로 질의 수행 계획 생성 시간 절약을 통한 성능 향상
- 환경설정(cubrid.conf)
  - max\_plan\_cache\_entries
    - 쿼리플랜을 캐쉬에 저장하도록 설정하는 파라미터
    - 0보다 큰 값일 경우 설정값 만큼의 쿼리플랜을 캐쉬에 저장한다.
    - default 값은 1000으로 1000개의 질의에 대한 질의수행계획을 캐쉬에 저장한다.
- 캐쉬를 사용하지 않고 질의 수행 계획을 질의 수행시 생성하도록 설정
  - 질의에 /\*+ RECOMPILE +/- 사용
    - select /\*+ RECOMPILE \*/ \* from record where ...



## ● 질의 결과 캐쉬

- 동일한 사용자의 동일한 질의에 대하여 질의 수행결과를 캐쉬
- 질의 플랜 캐쉬 설정시 사용 가능
  - max\_plan\_cache\_entries >= 1
- 동일한 질의 수행시 질의를 재수행하지 않고 캐쉬된 결과를 이용함으로써 수행 성능 향상
  - 검색위주의 시스템에 유용
  - 일반적인 업무(입력/수정/삭제가 빈번한 경우)의 경우 캐쉬 내용 업데이트에 따른 over-head 증가로 인한 성능 감소 가능성
- 질의결과 캐쉬 되지 않는 경우
  - sys\_date, current\_time, current\_user 와 같이 질의 수행시 마다 값이 변하는 함수가 사용된 경우
  - method, JAVA stored procedure 가 사용된 경우
  - prepare 단계에서 host 변수에 대한 data type을 알아내기 어려운 경우
  - 질의 수행 계획 캐쉬가 갱신되는 경우
    - /\*+ RECOMPILE +/- 사용시





- 환경설정(cubrid.conf)
  - max\_query\_cache\_entries
    - 캐쉬에 저장할 질의수행결과 최대 개수
    - 0보다 큰 값일 경우 설정값 만큼의 질의수행결과를 캐쉬에 저장한다.
    - default 값은 -1로 사용하지 않는다.
  - query\_cache\_mode
    - 질의 결과 캐싱 방법 설정
    - 1 : 모든 질의에 대하여 결과 캐쉬
    - 2 : 질의상에 명시(/\*+ QUERY\_CACHE(1) \*/)된 질의만 결과 캐쉬
      - » select /\*+ QUERY\_CACHE(1) \*/ \* from ...



- JDBC 를 통한 질의 수행 결과를 CUBRID Broker 에 캐쉬
  - 환경설정(cubrid\_broker.conf)
    - jdbc\_cache
      - default 는 0으로, 0인 경우 캐쉬하지 않는다.
      - 1일경우 질의 수행 결과를 캐쉬한다.
    - jdbc\_cache\_only\_hint
      - jdbc\_cache 가 1일 경우 유효함.
      - default 는 0이며, 모든 질의에 대하여 캐쉬한다.
      - 1일 경우 질의 힌트에 캐쉬가 명시된 경우에만 캐쉬한다.
    - jdbc\_cache\_life\_time
      - 캐쉬의 유효 시간을 ms 단위로 설정
  - 질의상의 힌트 설정
    - /\*+ JDBC\_CACHE \*/ : 캐쉬를 사용하고, jdbc\_cache\_life\_time 동안 유효함
    - /\*+ JDBC\_CACHE(life\_time) \*/ : 캐쉬를 사용하고, 주어진 life\_time 동안 유효함.



- 일정시간동안 발생한 여러 커밋을 한꺼번에 처리
- 단일 커밋을 바로 처리하지 않음
  - 사용자의 커밋에 대한 응답은 그룹커밋 수행시점에 발생
- 커밋처리가 많은 응용의 사용시 빈번한 커밋처리로 인한 성능 저하 감소를 통한 처리 성능 향상
- 설정(cubrid.conf)
  - group\_commit\_interval\_in\_msecs
    - 밀리세컨드 단위로 그룹커밋 수행시간 설정
    - default 는 0이며, 0인 경우 그룹커밋을 수행하지 않음.
    - 지정된 시간동안 모여진 커밋을 한꺼번에 수행



- 커밋 수행시 커밋 수행 완료를 기다리지 않고 바로 수행 완료 처리
- 실질적인 커밋 수행은 커밋 스레드에 의하여 처리
- 장애 발생시 데이터 유실 가능성
- 데이터 유실 가능성이 없는 독립적인 대량의 데이터 입력에 적합
- 설정(cubrid.conf)
  - `async_commit`
    - default 는 0이며, 0인 경우 일반적인 커밋 수행
    - 1로 설정시 비동기 커밋 수행



## ● 데이터 입력

- company 테이블에 (10, ' 회사10' ), (20, ' 회사20' )를 입력
- client 테이블에 임의의 id,이름과 comp\_id가 10인 회사id를 sub-query 형태로 입력
- client 테이블에 임의의 id,이름과 comp\_id가 20인 회사id를 insert-select 형태로 입력
- client 테이블 검색하여 각 데이터 입력정보 확인

## ● 데이터 수정

- client 테이블에 임의의 id,이름을 입력
- client 테이블을 검색하여 각 데이터 입력정보 확인
- 앞서 입력한 client 테이블의 데이터에 comp\_id를 10으로 수정
- client 테이블을 검색하여 각 데이터 입력정보 확인



- 데이터 검색
  - 88올림픽에 참가한 나라중 메달을 획득한 나라와 메달 정보 검색
    - 올림픽 참가 나라 테이블 : participant
    - 메달 획득 정보 테이블 : game
  - 88올림픽에 참가한 나라들의 메달 획득 정보 검색
- 인덱스 지정
  - 올림픽 개최 도시에 대하여 연도순으로 정렬
    - 올림픽 개최 도시 테이블 : olympic
  - 올림픽 개최 도시에 대하여 최근연도를 먼저나오도록 정렬

## 7. 연산자와 함수



CUBRID



- 비교연산자
  - 필드값등을 다른 값과 비교하기 위해 사용

비교 연산자	의미
=	equal to
<>	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to





## ● 산술연산자

- 필드값등을 다른 값과 더하거나 빼기 등을 위해 사용

산술 연산자	의미
+	수를 더하거나, 문자열을 합치거나, 날짜(시각)을 더함 날짜+날짜, 시각+시각 : 에러 날짜+수 : 날짜에 일자를 더한 날짜 시각+수 : 시각에 초를 더한 시각
-	수를 빼거나, 날짜(시간)을 뺌 날짜-날짜 : 날짜수 시각-시각 : 초 날짜-수 : 날짜에 일자를 뺀 일자 시각-수 : 시각에 초를 뺀 시각
*	수를 곱함
/	수를 나눔(0으로 나누면 에러)



