

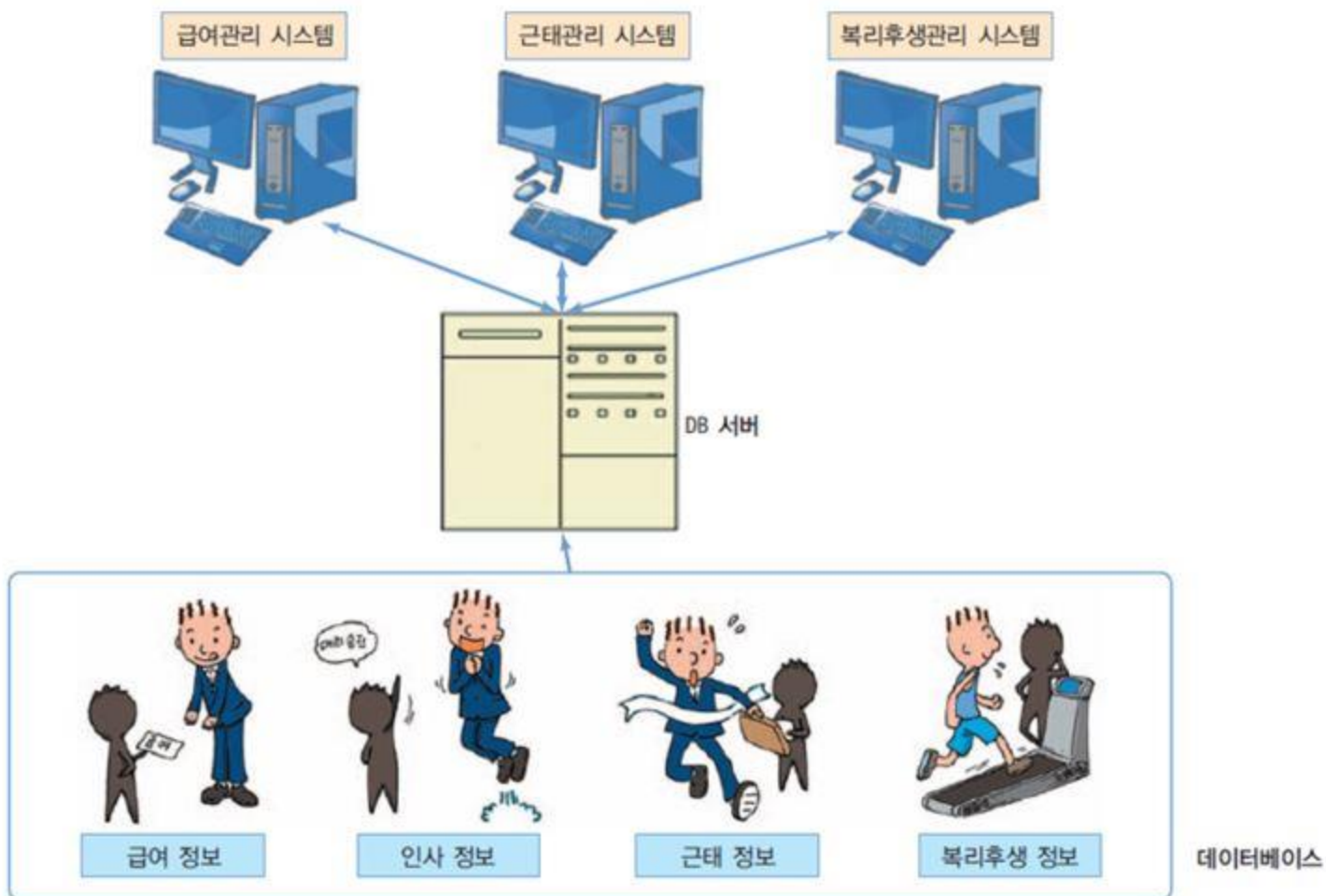
# 데이터베이스의 개념

2

- 데이터베이스
  - ▣ 여러 응용 시스템들의 통합된 정보들을 저장하여 운영할 수 있는 공용 데이터들의 집합
  - ▣ 데이터의 저장, 검색, 갱신을 효율적으로 수행할 수 있도록 데이터를 고도로 조직화하여 저장
- DBMS
  - ▣ 데이터베이스 관리 시스템(DataBase Management System)
    - 오라클(Oracle), 마이크로소프트의 SQL Server, MySQL, IBM의 DB2 등
- 데이터베이스 종류
  - ▣ 관계형 데이터베이스
    - 키(key)와 값(value)들의 관계를 테이블로 표현한 데이터베이스 모델
    - 키는 테이블의 열(column)이 되며 테이블의 행(row)은 하나의 레코드(record)를 표현
    - 현재 사용되는 대부분의 데이터베이스는 관계형 데이터베이스
  - ▣ 객체 지향 데이터베이스
    - 객체 지향 프로그래밍에 쓰이는 것으로, 정보를 객체의 형태로 표현하는 데이터베이스 모델
    - 오브젝트 데이터베이스(object database)라고도 부른다.

# 기업 내의 데이터베이스 사례

3



# 관계형 데이터베이스 구조

4

- 관계형 데이터 베이스
  - 데이터들이 다수의 테이블로 구성
  - 각 행은 하나의 레코드
  - 각 테이블은 키(key)와 값(value) 들의 관계로 표현
  - 여러 테이블 간에 공통된 이름의 열 포함
    - 이런 경우 서로 다른 테이블 간에 관계(relation)가 성립
  - 대부분의 데이터베이스는 관계형 데이터베이스
    - JDBC API

두 테이블이 동일한  
키를 가지고 있음 : 관계

employee_id	name
00001	김철수
00002	최고봉
00003	이기자

테이블 A

payroll_id	amount	employee_id
00000001	1000000	00002
00000002	2000000	00010
00000003	3000000	00021

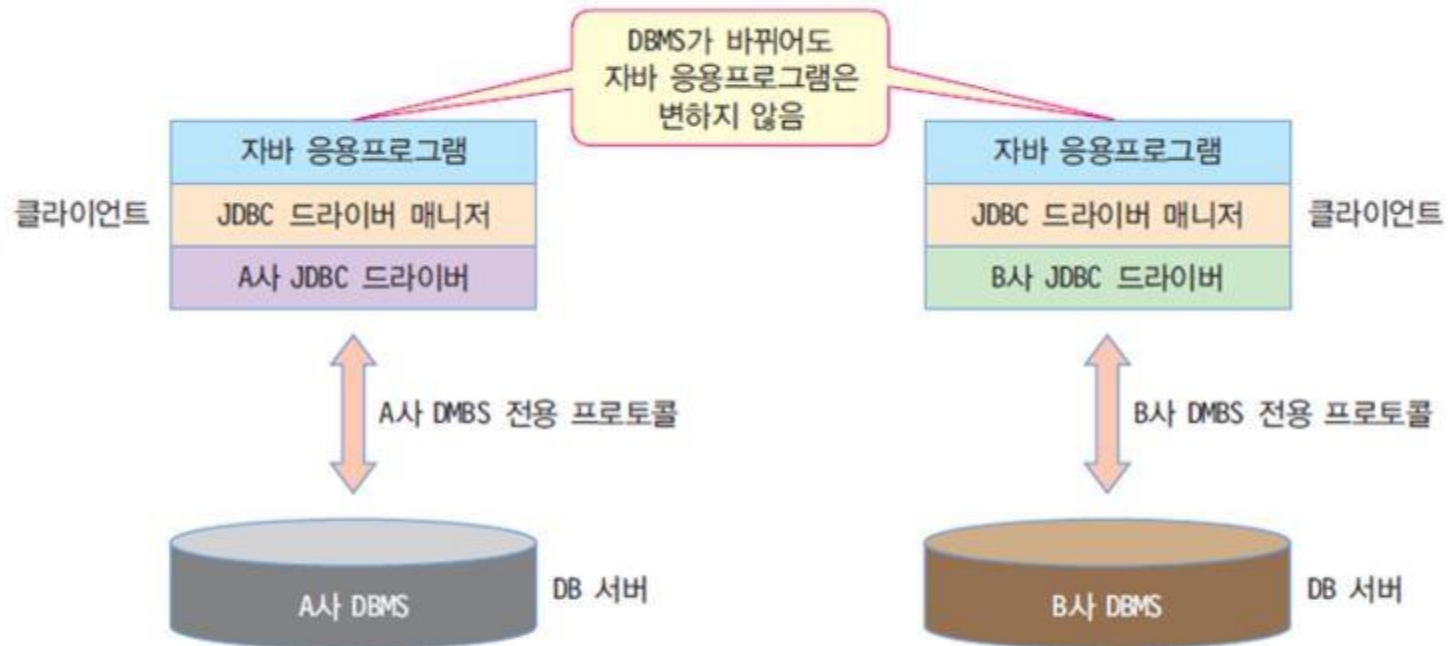
테이블 B

# SQL과 JDBC

- SQL(Structured Query Language)
  - ▣ 관계형 데이터베이스 관리 시스템에서 사용
  - ▣ 데이터베이스 스키마 생성, 자료의 검색, 관리, 수정, 그리고 데이터베이스 객체 접근 관리를 위해 고안된 언어
  - ▣ 데이터베이스로부터 정보를 추출하거나 갱신하기 위한 표준 대화식 프로그래밍 언어
    - 다수의 데이터베이스 관련 프로그램들이 SQL을 표준으로 채택
- JDBC (Java DataBase Connectivity)
  - ▣ 관계형 데이터베이스에 저장된 데이터를 접근 및 조작할 수 있게 하는 API
  - ▣ 다양한 DBMS에 대해 일관된 API로 데이터베이스 연결, 검색, 수정, 관리 등을 할 수 있게 함

# JDBC 구조

7

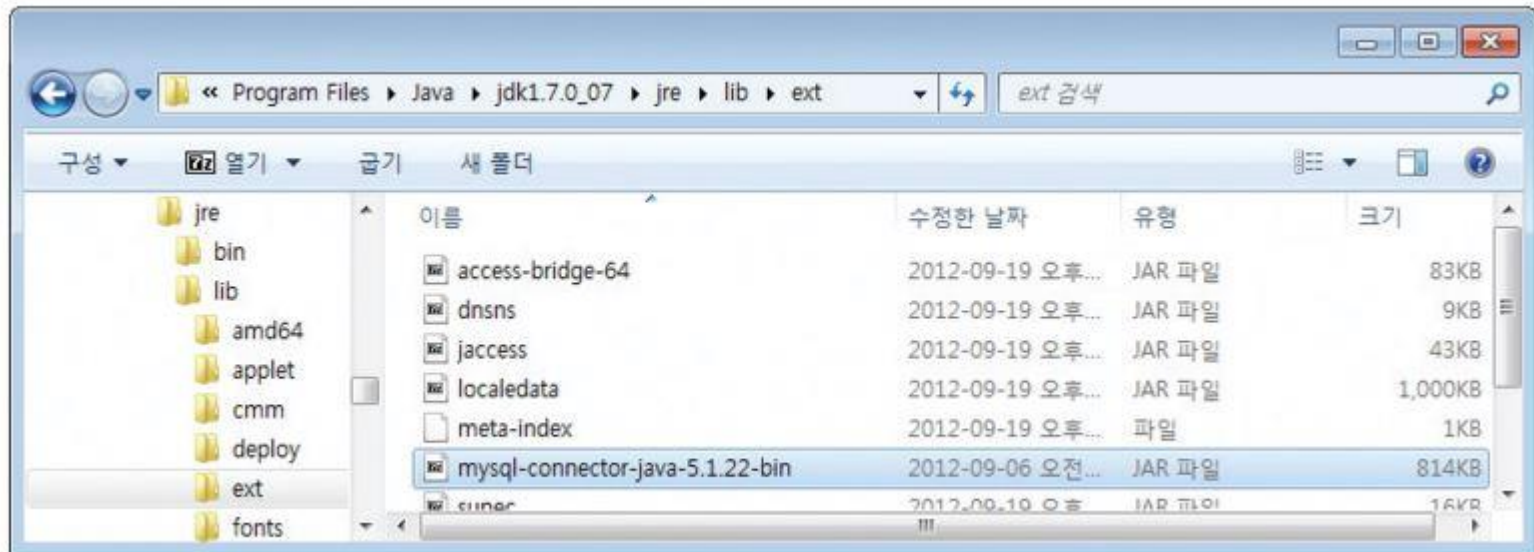


- JDBC 드라이버 매니저
  - 자바 API에서 지원하며 DBMS를 접근할 수 있는 JDBC 드라이버 로드
- JDBC 드라이버
  - DBMS마다 고유한 JDBC 드라이버 제공, JDBC 드라이버와 DBMS는 전용 프로토콜로 데이터베이스 처리
- DBMS
  - 데이터베이스 관리 시스템, 데이터베이스 생성·삭제, 데이터 생성·검색·삭제 등 전담 소프트웨어 시스템



## MySQL 설치

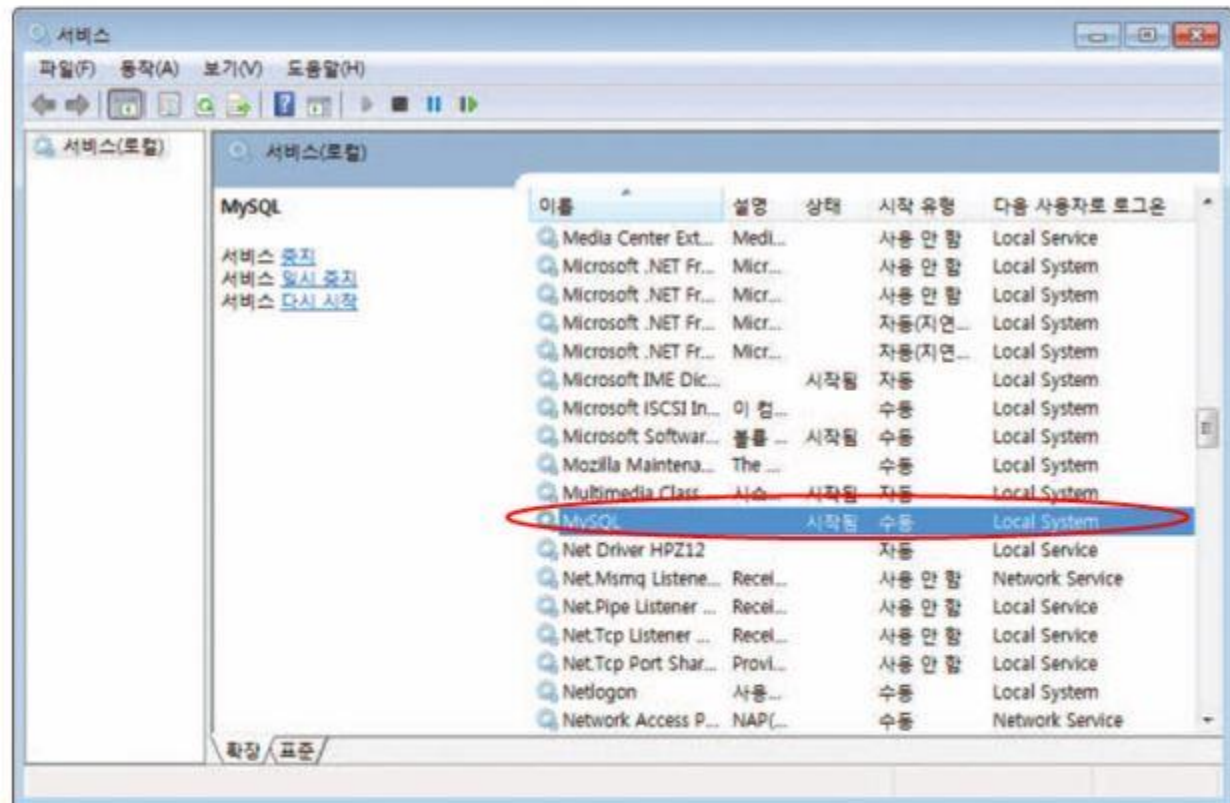
<http://www.mysql.com/downloads/> 에서 서버프로그램을 다운받는다.