- 병합연산자

병합 연산자	의미
+,	두 개의 문자열을 합침 NULL    'string' → NULL

- 집합연산자

집합 연산자	의미
UNION	두 질의의 결과를 합침. 중복 제거
UNION ALL	두 질의의 결과를 합침. 중복 제거 <b>않</b> 함
DIFFERENCE	앞부분 질의 결과에서 뒷부분 질의 결과를 제거
INTERSECTION	두 질의에 공통적으로 들어간 결과



## ● CAST

- 필드값등을 다른 도메인(타입)으로 변환
  - 문자열 타입의 길이 변환
  - 문자열 타입의 숫자형 변환 제약

```
select name, cast( name as char(40) ) from nation
name                                cast(name as char(40))
=====
'Serbia'                            'Serbia'
```

```
select cast(1 as varchar(10) ) from db_root
select cast( '1' as int) from db_root
select cast( '1 ' as int) from db_root -- 에러
select cast(1.2 as numeric(5,2)) from db_root
```



- `to_char(expression [, format [, language]])`
  - 시간/날자/숫자형 데이터를 문자형으로 변환
- `to_date(expression [, format [, language]])`
  - 문자형 데이터를 날짜형으로 변환
- `to_time(expression [, format [, language]])`
  - 문자형 데이터를 시간형으로 변환
- `to_timestamp(expression [, format [, language]])`
  - 문자형 데이터를 timestamp 형으로 변환
- `to_number(expression [, format])`
  - 문자형 데이터를 숫자형으로 변환



```
create class test_class (  
    d date,      t time,      ts timestamp,  
    ds string,   dt string,   dts string )  
insert into test_class values(sysdate, systime, systimestamp, '12/25/2008', '13:30:01', '20071230')  
  
select to_char(d) from test_class  
select to_char(d, 'yyyy/mm/dd DY') from test_class  
select to_char(d, 'yyyy/mm/dd DY', 'Ko_KR') from test_class  
  
select to_char(t, 'HH24"시"MI"분"SS"초"') from test_class  
  
select to_number(to_char(t, 'ss'), '99') from test_class  
select to_number(to_char(t, 'ss'), '00') from test_class  
  
select to_date(ds, 'MM/DD/YYYY') from test_class  
select to_time(dt, 'HH:MI:SS') from test_class -- 예러  
select to_time(dt, 'HH24:MI:SS') from test_class  
select to_date(dts, 'YYYYMMDD') from test_class
```



## ● date/time/timestamp 변환 형식

형식	설명	En_US	Ko_KR
CC	세기	21	21
YYYY, YY	4자리 년도, 2자리 년도	2008, 08	2008, 08
Q	분기 (ex. '1월 ~ 3월' = 1)	1	1
MM	月(숫자 01-12)	01	01
MONTH	月の 이름(9자리 문자열)	JANUARY	1월
MON	月 이름의 축약어	JAN	1
DD	日(숫자 01-31)	01	01
DAY	요일	MONDAY	월요일
DY	요일 축약어	MON	월
AM or PM	오전/오후	AM	오전
HH or HH12	시간(1-12)	1	1
HH24	시간(0-23)	0	0
MI	분(0-59)	0	0
SS	초(0-59)	0	0
- / , . ; : "text"	임의의 문자열		



## ● 숫자형 변환 형식

형식	설명	En_US	Ko_KR
9	'9' 의 개수만큼 숫자가 반환, 앞자리는 비움	9999	9999
0	'9' 의 개수만큼 숫자가 반환, 앞자리는 0으로 채움	0999	0999
S	+/-가 앞에 붙는다.	S9999	S9999
C	ISO currency가 앞에 붙는다	C9999	C9999
,	콤마 삽입	9,999	9,999
.	소수점 삽입	9.999	9.999
EEEE	Scientific notation 반환	9.99EEEE	9.99EEEE



- extract()

- 날짜나 시간 형의 데이터에서 특정 부분(연,월,일,시,분,초)의 값을 꺼냄

```
/*
create class test_class (
    d date,      t time,      ts timestamp,
    ds string,   dt string,   dts string )
insert into test_class values(sysdate, systime, systimestamp, '12/25/2008', '13:30:01', '20071230')
*/

select extract(YEAR from d) from test_class
select extract(MONTH from d) from test_class
select extract(DAY from d) from test_class

select extract(HOUR from t) from test_class
select extract(MINUTE from t) from test_class
select extract(SECOND from t) from test_class
```





- add\_months()
  - 주어진 날짜에 달을 더한 후의 날짜
- last\_day()
  - 주어진 날짜의 달의 마지막 날짜
- months\_between()
  - 주어진 날짜간의 개월수 (예, 3.2개월차)
- sys\_date, sys\_time, sys\_timestamp
  - 현재 날짜, 시간, timestamp
    - sysdate, systime, systimestamp,
    - current\_date, current\_time, current\_timestamp

```
select add_months(d, 2) from test_class  
select last_day(d), last_day(date '02/01/2008') from test_class
```

```
select months_between(d, date '2008-12-25') from test_class // 결과: 음의 실수  
select months_between(date '2008-12-25', d) from test_class // 결과: 양의 실수
```

```
select sysdate, sys_time from db_root
```



- mod()
  - 주어진 수를 두번째 인자로 나눈 나머지
- rand(), drand()
  - 한 질의에 대하여 난수값 발생. 결과가 여러 개일 경우 동일한 난수값
  - rand() : 임의의 정수 난수값
  - drand() : 0 ~ 1 사이의 임의의 실수 난수값
- random(), drandom()
  - 한 질의결과에 대하여 난수값 발생. 결과가 여러 개일 경우 서로 다른 난수값
  - random() : 임의의 정수 난수값
  - drandom() : 0 ~ 1 사이의 임의의 실수 난수값

```
select mod(9, 4) from db_root  
select mod(9, -4), mod(-9, 3), mod(-9, -2) from db_root  
select mod(3, 0) from db_root
```

```
select rand() from db_class where rownum < 10  
select random() from db_class where rownum < 10  
select drand() from db_class where rownum < 10  
select drandom() from db_class where rownum < 10
```



- **greatest()**
  - 주어진 값들중 가장 큰 수
- **least()**
  - 주어진 값들중 가장 작은 수
- **power()**
  - 제곱값
- **sign()**
  - 주어진 숫자의 부호
- **abs()**
  - 절대값

```
select greatest(1,2,3) from db_root  
select least(1,2,3) from db_root
```

```
select power(2, 9) from db_root // 29
```

```
select sign(1), sign(0), sign(-1) from db_root
```

```
select abs(1.345), abs(-2) from db_root
```