# MySQL 서버 실행

17

- 서비스 관리자 이용
  - 윈도우 서비스로 동작하게 설정하였으므로 제어판 시스템 및 보안 관리 도구에서 서비스 관리자를 실행하여 실행 상태 확인
  - 서비스 관리자에서 수동으로 실행 시작 및 중지

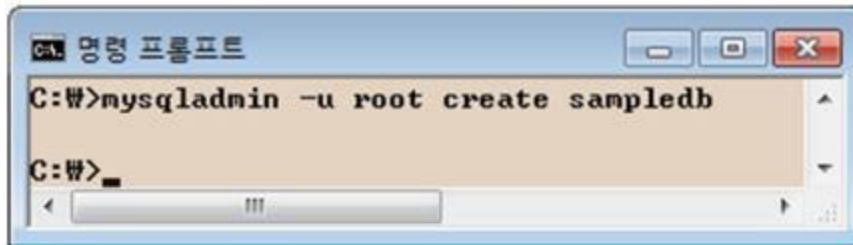


# 실습용 데이터베이스 생성하기

18

## □ 데이터베이스 생성

- ▣ MySQL 설치 디렉터리의 bin\mysqladmin.exe 프로그램 이용



- -u 옵션은 root 계정으로 명령 수행을 의미
- create는 DB 생성 명령
- sampledb는 생성할 DB 이름

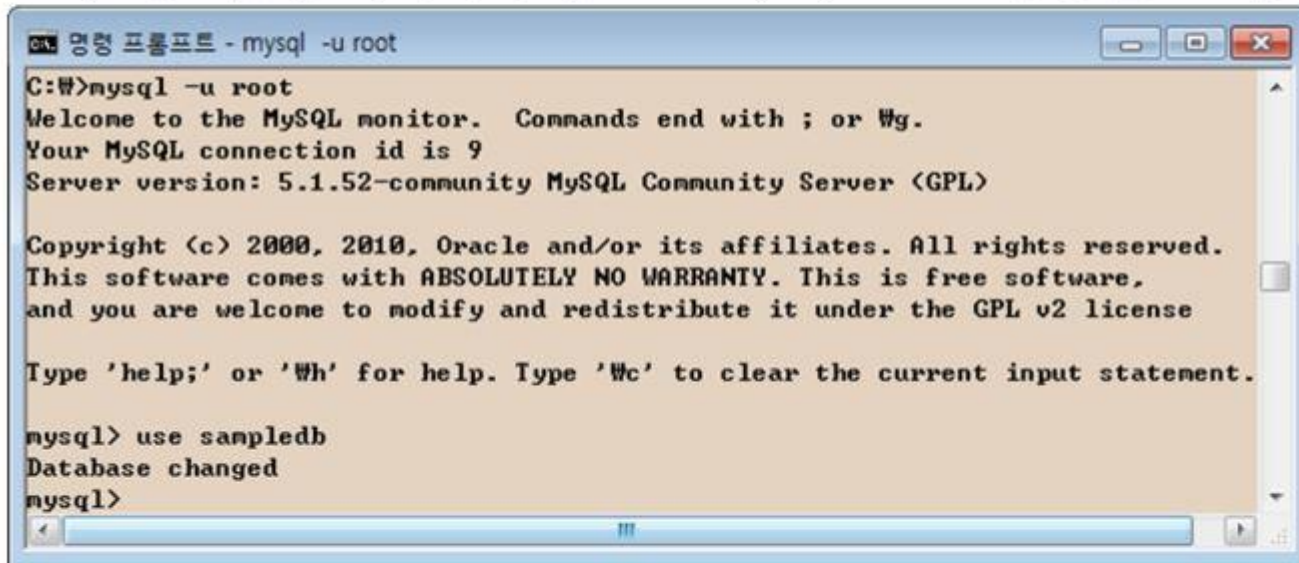
**이번 실습에서는 javadb로 만든다.**

> create database javadb;



## □ DB 사용

- MySQL 설치 디렉터리의 bin\mysql.exe로 생성된 DB에 접속



```
명령 프롬프트 - mysql -u root
C:\>mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.1.52-community MySQL Community Server (GPL)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use sampledb
Database changed
mysql>
```

- -u 옵션은 root 계정으로 명령 수행을 의미
- use는 DB 사용 명령
- sampledb는 사용할 DB 이름

> use javadb;

# 테이블 생성

20

- 데이터 저장을 위해 테이블을 생성
- 다음과 같은 구조의 student 테이블 생성

id	name	dept
char(7)	varchar(10)	varchar(20)

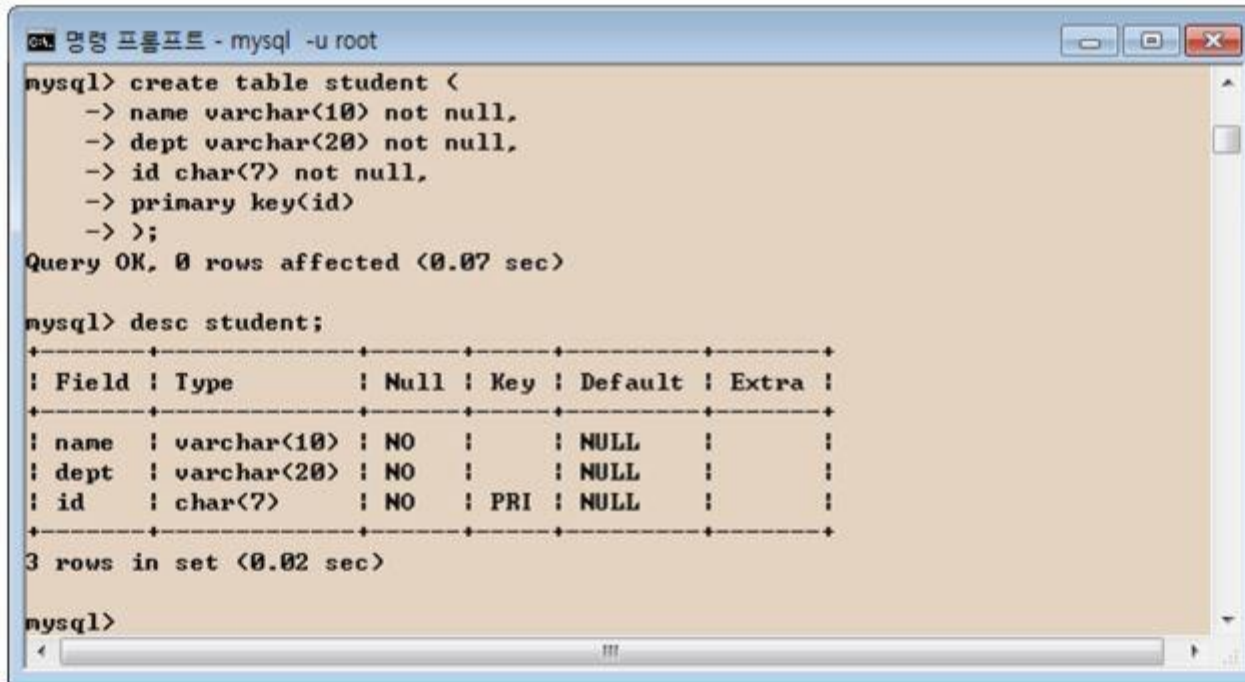
- ▣ name은 varchar 타입으로 10자
  - ▣ dept는 varchar 타입으로 20자
  - ▣ id은 char 타입으로 7자
- 저장할 데이터

id	name	dept
1091011	김철수	컴퓨터시스템
0792012	최고봉	멀티미디어
0494013	이기자	컴퓨터공학

# 테이블 생성

21

- create 문으로 테이블 생성



```
mysql> create table student (<br>  -> name varchar(10) not null,<br>  -> dept varchar(20) not null,<br>  -> id char(7) not null,<br>  -> primary key(id)<br>  -> );<br>Query OK, 0 rows affected (0.07 sec)<br><br>mysql> desc student;<br>+-----+<br>| Field | Type          | Null | Key | Default | Extra |<br>+-----+<br>| name  | varchar(10)   | NO   |     | NULL    |      |<br>| dept  | varchar(20)   | NO   |     | NULL    |      |<br>| id    | char(7)       | NO   | PRI | NULL    |      |<br>+-----+<br>3 rows in set (0.02 sec)<br><br>mysql>
```

- create table 다음에 테이블 이름 지정
- 열 이름 데이터 타입(데이터 크기)을 콤마로 분리하여 나열
- not null은 행의 데이터에 null값이 올 수 없음을 의미
- primary key는 키로 사용될 행 지정
- desc 명령은 생성된 테이블의 구조 표시

# 데이터 추가

22

## □ insert 문으로 테이블의 데이터 추가



```
명령 프롬프트 - mysql -u root
mysql> insert into student (name,dept,id) values('김철수','컴퓨터시스템','1091011');
Query OK, 1 row affected (0.03 sec)

mysql> insert into student (name,dept,id) values('최고봉','멀티미디어','0792012');
Query OK, 1 row affected (0.02 sec)

mysql> insert into student (name,dept,id) values('이기자','컴퓨터공학','0494013');
Query OK, 1 row affected (0.05 sec)

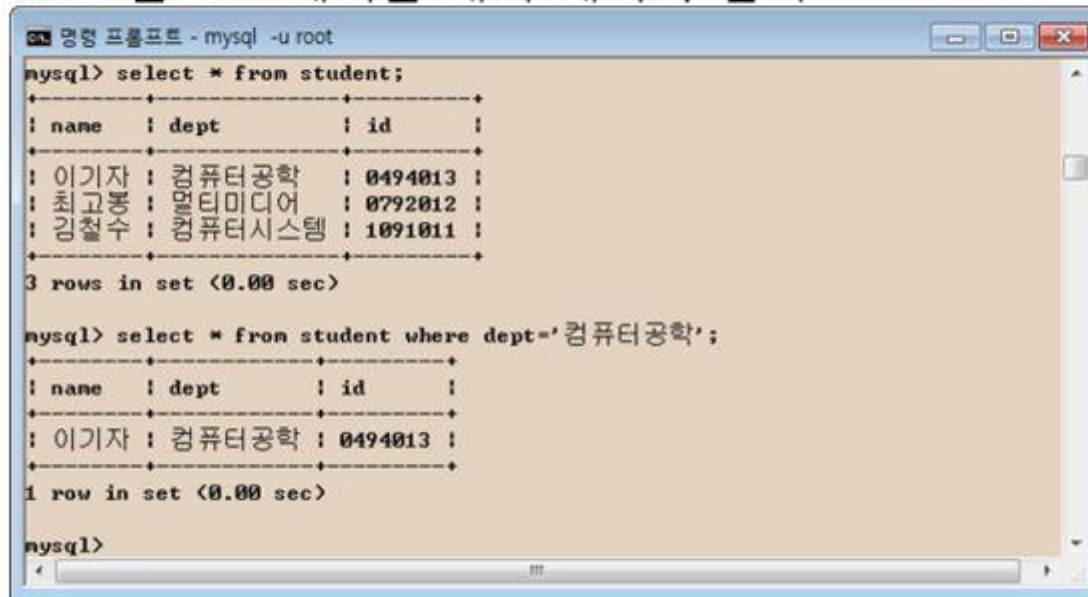
mysql>
```

- insert into 다음에 테이블 이름 지정
- 테이블 이름 다음 괄호 안에 열 이름을 콤마로 구분하여 나열
- values 다음 괄호 안에 열의 값들을 콤마로 구분하여 나열
- 문자 타입의 데이터는 단일 인용 부호로 묶어서 표시함에 유의

# 데이터 검색

23

## □ select문으로 테이블 내의 데이터 검색



```
명령 프롬프트 - mysql -u root
mysql> select * from student;
+----+-----+-----+
| name | dept   | id    |
+----+-----+-----+
| 이기자 | 컴퓨터공학 | 0494013 |
| 최고봉 | 멀티미디어 | 0792012 |
| 김철수 | 컴퓨터시스템 | 1091011 |
+----+-----+-----+
3 rows in set (0.00 sec)

mysql> select * from student where dept='컴퓨터공학';
+----+-----+-----+
| name | dept   | id    |
+----+-----+-----+
| 이기자 | 컴퓨터공학 | 0494013 |
+----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

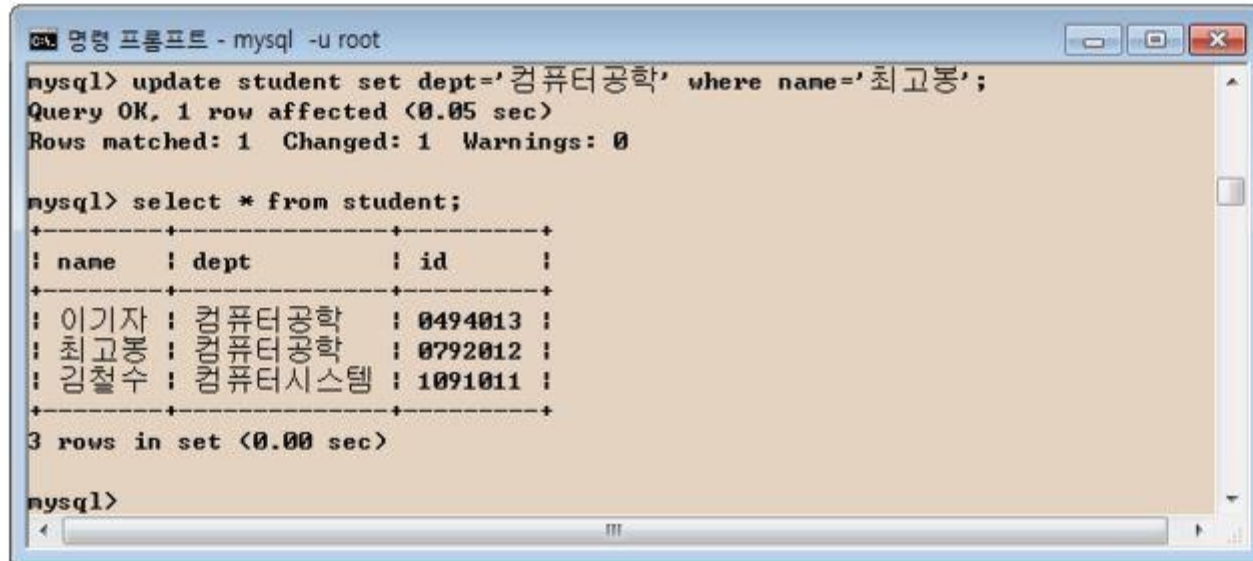
- select 다음에는 데이터를 추출할 열 이름을 콤마로 분리하여 나열
- 모든 열에 대해 데이터를 추출할 때는 \*를 열 이름 대신 사용
- from 다음에 테이블 이름을 지정
- where 다음에 검색 조건 지정. 위의 예에서 dept 값이 '컴퓨터공학'인 레코드 검색
- where는 생략 가능



# 데이터 수정

24

- update문을 이용하여 데이터 수정



```
mysql> update student set dept='컴퓨터공학' where name='최고봉';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from student;
+-----+-----+-----+
| name  | dept      | id      |
+-----+-----+-----+
| 이기자 | 컴퓨터공학 | 0494013 |
| 최고봉 | 컴퓨터공학 | 0792012 |
| 김철수 | 컴퓨터시스템 | 1091011 |
+-----+-----+-----+
3 rows in set (0.00 sec)