- **ceil()**
  - 주어진 값보다 크거나 같은 정수
- **floor()**
  - 주어진 값보다 작거나 같은 정수
- **round()**
  - 주어진 수를 소수점 기준 두번째 인자 자리수까지 남기고 그 다음 자리에서 반올림
- **trunc()**
  - 주어진 수를 소수점 기준 두번째 인자 자리수까지 남기고 그 다음 자리부터는 버림

```
select ceil(1.3), ceil(1) from db_root  
select floor(1.3), floor(1) from db_root
```

```
select round(1.235, 2), round(1.234, 2) from db_root  
select trunc(1.235, 2), trunc(1.234, 2) from db_root
```

```
select round(125.5, -1), round(123.4, -1) from db_root  
select trunc(125.5, -1), trunc(123.4, -1) from db_root
```

# Aggregation 함수



CUBRID  
Driving the DBMS REVOLUTION

- avg()
  - 그룹핑된 레코드들에 대하여 주어진 인자의 평균값
- count()
  - 그룹핑된 레코드의 개수
- min()
  - 그룹핑된 레코드들에 대하여 주어진 인자의 최소값
- max()
  - 그룹핑된 레코드들에 대하여 주어진 인자의 최대값
- sum()
  - 그룹핑된 레코드들에 대하여 주어진 인자의 합계

```
select sum(gold), max(gold), min(gold), avg(gold), count(*)  
from participant  
where host_year = 1988
```

# Aggregation 함수 (계속)



CUBRID  
Driving the DBMS REVOLUTION

- variance()
  - 그룹핑된 레코드들에 대하여 주어진 인자의 분산
- stddev()
  - 그룹핑된 레코드에 대하여 주어진 인자의 표준편차

```
select variance(gold), stddev(gold)
from participant
where host_year = 1988
```



- length()
  - 문자열 길이를 구함 (byte 단위)
- position()
  - 주어진 문자열에서 다른 문자열의 시작위치를 구함
    - 반환값: 성공시 >0, 실패시 0, 찾는 문자열이 ' ' 인 경우 1, NULL 이면 NULL
- instr()
  - position() 과 동일하나 문자열 찾는 위치 지정 가능
    - 찾는 위치가 - 일 경우 뒤에서 부터 위치를 찾아 그곳에서 부터 문자열을 찾음

```
select length('문자열length') from db_root
```

```
select position('A' in 'Abcd') from db_root  
select position('A' in 'abcd') from db_root
```

```
select instr('Abcd', 'A') from db_root  
select instr('Abcd', 'A', 2) from db_root  
select instr('AbcdA', 'A', -2) from db_root
```



- substring()

- 주어진 문자열을 특정 위치에서 주어진 개수만큼 잘라 넘겨줌
  - 위치가  $\leq 0$  인 경우 1로 치환
  - 개수는  $< 0$ 이면 무시되고 0인 경우 ‘ ’를 넘겨줌

- substr()

- substring() 과 동일하나, 위치나 개수가  $\leq 0$  일 경우 동작방식이 다름
  - 위치가 0일 경우 1로 치환,  $< 0$  인 경우 뒤에서부터 위치 찾음
  - 개수가 0이면 ‘ ’를 넘겨주고,  $< 0$ 이면 NULL 을 넘겨줌

```
select substring('abcde' from 3) from db_root
select substr('abcde', 3) from db_root
```

```
select substring('abcde' from 3 for 1) from db_root
select substr('abcde', 3, 1) from db_root
```

```
select substring('abcde' from 0 for -1) from db_root
select substr('abcde', 0, -1) from db_root
```

```
select substring('abcde' from 3 for 0) from db_root
select substr('abcde', 3, 0) from db_root
```

```
select substring('abcde' from -3 for 2) from db_root
select substr('abcde', -3, 2) from db_root
```





- upper(), lower()
  - 문자열을 대문자, 소문자로 변환
- trim(), ltrim(), rtrim()
  - 문자열의 양쪽, 왼쪽, 오른쪽에 대하여 trim 문자열을 제거
- lpad(), rpad()
  - 문자열의 왼쪽, 오른쪽에 문자열 추가

```
select upper('abcde123'), lower('ABCDE123') from db_root
```

```
select trim(' abc '), ltrim(' abc '), rtrim(' abc ') from db_root
```

```
select trim(both from ' abc '), trim(leading from ' abc '), trim(trailing from ' abc ') from db_root
```

```
select trim('ag' from 'agbcdefag') from db_root
```

```
select ltrim('agbcdefag', 'ag') from db_root
```

```
select rtrim('agbcdefag', 'ag') from db_root
```

```
select lpad('abcde', 10) from db_root
```

```
select lpad('abcde', 10, '*') from db_root
```

```
select lpad('', 10, '*'), lpad(NULL, 10, '*'), lpad(NULL, 10, NULL) from db_root
```

```
select lpad('abcde', 3), lpad('abcde', -2) from db_root
```



- **replace()**
  - 주어진 문자열에서 검색문자열을 찾아 치환문자열이 있는 경우 치환하고, 없는 경우 삭제
- **translate()**
  - 주어진 문자열에서 검색문자열의 각문자에 대하여 치환문자열의 대응되는 각문자로 치환하고, 대응하는 문자가 없는 경우 삭제

```
select replace('abcde', 'bd') from db_root  
select replace('abcde', 'bc', 'BBCC') from db_root  
select replace('abcde', 'bc') from db_root
```

```
select translate('abcde', 'bd', '') from db_root  
select translate('abcde', 'bd', 'B') from db_root  
select translate('abcde', 'bd', 'BD') from db_root
```



- case
  - 주어진 조건(조건식사용가능)에 따라 결과값을 결정
- decode()
  - 주어진 값에 따라서 결과 값을 결정

```
select case gender when 'M' then '남성'
                when 'F' then '여성'
```

```
end
```

```
from event
```

```
select decode(gender, 'M', '남성', 'F', '여성') from event
```

```
select decode(gender, 'M', '남성', 'F', '여성', '중성') from event
```

```
select case when host_year between 1900 and 1999 then '1900년대'
```

```
        when host_year >= 2000 then '2000년대'
```

```
        else '알수없는연대' end
```

```
from record
```



- coalesce
  - 주어진 값이 NULL 인 경우 다른 값으로 변환
- nvl
  - coalesce 와 동일
- nvl2
  - 주어진 값이 NULL 이 아니면 첫번째 인자값, NULL 이면 두번째 인자값

```
select (select name from nation where code=a.nation_code),  
(select name from event where code = b.event_code), nvl(medal, 'No medal')  
from participant a left outer join game b on a.nation_code = b.nation_code and a.host_year =  
b.host_year  
where a.host_year = 1988
```

```
select (select name from nation where code=a.nation_code),  
(select name from event where code = b.event_code), nvl2(medal, 'Medal', 'No medal')  
from participant a left outer join game b on a.nation_code = b.nation_code and a.host_year =  
b.host_year  
where a.host_year = 1988
```