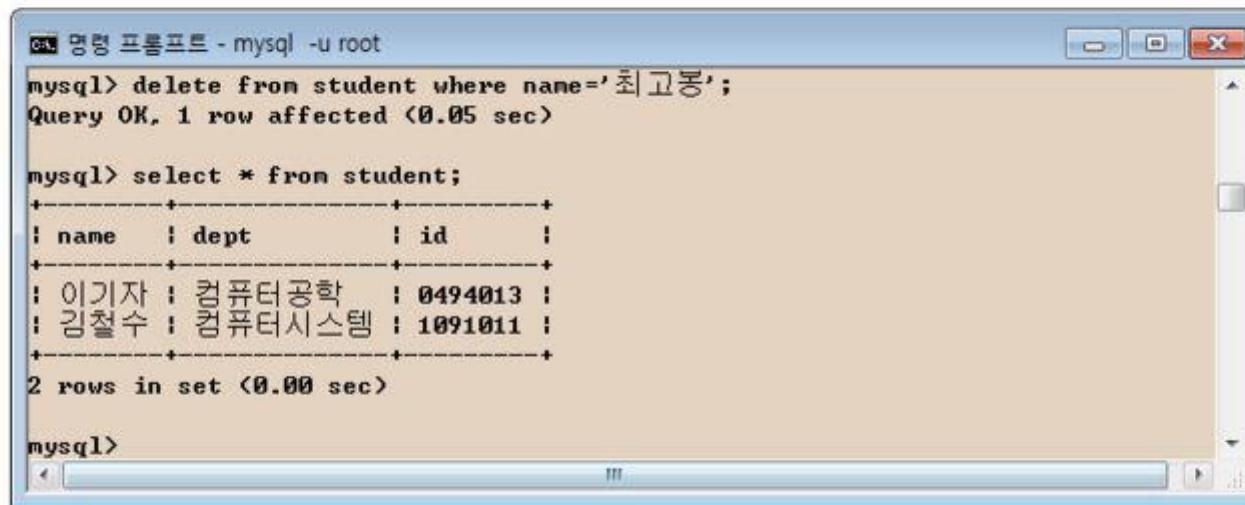
mysql>
```

- update 다음에는 테이블 이름 지정
- set 다음에 수정할 열의 이름과 값을 콤마로 분리하여 나열
- where 다음에는 검색 조건을 지정. 위의 예에서는 name 값이 '최고봉'인 레코드의 데이터 수정
- where는 생략 가능

# 레코드 삭제

25

- delete문을 이용하여 데이터 삭제



```
mysql> delete from student where name='최고봉';
Query OK, 1 row affected (0.05 sec)

mysql> select * from student;
+-----+-----+-----+
| name  | dept      | id      |
+-----+-----+-----+
| 이기자 | 컴퓨터공학 | 0494013 |
| 김철수 | 컴퓨터시스템 | 1091011 |
+-----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

- delete from 다음에는 테이블 이름 지정
- where 다음에는 검색 조건을 지정. 위의 예에서는 name 값이 '최고봉'인 레코드 삭제
- where는 생략 가능

# JDBC 프로그래밍

26

## □ DB 연결 설정

### ▣ JDBC 드라이버 로드

```
try {  
    Class.forName("com.mysql.jdbc.Driver");  
} catch (ClassNotFoundException e) {  
    e.printStackTrace();  
}
```

- Class.forName()은 동적으로 자바 클래스 로딩
- MySQL의 JDBC 드라이버 클래스인 *com.mysql.jdbc.Driver* 로드
- 드라이버의 클래스 이름은 DB의 드라이버마다 다를 수 있으므로 JDBC 드라이버 문서 참조할 것
- 자동으로 드라이버 인스턴스를 생성하여 DriverManager에 등록
- 해당 드라이버가 없으면 ClassNotFoundException 발생

# 자바 응용프로그램과 JDBC의 연결

27

## ■ 연결

```
try {  
    Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/sampledb", "root", "");  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

"javadb" "1234"

- DriverManager는 자바 어플리케이션을 JDBC 드라이버에 연결시켜주는 클래스로서 getConnection() 메소드로 DB에 연결하여 Connection 객체 반환
- getConnection에서 jdbc: 이후에 지정되는 URL의 형식은 DB에 따라 다르므로 JDBC 문서를 참조
  - MySQL 서버가 같은 컴퓨터에서 동작하므로 서버 주소를 localhost로 지정
  - MySQL의 경우 디폴트로 3306 포트를 사용
  - sampledb는 앞서 생성한 DB의 이름
- "root" 는 DB에 로그인할 계정 이름이며, "" 는 계정 패스워드

## 예제 17-1 : sampledb의 데이터베이스 연결하는 JDBC 프로그램 작성

28

JDBC를 이용하여 sampledb 데이터베이스에 연결하는 자바 응용프로그램을 작성하라.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class JDBC_Ex1 {
    public static void main (String[] args) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/sampled", "root", "1234");
            System.out.println("DB 연결 완료");
        } catch (ClassNotFoundException e) {
            System.out.println("JDBC 드라이버 로드 에러");
        } catch (SQLException e) {
            System.out.println("DB 연결 오류");
        }
    }
}
```

javadb"      "1234"

DB 연결 오류

또는

DB 연결 완료



# 데이터베이스 사용

29

## □ Statement 클래스

- SQL문을 실행하기 위해서는 Statement 클래스를 이용
- 주요 메소드

메소드	설명
ResultSet executeQuery(String sql)	주어진 SQL문을 실행하고 결과는 ResultSet 객체에 반환
int executeUpdate(String sql)	INSERT, UPDATE, 또는 DELETE과 같은 SQL문을 실행하고, SQL문 실행으로 영향을 받은 행의 개수나 0을 반환
void close()	Statement 객체의 데이터베이스와 JDBC 리소스를 즉시 반환

- 데이터 검색을 위해 executeQuery() 메소드 사용
- 추가, 수정, 삭제와 같은 데이터 변경은 executeUpdate() 메소드 사용

# 데이터베이스 사용

30

## ResultSet 클래스

- SQL문 실행 결과를 얻어오기 위해서는 ResultSet 클래스를 이용
- 현재 데이터의 행(레코드 위치)을 가리키는 커서(cursor)를 관리
- 커서의 초기 값은 첫 번째 행 이전을 가리킴
- 주요 메소드

메소드	설명
boolean first()	커서를 첫 번째 행으로 이동
boolean last()	커서를 마지막 행으로 이동
boolean next()	커서를 다음 행으로 이동
boolean previous()	커서를 이전 행으로 이동
boolean absolute(int row)	커서를 지정된 행으로 이동
boolean isFirst()	첫 번째 행이면 true 반환
boolean isLast()	마지막 행이면 true 반환
void close()	ResultSet 객체의 데이터베이스와 JDBC 리소스를 즉시 반환
Xxx getXxx(String columnName)	Xxx는 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 이름에 해당하는 데이터를 반환. 예를 들어, int형 데이터를 읽는 메소드는 getInt()이다.
Xxx getXxx(int columnIndex)	Xxx는 해당 데이터 타입을 나타내며 현재 행에서 지정된 열 인덱스에 해당하는 데이터를 반환. 예를 들어, int형 데이터를 읽는 메소드는 getInt()이다.