- nullif()
  - 주어진 두개의 값이 같으면 NULL, 다른 경우 첫번째 인자

```
select * from my_class where nullif(attr1, '') is null
```

```
select * from my_class where nvl(attr1, '') = ''
```



- rownum
  - 질의 수행결과 생성되는 결과 튜플에 대하여 일련번호 부여
  - order by, group by 전에 생성되므로 order by, group by 사용시 결과 주의
  - 임의 번째의 결과 선택 가능

```
select * from history where rownum <= 10  
select * from history where rownum between 11 and 20
```

```
select athlete,score from history  
where inst_num() between 2 and 10 group by athlete,score
```

```
select athlete,score from history  
order by score for orderby_num() between 3 and 10
```



- serial

- 일련번호 생성
- 트랜잭션의 영향을 받지 않음
  - rollback 시 사용된 번호 반환되지 않음
- 생성

```
CREATE SERIAL serial_identifier  
    [START WITH n]  
    [INCREMENT BY n]  
    [MINVALUE n | NOMINVALUE]  
    [MAXVALUE n | NOMAXVALUE]  
    [CYCLE | NOCYCLE]
```

START WITH n : 일련번호의 시작 번호 값을 n으로 지정  
INCREMENT BY n : 생성되는 번호들의 증감 폭을 n으로 지정(default : 1)  
MINVALUE n : 최소값을 n으로 설정(default : 1,  $-10^{36}$ )  
MAXVALUE n : 최대값을 n으로 설정(default :  $10^{36}$ , -1)  
CYCLE/NO CYCLE : 최대(최소)값에 도달했을 경우, 값을 재 생성 여부 지정



## – 수정

- MINVALUE 는 현재값보다 같거나 작아야 함
- MAXVALUE 는 현재값보다 같거나 커야 함

```
ALTER SERIAL serial_identifier  
  [START WITH n]  
  [INCREMENT BY n]  
  [MINVALUE n | NOMINVALUE]  
  [MAXVALUE n | NOMAXVALUE]  
  [CYCLE | NOCYCLE]
```

START WITH n : 일련번호의 시작 번호 값을 n으로 지정  
INCREMENT BY n : 생성되는 번호들의 증감 폭을 n으로 지정(default : 1)  
MINVALUE n : 최소값을 n으로 설정(default : 1,  $-10^{36}$ )  
MAXVALUE n : 최대값을 n으로 설정(default :  $10^{36}$ , -1)  
CYCLE/NO CYCLE : 최대(최소)값에 도달했을 경우, 값을 재 생성 여부 지정

## – 삭제

```
DROP SERIAL serial_identifier
```





- serial 개체 접근

- current\_value / currval : 현재값
- next\_value / nextval : 다음값
  - 현재값을 증가치만큼 증가시킨후 증가된 값을 돌려줌
  - 생성후 처음 사용시는 초기값을 그대로 돌려줌 (증가시키지 않음)

```
create serial s_no  
  
select s_no.next_value from db_root  
select s_no.current_value from db_root
```

- 현재값 변경

- 시스템 테이블 직접 수정

```
update db_serial set current_val = 100 where name = 's_no'
```



## ● AUTO\_INCREMENT

- 레코드 입력시 자동증가되는 숫자 필드 지정
  - smallint, int, numeric(p,0)
- 시작값, 증가값 지정 가능
  - default : 시작값=1, 증가값=1
  - AUTO\_INCREMENT(10, 2)
- 사용자 임의의 값 입력 허용
  - 자동증가시 중복된 값 입력 가능성 : unique 등 제한 필요

```
create class bbs (  
    id int auto_increment,  
    title      string,  
    cnt        int default 0  
)
```

```
insert into bbs(title) values( '자동증가 자동입력' )
```

```
insert into bbs(id, title) values(1, '자동증가 임의입력' )
```



- **incr()**
  - 정수형 필드에 대하여 select 절에 사용시 해당 필드의 값을 1씩 자동 증가
    - 현재값을 돌려준 후 증가
    - select 수행후 값을 증가시킴
- **WITH INCREMENT FOR**
  - incr() 과 동일한 효과
    - incr() 과 같이 사용가능하나 2번 증가
  - where 절 다음에 위치

```
select title, incr(cnt) from bbs where id = 1  
select title, cnt from bbs where id = 1 with increment for cnt  
  
select title, incr(cnt) from bbs where id = 1 with increment for cnt
```



- `current_user / user`
  - 데이터베이스에 대한 현재 사용자 ID

```
select current_user from db_root
```

```
select * from db_user where name = current_user
```



## ● 산술/병합/형변환 연산자

- 올 크리스마스까지 몇 달, 며칠이 남았는지 확인
- 교육이 끝나려면 몇 시간이 남았는지를 xx시간xx분xx초로 표시
- 올해가 몇년도인지를 두가지 이상의 방법으로 구함
- 이번달의 마지막날이 며칠인지 확인

## ● 함수

- 1~100 사이의 임의의 숫자 구하기
- 다음 값(3.141592653)을 소수 7자리에서 반올림과 버림
- 10번 버스를 탈 수 있는 승강장의 개수 구하기
- 다음 문자열( 'substring xyzxy' )의 길이, 'str' 의 위치, 4번째부터 6개의 문자열 추출, 문자열뒤의 'xy' 제거, 's' 를 'S' 로 치환
- 올림픽 결과 수여한 메달에 대하여 'G':'금메달', 'S':'은메달', 'B':'동메달' 로 표시
  - 올림픽 메달 테이블 : game
- 올림픽 개최 연도가 1900년대이면 '1900년대', 2000년대이면 '2000년대', 그외는 '기타 연대' 로 표시
  - 올림픽 개최 연도 테이블 : olympic



- rownum

- 올림픽개최정보를 11번째 부터 20번째까지 보여줌
  - 올림픽개최정보 테이블 : olympic
- 올림픽개최정보를 연도순로 정렬하여 그중 11번째 부터 20번째 까지 보여줌
- 성능을 고려하여 바로 위의 질의 수정
- 올림픽개최정보를 국가별로 그룹핑하고 그중 11번째 부터 20번째 까지 보여줌

- serial

- 임의의 serial 개체를 하나 만든후 다음값은 얻고, 현재값을 확인한다.
- serial 의 다음값을 얻은후 rollback 후 현재값을 확인(autocommit off)

- 자동증가, 클릭카운트

- 임의의 테이블을 만들고 자동증가 필드 및 카운트 필드 생성
- 데이터 입력 (자동증가필드에 1입력, 자동증가필드에 입력않함)
- 검색을 통한 데이터 확인 및 데이터 삭제후 재 입력(자동증가필드 입력않함)
- 클릭카운터를 이용하여 카운트 값 증가 및 확인