# 데이터베이스 사용

31

## □ 테이블의 모든 데이터 검색

```
Statement stmt = conn.createStatement();  
ResultSet rs = stmt.executeQuery("select * from student");
```

- Statement의 executeQuery()는 SQL문의 실행하여 실행 결과를 넘겨줌
- 위의 SQL문의 student 테이블에서 모든 행의 모든 열을 읽어 결과를 rs에 저장

## □ 특정 열만 검색

```
ResultSet rs = stmt.executeQuery("select name, id from student");
```

- 특정 열만 읽을 경우는 select문을 이용하여 특정 열의 이름 지정

## □ 조건 검색

```
rs = stmt.executeQuery("select name, id, dept from student where id='0494013'");
```

- select문에서 where절을 이용하여 조건에 맞는 데이터 검색

# 데이터베이스 사용

32

## □ 검색된 데이터의 사용

```
while (rs.next()) {  
    System.out.println(rs.getString("name"));  
    System.out.println(rs.getString("id"));  
    System.out.println(rs.getString("dept"));  
}  
rs.close();
```

### □ Statement객체의 executeQuery() 메소드

- ResultSet 객체 반환

### □ ResultSet 인터페이스

- DB에서 읽어온 데이터를 추출 및 조작할 수 있는 방법 제공

### □ next() 메소드

- 다음 행으로 이동

김철수	컴퓨터시스템	1091011
최고봉	멀티미디어	0792012
이기자	컴퓨터공학	0494013

next() : true  
next() : true  
next() : true  
next() : true  
next() : false

# 데이터베이스 사용

33

- ResultSet의 getXXX() 메소드
  - 해당 데이터 타입으로 열 값을 읽어옴
  - 인자로 열의 이름이나 인덱스를 줄 수 있음
  - DB의 데이터 타입에 해당하는 자바 데이터 타입으로 데이터를 읽어야 함.
  - 모든 데이터 타입에 대해 getString() 메소드로 읽을 수 있으나 사용할 때는 해당 데이터 타입으로 변환해서 사용
- ResultSet에서 모든 데이터를 다 읽어들이고 후에는 close()를 호출하여 자원 해제



# 예제: 데이터 검색과 출력

```
public class Jdbc_ex2 {
```

```
    public static void main(String[] args) {  
        try {
```

```
            Class.forName("com.mysql.jdbc.Driver");
```

```
            System.out.println("드라이버 로딩");
```

```
            Connection connect =
```

```
DriverManager.getConnection("jdbc:mysql://localhost:3306/javadb","root","1234");
```

```
            System.out.println("MySql의 javadb와 연결");
```

```
Statement statement = connect.createStatement();
```

```
ResultSet rs = statement.executeQuery("select * from student");
```

```
while(rs.next()){
```

```
    System.out.print(rs.getString(1)+"  ");
```

```
        //rs.getString("name")
```

```
    System.out.print(rs.getString(2)+"  ");
```

```
    System.out.println(rs.getString(3));
```

```
}
```

```
}catch(ClassNotFoundException e) {  
    System.out.println(e.toString());  
}catch(SQLException e1){  
    System.out.println(e1.toString());  
}  
  
}  
  
}
```

# 데이터의 변경

37

## □ 레코드 추가

```
stmt.executeUpdate("insert into student"+  
" (name,id,dept) values('윤미영','0000015','자바')");
```

- DB에 변경을 가하는 조작은 executeUpdate() 메소드 사용
- SQL문 수행으로 영향을 받은 행의 개수 반환

## □ 데이터 수정

```
stmt.executeUpdate("update student set id='0189011' where name='"+  
new String("아무개".getBytes(), "ISO-8859-1") +"'");
```

- where문의 MySQL에서 처리되므로 문자열을 Unicode에서 ISO-8859-1로 변환에 주의

## □ 데이터 삭제

```
stmt.executeUpdate("delete from student "+  
"where id='0000015'");
```

```
Statement statement = connect.createStatement();
```

```
int resultNum =statement.executeUpdate("insert into student"+  
" (name,id,dept) values('윤미영','0000015','자바')");
```

```
//int resultNum =statement.executeUpdate("delete from student "+  
// "where id='0000015'");
```

```
ResultSet rs = statement.executeQuery("select * from student");
```

```
while(rs.next()){  
    System.out.print(rs.getString(1)+" ");  
    System.out.print(rs.getString(2)+" ");  
    System.out.println(rs.getString(3));  
  
}
```

## □ PreparedStatement 인터페이스 기능

- SQL 문 안에 매개변수들을 쉽게 삽입(insert) 가능
- 새로운 매개변수들과 함께 쉽게 재사용도 가능
- 실행된 구문의 성능이 향상 가능
- Batch 업데이트가 더욱 쉽게 가능

SQL String을 분석하고, 그것을 위해 쿼리 플랜을 짜는 것은 시간이 걸립니다  
(\* 쿼리플랜 : DB를 가장 효율적인 쿼리 실행 방법에 대한 분석)

만약 모든 쿼리에 대한 SQL 문이 새로 작성되거나, 데이터베이스에 갱신되면 데이터베이스는 SQL 문을 분석해야 하고, 쿼리를 위해(for queries) 쿼리플랜을 생성해야 합니다. 이 때, PreparedStatement의 재사용을 통해 사용자는 SQL 분석과 연속 쿼리에 대한 쿼리플랜을 재사용이 가능합니다. 이것은 구문 분석과 쿼리 플랜을 감소시키고, 각 실행에 대한 오버헤드를 줄여 쿼리 실행 속도를 증가시킵니다.



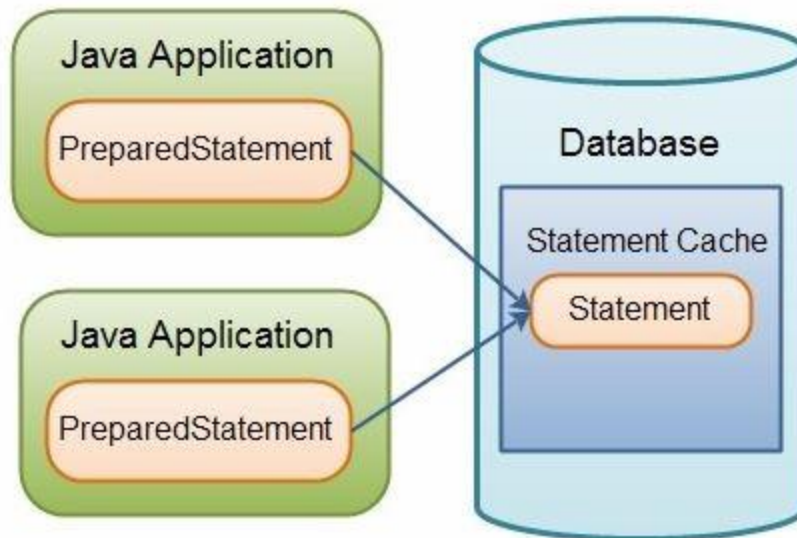
PreparedStatement의 재사용 가능성은 두 가지가 있습니다.

1. JDBC에서 PreparedStatement의 재사용
2. DB에서 PreparedStatement의 재사용

첫 번째는, JDBC 드라이버는 PreparedStatement 객체들을 내부적으로 저장할 수 있어, PreparedStatement를 재사용할 수 있습니다. 이 방법은 PreparedStatement의 실행시간을 조금 줄일 수 있습니다.

두 번째는, 잠재적으로 저장된 구문 분석과 쿼리 플랜이 자바 애플리케이션을 통해 재사용되는 것인데, 예로는 같은 DB를 사용하는 애플리케이션 서버들이(server in a cluster) 있습니다.

DB에서 statement의 저장상태를 표현한 다이어그램을 보면,



The caching of PreparedStatement's in the database.

이 다이어그램은 JDBC 드라이버의 preparedStatement 캐시를 보여주는 게 아니므로, 당신은 그것을 생각해봐야 합니다.

## 1. PreparedStatement의 생성

PreparedStatement를 재 사용하기 위해서는 생성을 먼저 해야겠죠?

Connection.prepareStatement()를 이용해 이처럼 생성합니다!

```
String sql = "select * from student where id =?";
```

```
PreparedStatement preparedStatement = connection.prepareStatement(sql);
```

이제 이 PreparedStatement에 매개변수들을 삽입할 준비가 다 되었습니다.

## 2. PreparedStatement에 매개변수 삽입

위처럼 preparedStatement가 생성되면, 당신은 물음표 표시가 있는 곳에 매개변수를 삽입할 수 있는데, 이는 setType()을 사용해 삽입할 수 있습니다.

```
preparedStatement.setString(1,"0000015");
```

첫 번째 자리의 숫자(1)는 삽입해야 할 값의 첫 번째 인덱스 번호를 나타내는 것이고 두 번째 자리의 숫자 (123)는 SQL 문에 삽입될 값을 의미합니다.

### 3. PreparedStatement 실행하기

PreparedStatement를 실행하는 것은 보통의 statement를 실행하는 것과 비슷한데, 쿼리를 실행하기 위해서는 executeQuery() 또는 executeUpdate 메소드를 호출해야 합니다. executeQuery()의 예로

```
String sql = "select * from student where id =?";
```

```
PreparedStatement preparedStatement = connection.prepareStatement(sql);
```

```
preparedStatement.setString(1,"0000015");
```

```
ResultSet result = preparedStatement.executeQuery();
```

이번에는 executeUpdate()예를 보면,

```
String sql = "insert into student (name,dept,id) values(?,?,?)";  
  
PreparedStatement pre = connection.prepareStatement(sql);  
  
pre.setString(1, "Lee");  
pre.setString(2, "comp");  
pre.setString(3, "0000020");  
  
int rowsAffected = pre.executeUpdate();
```

executeUpdate() 메소드는 DB를 업데이트할 때 사용됩니다. 이 메소드는 업데이트에 영향을 받는 DB 안의 많은 데이터를 int값을 반환합니다.

#### 4. PreparedStatement의 재사용

PreparedStatement이 준비되면, 이것은 실행 후에 재사용이 가능합니다. 당신은 PreparedStatement를 매개변수에 새로운 값들을 세팅한 후 재사용하여 이것을 다시 실행시킬 수 있습니다.

```
String sql = "insert into student (name,dept,id) values(?,?,?)";
```

```
PreparedStatement pre = connection.prepareStatement(sql);
```

```
pre.setString(1, "Lee");
```

```
pre.setString(2, "comp");
```

```
pre.setString(3, "0000020");
```

```
int rowsAffected = pre.executeUpdate();
```

```
pre.setString(1, "park");
```

```
pre.setString(2, "media");
```

```
pre.setString(3, "0000021");
```

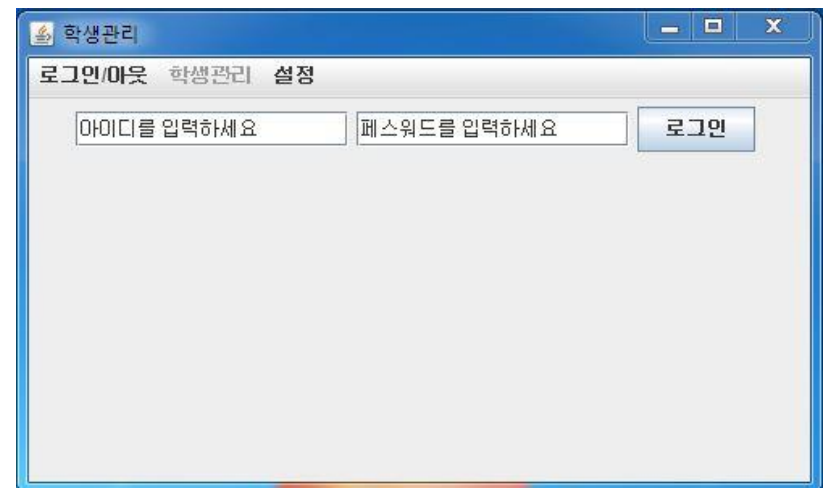
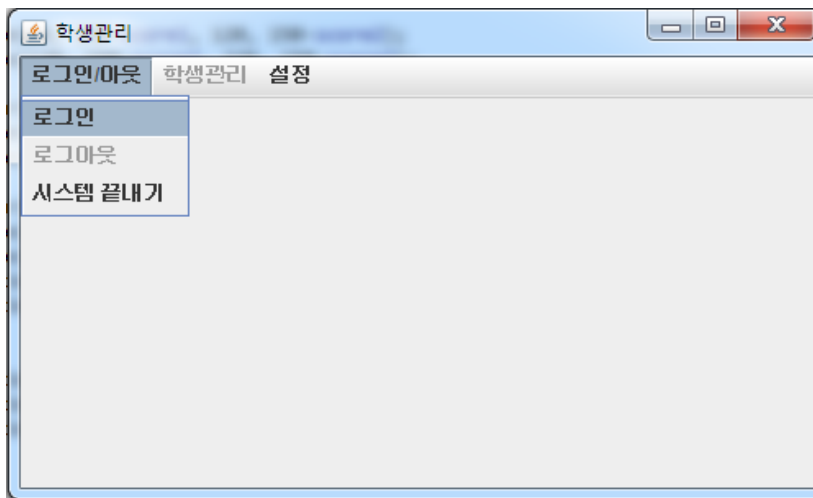
```
int rowsAffected = pre.executeUpdate();
```

executeQuery()메소드를 사용해 쿼리를 실행하기 위한 작업은 ResultSet값을 다시 반환합니다.

## 응용하기

테이블 이름: manager

ID	MANAGER_ID	pwd
int(3)	varchar(50)	varchar(50)





<mysql패스워드 암호화 password('1234')>

- password()함수를 적용하기 위해서 항목의 데이터 타입을 넉넉하게 할당한다.

- pwd 항목을 varchar(100)으로 변경한다.

```
mysql> alter table manager modify column pwd varchar(100) not null;
```

```
mysql> insert into manager (id, manager_id , pwd)
        values(2, 'aaa', password('1234'));
```

...

```
String sql = "select manager_id from manager where pwd= password(?);  
PreparedStatement pre = connect.prepareStatement(sql);
```

```
pre.setString(1, "1234");  
ResultSet rs = pre.executeQuery();
```

```
if(rs.next()){  
    System.out.println(rs.getString(1));  
}
```

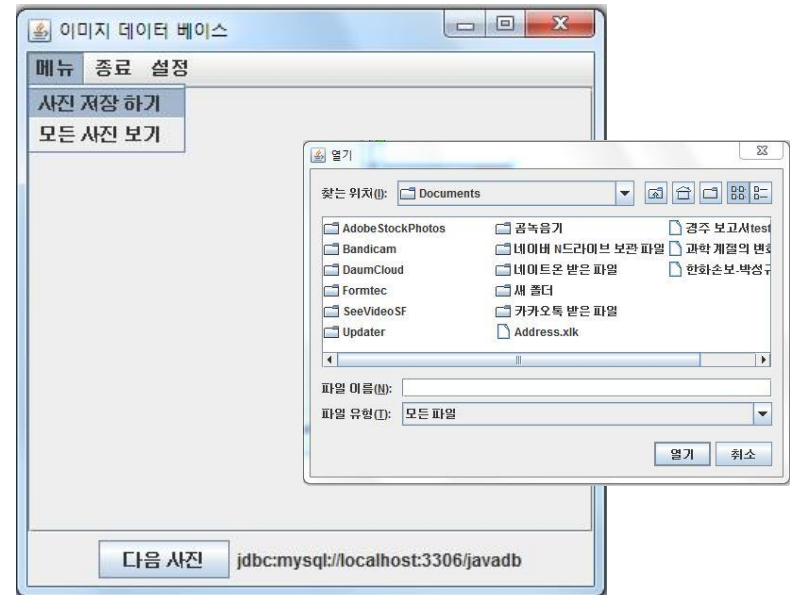
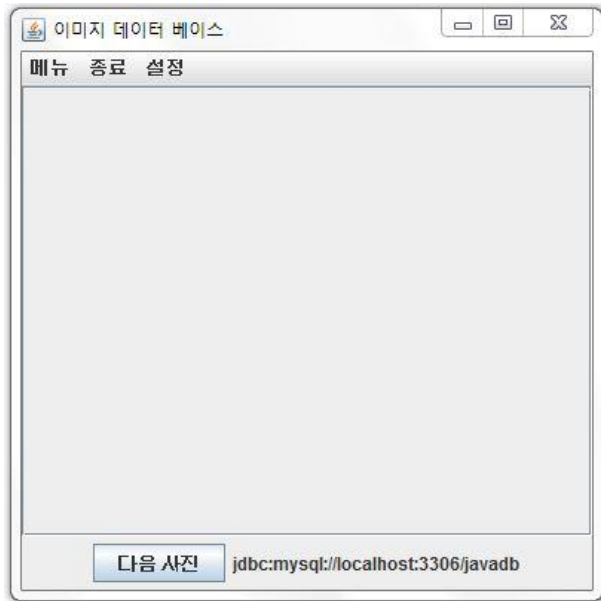
.....