## 8. 트리거



CUBRID



- 데이터베이스 요청 처리시 사용자가 정의한 행위를 수행하도록 지정
- 일반적인 사용 목적
  - 무결성 제약조건 과 세션 제약조건 강제 수행
  - 병렬 갱신 수행
  - 통계정보 수집 및 관리
  - 단계적인 삭제 수행
- 생성구문

```
CREATE TRIGGER trigger_name  
[ STATUS { ACTIVE | INACTIVE } ]  
[ PRIORITY key ]  
event_time event_type [ event_target ]  
[ IF condition ]  
EXECUTE [ AFTER | DEFERRED ] action [ ; ]
```

event\_time : BEFORE, AFTER, DEFERRED

event\_type : DELETE, STATEMENT DELETE, INSERT, STATEMENT INSERT, UPDATE,  
STATEMENT UPDATE, ROLLBACK, COMMIT

event\_target : ON class\_name , ON class\_name ([ CLASS ] attribute\_name)

condition : expression

action : REJECT, INVALIDATE TRANSACTION, PRINT message, CALL statement, INSERT statement,  
UPDATE statement, DELETE statement, EVALUATE statement





- 트리거 이름
  - 유일한 이름 사용, 예약어 사용불가, 클래스이름과 중복 가능
- 상태 (선택사항, default:active)
  - active: 트리거 수행, inactive: 트리거 수행 안함
- 우선순위 (선택사항, default:0.0)
  - 수행가능한 트리거의 수행 우선 순위 지정
    - 0보다 크거나 같은 실수형으로 부여
- 이벤트 시점
  - 데이터베이스 요청처리시 트리거 수행 시점 지정
    - before : 요청처리전
    - after : 요청처리후
    - deferred : 트랜잭션 처리시, 이벤트 타입이 commit/rollback 시에는 사용 불가
- 이벤트 종류
  - 데이터베이스 요청의 종류
    - 레코드 단위 : insert, update, delete
    - SQL 문장 단위 : statement insert, statement update, statement delete
    - 트랜잭션 처리 : commit, rollback



- 이벤트 대상
  - 테이블 이름
    - update 의 경우 필드명 지정 가능 : 테이블이름(필드명)
  - 이벤트 종류가 commit/rollback 일 경우 이벤트 대상 없음
- 조건 (선택사항)
  - 트리거 action 부분 수행 여부 판단
    - else 절 없음, else 절이 필요한 경우 여러 개의 트리거로 분리, 필요시 우선순위 지정
- correlation name
  - 질의 처리 대상 레코드에 대한 별칭

	BEFORE	AFTER	DEFERRED
INSERT	NEW	OBJ	OBJ
UPDATE	OBJ/NEW	OLD	OBJ
DELETE	OBJ	N/A	N/A



- action

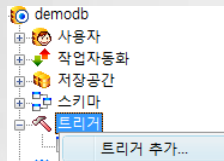
- after/ deferred : 실행 시점 지정
- 실행 내역 지정
  - reject : 실행 취소(이벤트 시점이 before인 경우만 가능), 트랜잭션은 남아있으므로 트랜잭션 처리 필요
  - invalidate transaction : commit 실행 취소, rollback 처리만 가능
    - commit 은 무조건 취소되므로, 해당 트리거 삭제도 commit 되지 못함. 해당 트리거를 inactive 시킨후 삭제 가능
  - print : 화면에 주어진 메시지 출력
  - call : stored procedure 호출
  - insert, update, delete : SQL 문 수행

```
CREATE TRIGGER tr_upd_1
PRIORITY 1.0
before update on record(athlete_code)
IF (select count(*) from athlete where code=obj.athlete_code) < 1
EXECUTE reject
```

```
CREATE TRIGGER tr_ins_1
PRIORITY 1.0
after insert on record
IF (select max(score) from history where host_year = obj.host_year and event_code =
obj.event_code) > obj.score
EXECUTE insert into history(event_code, athlete, host_year, score, unit)
values(obj.event_code, (select name from athlete where code=obj.athlete_code), obj.host_year,
obj.score, obj.unit)
```



## ● CUBRID manager client 이용



**트리거 추가**

트리거는 미리 정의된 데이터베이스 활동에 대해 반응할 행동을 정의하기 위해 생성됩니다. 정의된 데이터베이스 활동이 감지되면 관련된 트리거가 작동을 합니다. 작동된 트리거는 데이터베이스의 또다른 활동을 시작시킬 수 있습니다.

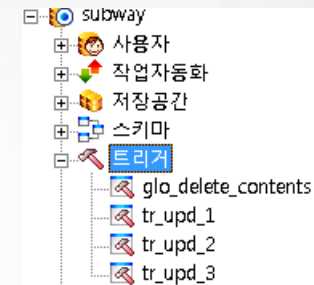
**트리거 추가**

이름	tr_ins
조건 확인 시점	BEFORE
이벤트	INSERT
이벤트 대상	record
조건	and event_code = obj.event_code) > obj.score
수행 시점	-----
내용	OTHER STATEMENT
values(obj.event_code, (select name from athlete where code=obj.athlete_code), obj.host_year, obj.score, obj.unit)	

**추가설정**

<input checked="" type="checkbox"/> 트리거 상태 선택	트리거 상태	ACTIVE
<input checked="" type="checkbox"/> 트리거 우선순위 설정	트리거 우선순위의	0.01

확인(O) 취소(C)





- 이름 변경

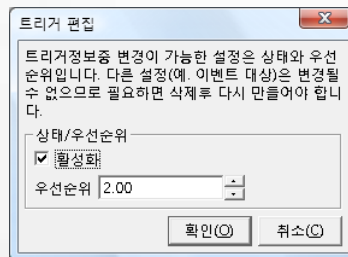
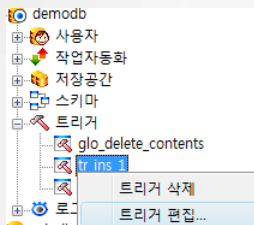
```
rename trigger tr_upd_1 as tr_upd
```

- 설정 변경

- 상태 및 우선순위 조정

```
alter trigger tr_ins_1 status active  
alter trigger tr_ins_1 priority 2.0
```

- CUBRID manager client 이용

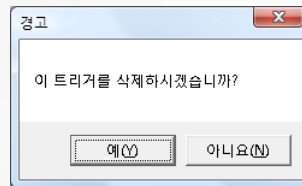
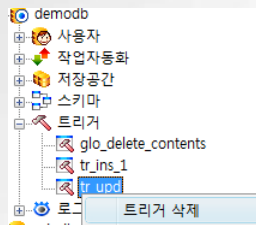




## ● 삭제

```
drop trigger tr_upd
```

– CUBRID manager client 이용



# deferred 트리거 실행/취소



CUBRID  
Driving the DBMS REVOLUTION

- deferred 트리거 실행

- 트랜잭션 처리 시점까지 지연된 트리거를 강제 수행

```
execute deferred trigger tr_deferred_upd1  
execute deferred trigger all triggers
```