# 데이터베이스로 사진(16M이하) 저장 및 추출하기 (javadb 데이터베이스 이용)

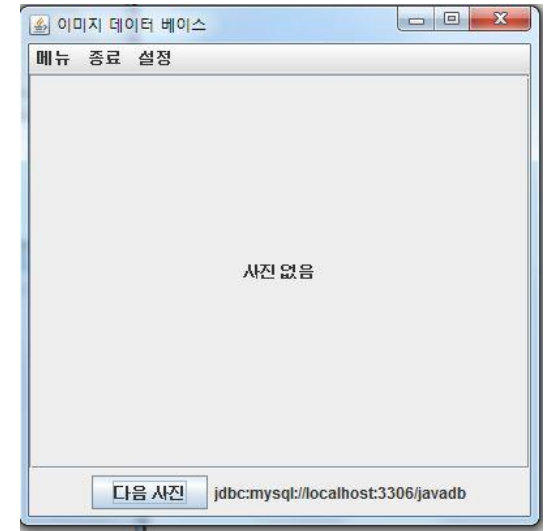
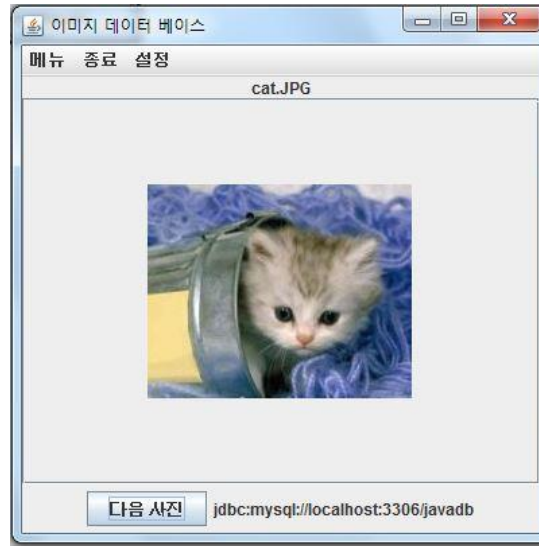
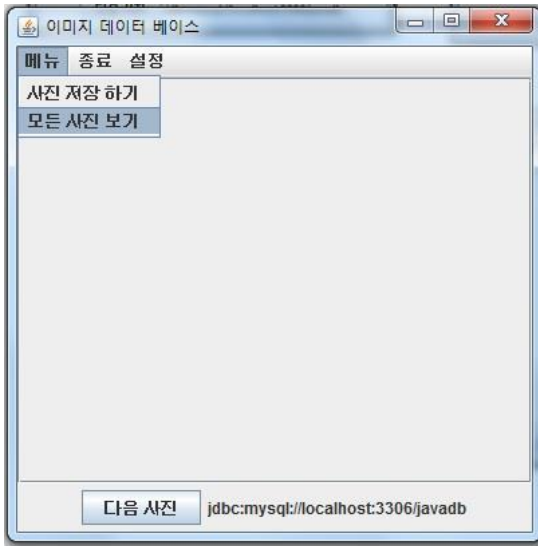
테이블 이름: images

ID	FILENAME	FILE
int(11)	varchar(50)	mediumblob

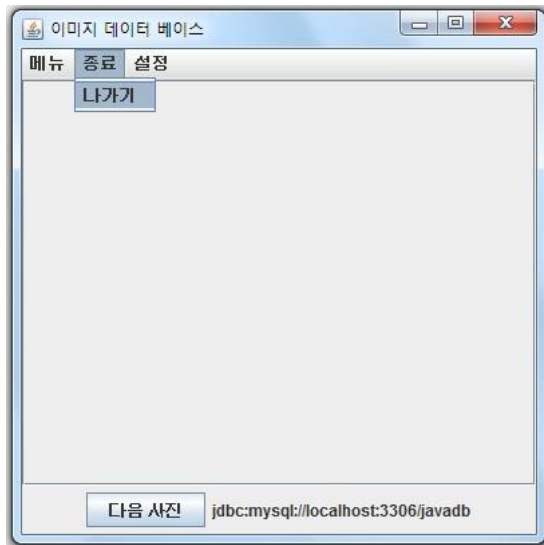
사진저장하기 (save – ActionListener –insertImage(File file))



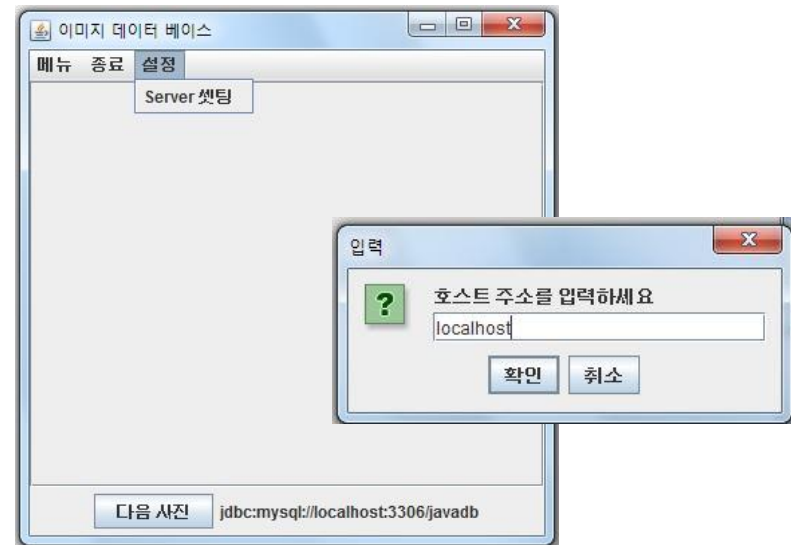
모든사진보기 (view – ActionListener –showphtos() )



나가기 (exit-ActionListener –exit(0))



Server셋팅 (setting – ActionListener –serverSetting() )



```

void insertImage(File file) {
    try{
        FileInputStream fin = new FileInputStream(file); // 파일 입력 스트림 생성
        PreparedStatement pre = conn.prepareStatement("insert into images (ID, FILENAME, FILE) VALUES (?, ?, ?)");
        pre.setInt(1,++numberOfRecord);
        pre.setString(2,file.getName());
        pre.setBinaryStream(3,fin,(int)file.length());
        pre.executeUpdate(); // DB에 사진 저장
        pre.close();
    } catch (Exception e){
        e.getMessage();
    }
}

```

```

void serverSetting() {
    try {

        conn = DriverManager.getConnection("jdbc:mysql://" + addr + ":3306/javadb", "root","1234"); // JDBC 연결
        status.setText("jdbc:mysql://" + addr + ":3306/javadb");
        stmt = conn.createStatement(); // SQL문 처리용 Statement 객체 생성
        ResultSet srs;
        srs = stmt.executeQuery("select count(*) from images"); // 레코드 개수를 얻어오는 쿼리
        srs.next();
        numberOfRecord = srs.getInt(1);

    } catch (SQLException e) {
        e.getMessage();
    }
}

```

```
void showPhotos() {
    try {
        viewRS = stmt.executeQuery("select * from images"); // DB에서 모든 사진을 얻어옴
        if (viewRS.next()) { // 첫번째 사진 표시
            Blob b = viewRS.getBlob("FILE"); // DB에서 바이너리 데이터 얻어옴
            img = new ImageIcon(b.getBytes(1, (int) b.length())); // 바이너리 데이터를 이미지 포맷으로 변환
            imageLabel.setIcon(img);
            imageLabel.setText(null);
            textLabel.setText(viewRS.getString("FILENAME")); // 파일 이름을 textLabel에 출력
        } else { // DB에 사진이 없는 경우
            imageLabel.setIcon(null);
            imageLabel.setText("사진 없음");
            textLabel.setText(null);
        }
    } catch (SQLException e) {
        e.getMessage();
    }
}
```