- deferred 트리거 수행 취소

- 트랜잭션 처리 시점까지 지연된 트리거가 수행되지 않도록 취소

```
drop deferred trigger tr_deferred_upd1  
drop deferred trigger all triggers
```



- 올림픽 개최정보 입력시 국가가 존재하는지 확인하여 존재하는 경우에만 입력
  - 올림픽 개최정보 테이블 : olympic
  - 국가 테이블 : nation
  - 2008년도 베이징 올림픽 정보 입력
  - 중국 정보 입력
  - 2008년도 베이징 올림픽 정보 입력



## 9. 분할



CUBRID



- 분할(partitioning)
  - 하나의 테이블을 여러 개의 테이블로 나누어 관리
  - partition : 나누어진 각 단위(서브 테이블)
  - 분할표현식 : 분할의 기준이 되는 표현식
  - 분할키 : 분할표현식에 사용되는 필드
    - 한 개의 필드만 지정 가능
  - 각 partition 에 대한 개별 조회 기능
    - 상속을 이용한 구현
    - 인덱스 생성시 각 분할에도 인덱스가 자동 생성
    - 입력,수정,삭제는 불가능
  - 최대 분할수 : 1024
  - 카다로그 : db\_partition



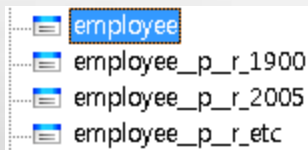
- 목적
  - 대용량 클래스의 관리 향상
  - 조회시 접근 범위를 줄임으로 성능 향상
  - I/O 분산을 통한 성능 향상 및 물리적 부하 감소
  - 여러 partition 으로 나눔으로서 전체 데이터 훼손 가능성 감소
- 분할 기법
  - range / list / hash partitioning



- range partitioning

- 값의 범위를 지정하는 분할표현식을 기준으로 테이블 분할
- 순차적이거나 연속적인 데이터 구조에 적합
- NULL 은 첫번째 분할에 저장됨
- 분할 예

```
create class employee (  
    empno      int,  
    hire_year  int,  
    hire_date  date          // trigger : after insert/update  
                                //      set hire_year = extract(year from hire_date)  
) partition by range(hire_year) (  
    partition r_1900 values less than(2000),  
    partition r_2005 values less than(2006),  
    partition r_etc values less than maxvalue  
)
```

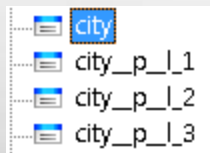




## ● list partitioning

- 일련의 값들 가지는 분할표현식을 기준으로 테이블 분할
- 값의 종류가 많지 않고 분포가 비슷한 경우 사용
- 분할표현식에 명시되지 않은 값 입력시 에러
  - NULL 도 명시하여야 함
- 분할 예

```
create class city (  
    city_id      int,  
    city_name    varchar(20)  
) partition by list(city_name) (  
    partition l_1 values in ( '서울특별시' ),  
    partition l_2 values in ( '부산광역시' , ' 인천광역시' , ' 대구광역시' , ' 광주광역시' , ' 대전광역  
시' , ' 울산광역시' ),  
    partition l_3 values in ( '경기도' ,NULL)  
)
```

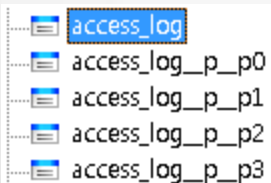




## ● hash partitioning

- 분할표현식에 적절할 해쉬함수를 적용하여 테이블 분할
- 각 분할에 적절하게 데이터가 분포되도록 함
- 관리편의성 보다는 저장, 조회의 성능 중심
- range나 list 를 사용하기 어려운 경우 사용
  - 개수는 2의 배수로 하는 것이 좋음
- 생성 예

```
create class access_log (  
    user_id    varchar(20),  
    contents   varchar(1000)  
) partition by hash(user_id) partitions 4
```





- 분할 → 일반 테이블

- 사용예

```
alter class access_log remove partitioning
```

- 일반 테이블 → 분할

- 분할이 아닌 일반 테이블을 분할로 변경
  - 분할표현식에 따라 기존 데이터 각 분할로 이동 (많은 시간 소요)
    - 분할표현식에 어긋나는 데이터가 있는 경우 분할 실패 (예. list partitions 시 list 에 없는 데이터 존재시)
  - 사용예

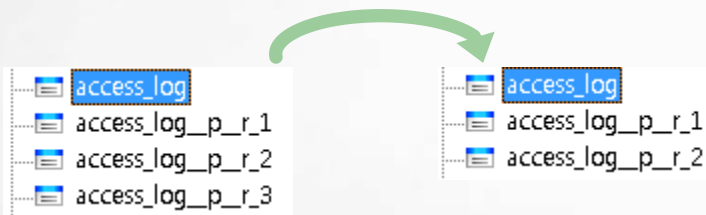
```
alter class access_log  
partition by range(user_id) (  
    partition r_1 values less than ( 'm' ),  
    partition r_2 values less than ( 'u' ),  
    partition r_3 values less than maxvalue  
)
```



## ● 분할 삭제

- 분할된 테이블에서 분할 제거
- 삭제시 해당 파티션에 있던 데이터도 같이 삭제됨
  - range partitioning 의 경우 첫번째 파티션이 삭제되면 이후 NULL 은 새로운 첫번째 파티션에 저장됨
- hash partitioning 의 경우 삭제할 수 없음
- 사용 예

```
alter class access_log drop partition r_3
```







## ● 분할 추가

- 생성된 분할에 새로운 분할을 추가
- range, list partitioning에만 가능
  - range의 경우 기존 range보다 큰 값으로만 추가 가능
- 사용 예

```
alter class access_log add partition (  
    partition r_4 values less than ( 'c' )  
)  
-- 예러
```

```
alter class access_log add partition (  
    partition r_4 values less than ( 'z' )  
)
```





## ● 분할 재구성

- 데이터의 손상없이 분할의 개수를 재조정
  - range, list 의 경우 분할표현식에서 기존 데이터의 손상이 발생하면 에러
- hash partitioning 의 경우 분할 개수 감소의 경우만 가능
- 사용 예

```
alter class employee reorganize partition r_1900, r_2005 into (  
    partition r_2005 values less than(2005)  
)  
-- 에러
```

```
alter class employee reorganize partition r_1900, r_2005 into (  
    partition r_2005 values less than(2006)  
)
```

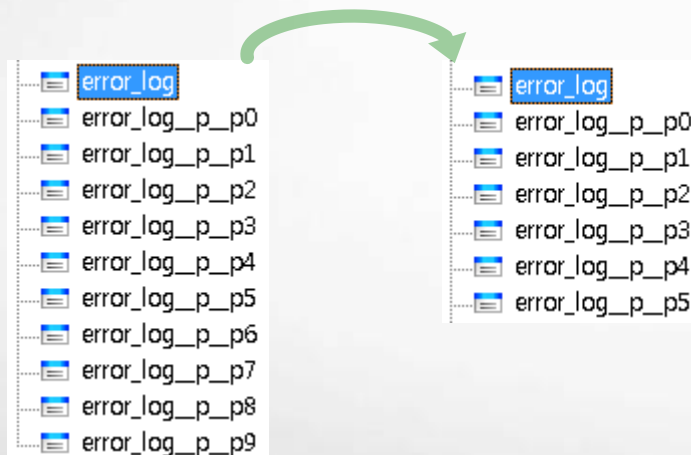
```
alter class employee reorganize partition r_etc into (  
    partition r_2006 values less than(2007),  
    partition r_etc values less than maxvalue  
)
```





## – 사용 예 (hash partitioning)

```
create class error_log (  
    error_code int,  
    error_msg varchar(1000)  
) partition by hash(error_code) partitions 10  
  
alter class error_log coalesce partition 4
```





- 데이터 조회

- 각 분할에 대한 조회 가능
  - 입력/수정/삭제 불가능
- 사용 예

```
select * from access_log__p__r_4  
select * from access_log where user_id >= 'x' user_id < 'z'
```

- 데이터 변경

- 분할키의 값이 변경된 경우, 분할표현식의 결과에 따라 다른 분할로 옮겨질 수 있음
  - 이 경우 삭제와 입력이 이루어 지므로 성능 저하



- 검색조건을 분석하여 검색 범위 한정
  - 분할표현식을 통한 검색대상 분할 선정
    - 검색범위 한정을 통한 처리할 데이터양 감소
- 검색표현식과 동일한 형태로 where 절에 사용시 프루닝 수행
  - 표현식이 extract(year from open\_date) 라면 where 절에서 똑같이 사용
  - range, list partitioning 의 경우 동등비교 또는 범위 비교시
  - hash partitioning 의 경우 동등비교시
- 프루닝이 잘 되지 않는 경우 분할을 지정하여 검색 가능
  - 분할표현식과 다른 형태로 질의시 해당 분할에 직접 질의

```
select * from employee where hire_date >= '01/01/2007'  
-- 프루닝 실패
```

```
select * from employee where hire_year = 2007  
select * from employee__p__r_etc where hire_date >= '01/01/2007'
```



- 아래 조건에 맞는 분할 테이블 작성
  - 고용인 테이블에 id,고용일 을 기본으로 하고 고용연도를 기준으로 분할
  - 우편번호 테이블에 시도,우편번호 를 기본으로 하고 시도를 기준으로 분할
  - 각 테이블에 분할키를 기준으로 조회후 프루닝 여부 확인
    - csq1 에서 확인

# 10. 자바 저장 프로시저



CUBRID



## ● 개요

- SQL 로 처리할 수 없는 복잡한 프로그램 로직 구현 가능
  - 코드 재사용 및 성능 개선
- 함수처럼 사용 가능
- JAVA 기반
  - JAVA 1.4 이상, JAVA 1.6 권장
- 자바 저장 프로시저의 이름의 최대 길이는 256자
- 자바 저장 프로시저 인자의 최대 개수는 64개

## ● 특징

- JAVA 언어의 장점 활용
  - 견고성
  - 생산성
  - 이식성
- 기존 JAVA 코드, 개발환경 재사용
  - 클라이언트 JDBC 응용 → 서버 JAVA 저장 프로시저
- 데이터베이스 비 종속적
- 클라이언트, 서버 데이터베이스 응용 개발 환경 통일





## ● JAVA class 작성

- 데이터베이스 작업을 위해서 CUBRID JDBC driver 로딩
- JDBC connection은 내부 connection(데이터베이스 서버에서의 connection) 사용
  - 내부에서 수행되므로 추가적인 연결 필요 없음

```
import cubrid.jdbc.driver.*;  
...  
Class.forName("cubrid.jdbc.driver.CUBRIDDriver");  
Connection con = DriverManager.getConnection( "jdbc:default:connection:" );
```

- JAVA class 의 메소드는 public static 으로 선언
- 저장프로시저 내부의 트랜잭션 처리는 무시됨
  - 내부 connection 사용시 저장프로시저를 호출한 트랜잭션의 일부로 처리
  - 단, 외부 connection 사용시는 별도의 트랜잭션이므로 트랜잭션 처리 필요
- 결과셋 반환 가능
  - JAVA 응용에서만 결과셋 처리 가능
    - Callable Statement 사용
  - ResultSet 을 CUBRIDResultSet 으로 변환후 setReturnable() 호출
  - stored procedure 의 return type 은 cursor
    - create function sp\_cursor() return cursor ...



- 데이터베이스와 JAVA 타입 비교
  - 인자 및 결과 타입

SQL Type	Java Type
CHAR, VARCHAR	java.lang.String, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.math.BigDecimal, byte, short, int, long, float, double
NUMERIC, SHORT, INT, FLOAT, DOUBLE	java.lang.Byte, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double, java.math.BigDecimal, java.lang.String, byte, short, int, long, float, double
DATE, TIME, TIMESTAMP	java.sql.Date, java.sql.Time, java.sql.Timestamp, java.lang.String
OBJECT	cubrid.sql.CUBRIDOID
CURSOR	cubrid.jdbc.driver.CUBRIDResultSet



## ● 작성 예

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

public class MySP {
    // 각 메소드는 반드시 public static 으로 선언되어야 하며, 성공시 0 을 돌려주므로 return type 을 int 로 한다.
    // argument 로 테이블 이름 및 각 입력값을 받아 입력한다.
    public static int InsertZipcode(String zip_code, String city_name) throws Exception {
        Connection conn = null;
        Statement stmt = null;

        try {
            // jdbc driver 를 지정하여, 로드시킨다.
            Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
            // 데이터베이스의 Connection을 얻는다. 자바 저장 함수와 프로시저가 데이터베이스 내에서 실행되기 때문에
            서버측 JDBC 드라이버를 사용한다.
            conn = DriverManager.getConnection("jdbc:default:connection:");

            String sql = "insert into zip_code(zip_code, city_name) values(";
            sql += "        " + zip_code + "," + city_name + ")";

            stmt = conn.createStatement();
            stmt.executeUpdate(sql);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



```
// stored procedure 내부에서의 commit/rollback 은 의미가 없으므로 생략한다.  
  
} catch ( SQLException e ) {  
    System.err.println(e.getMessage());  
    // 실제 에러 발생시 에러가 return 되지 않고, SQL 문장 에러처럼 처리된다.  
    return e.getErrorCode();  
} catch ( Exception e ) {  
    System.err.println(e.getMessage());  
    return -1;  
} finally {  
    if (stmt != null) stmt.close();  
    if (conn != null) conn.close();  
}  
  
return 0;  
}
```



```
public static String SelectZipcode(String city_name) throws Exception {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    String zip_code = null;

    try {
        Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
        conn = DriverManager.getConnection("jdbc:default:connection:");

        String sql = "select zip_code from zip_code";
        sql += " where city_name = '" + city_name + "'";

        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql);

        if (rs.next()) {
            zip_code = rs.getString("zip_code");
        } else {
            zip_code = "";
        }
    }
}
```



```
} catch ( SQLException e ) {  
    System.err.println(e.getMessage());  
} catch ( Exception e ) {  
    System.err.println(e.getMessage());  
} finally {  
    if (rs != null) rs.close();  
    if (stmt != null) stmt.close();  
    if (conn != null) conn.close();  
}  
return zip_code;  
}
```



```
public static ResultSet SelectZipcodeList(String city_name) throws Exception {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rs = null;
    String name = null;

    try {
        Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
        conn = DriverManager.getConnection("jdbc:default:connection:");

        String sql = "select zip_code from zip_code";
        sql += " where city_name like '%" + city_name + "%'";

        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql);

        // 결과 을 넘겨주기 위해 반환전에 CUBRIDResultSet 클래스로 캐스팅 하고 setReturnable() 메소드를 호
        // 출한 후 반환한다.
        ((CUBRIDResultSet)rs).setReturnable();
    }
}
```

# 자바 저장프로시저 작성 (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

```
} catch ( SQLException e ) {  
    System.err.println(e.getMessage());  
} catch ( Exception e ) {  
    System.err.println(e.getMessage());  
} finally {  
    if (rs != null) rs.close();  
    if (stmt != null) stmt.close();  
    if (conn != null) conn.close();  
}  
  
return rs;  
}  
  
}
```





- 외부 connection 사용
  - connection 명시
  - 별도의 트랜잭션이므로 트랜잭션 처리 필수

```
import cubrid.jdbc.driver.*;  
...  
Class.forName("cubrid.jdbc.driver.CUBRIDDriver");  
Connection con = DriverManager.getConnection("jdbc:CUBRID:localhost:33000:subway::", "public","")  
;  
...  
con.commit();
```



- JAVA 저장프로시저 사용 환경 설정
  - java\_stored\_procedure=yes (cubrid.conf) 설정후 데이터베이스 재구동 필요
- JAVA class 등록
  - compile된 JAVA class 를 데이터베이스에 등록
    - 데이터베이스디렉토리/java
  - 운영중 등록/변경 가능

```
loadjava demodb MySP.class
```

- stored procedure 등록
  - procedure 등록 : 결과값이 없는 경우

```
create procedure insert_zipcode(zip_code varchar, city_name varchar)
as language java
name 'MySP.InsertZipcode(java.lang.String, java.lang.String)'
```

- function 등록 : 결과값이 있는 경우

```
create function select_zipcode(city_name varchar) return varchar
as language java
name 'MySP.SelectZipcode(java.lang.String) return java.lang.String'

create function select_zipcode_list(city_name varchar) return cursor
as language java
name 'MySP.SelectZipcodeList(java.lang.String) return java.sql.ResultSet'
```



- 등록된 저장프로시저 확인
  - 시스템 카다로그를 통한 확인

```
select * from db_stored_procedure
```

- 저장프로시저 삭제
  - 이름 변경이나 속성 변경은 현재 지원되지 않음

```
drop procedure insert_bus  
drop function select_busnum
```



## ● 에러 로그

- 수행중 발생한 exception은 \$CUBRID/log 아래 기록됨
  - 예) demodb\_java.log
  - 화면 출력을 원하는 경우 CUBRID database server 를 command 형태로 구동시키면 현재 창에 에러 메시지 표시

## ● procedure 실행

- 돌려주는 결과가 없으므로, 주로 call 명령을 통한 실행

```
call insert_bus(1, 1, '1번출구', '100-1')
```

## ● function 실행

- 돌려주는 결과의 사용을 위해, 주로 SQL 문을 통한 실행

```
select select_busnum(bus_id) from bus
```

- 결과셋을 돌려주는 경우 JAVA 응용 사용

```
// result set 을 반환하는 stored procedure 를 호출하기 위해서는 CallableStatement 를 사용해야 한다.  
CallableStatement cstmt = conn.prepareCall("?=CALL select_gate(" + bus_num + ")");  
// esultSet을 반환 받기 위해 registerOutParameter의 OUT 인자를 Types.JAVA_OBJECT로 설정한다.  
cstmt.registerOutParameter(1, Types.JAVA_OBJECT);  
cstmt.execute();  
// getObject()로 가져온 후 ResultSet으로 캐스팅 한다.  
ResultSet rs = (ResultSet)cstmt.getObject(1);
```



## – JAVA 응용 예

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

public class CallTest {
    public static void main(String[] args) throws Exception {
        Connection conn = null;
        CallableStatement cstmt= null;
        ResultSet rs = null;

        try{
            Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
            // subway 테이블에 접근하기 위하여 public 로 로그인한다. 암호는 설정하지 않았으므로 없다.
            conn = DriverManager.getConnection("jdbc:CUBRID:localhost:33000:subway::", "public","");

            String gate = "";

            // result set 을 반환하는 stored procedure 를 호출하기 위해서는 CallableStatement 를 사용해야 한다.

            cstmt = conn.prepareCall("?=CALL select_gate('~' + bus_num + '~");
            // esultSet을 반환 받기 위해 registerOutParameter의 OUT 인자를 Types.JAVA_OBJECT로 설정한다.
            cstmt.registerOutParameter(1, Types.JAVA_OBJECT);
            cstmt.execute();
```

# 자바 저장프로시저 실행 (계속)



**CUBRID**  
Driving the DBMS REVOLUTION

```
// getObject()로 가져온 후 ResultSet으로 캐스팅 한다.
rs = (ResultSet)cstmt.getObject(1);

while(rs.next()) {
    // stored procedure 내에서 사용된 select 절의 이름을 사용한다.
    gate = rs.getString("gate");
    System.out.println("Gate : " + gate);
}

} catch (Exception e) {
    e.printStackTrace();
    System.err.println("SQLException:"+e.getMessage());
} finally {
    if (rs != null) rs.close();
    if (cstmt != null) cstmt.close();
    if ( conn != null ) conn.close();
}
}
```



- 작성 예를 실제 작성하여 실행해 본다
  - procedure, function, result set, result set 사용 예제 포함



장시간 교육에 수고 하셨습니다. 교육에 대한 여러분의 의견을 부탁드립니다  
[www.cubrid.com/customer\\_survey.php](http://www.cubrid.com/customer_survey.php)

**CUBRID**  
Beyond Software!