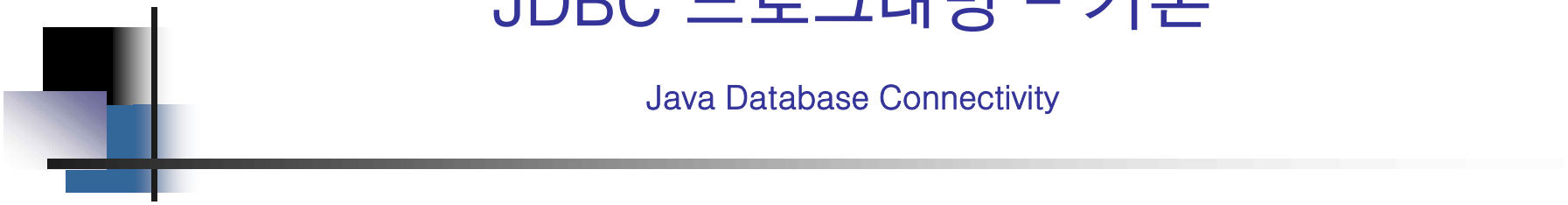


JDBC 프로그래밍 – 기본

Java Database Connectivity

남 광 우

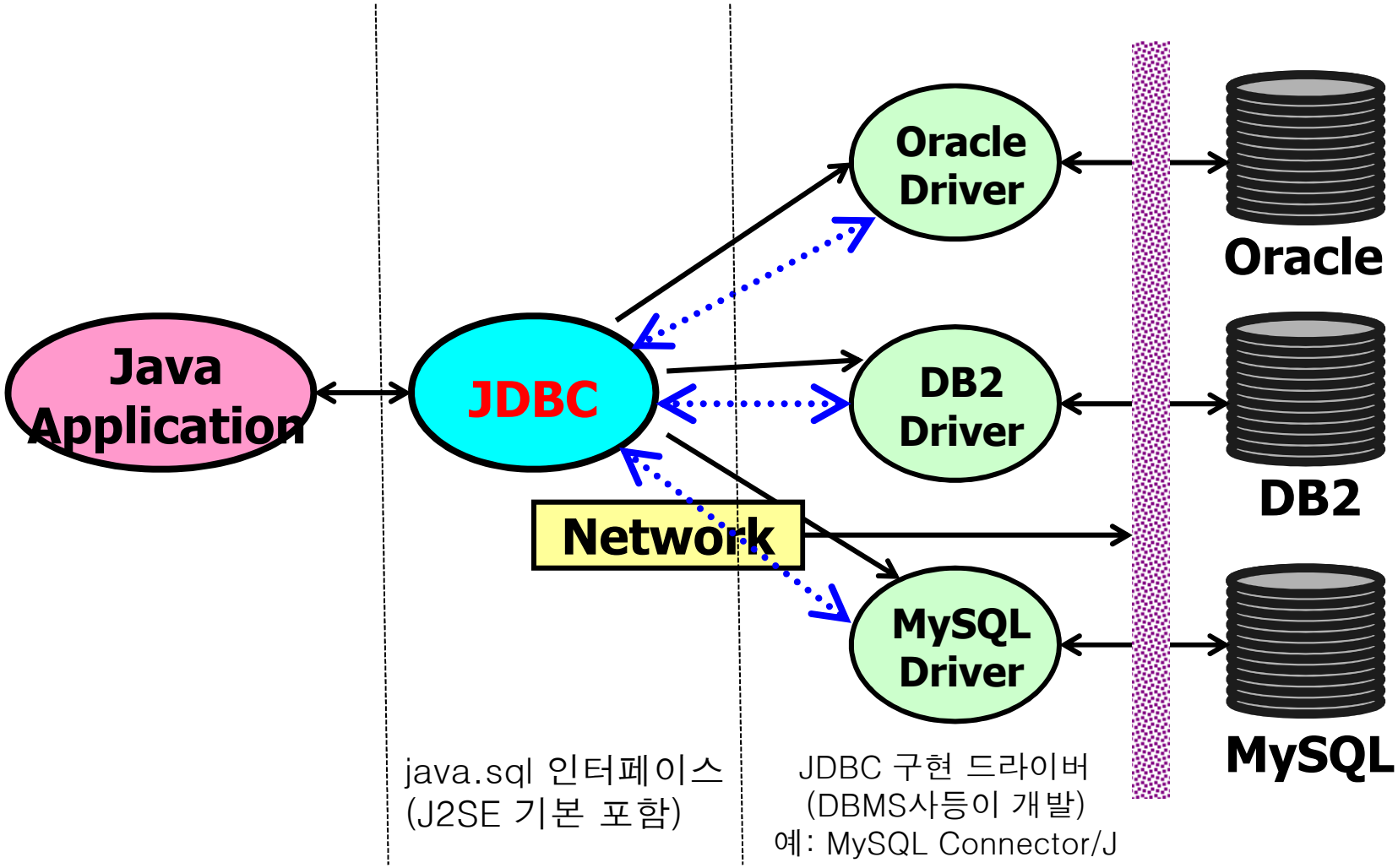
JDBC의 개요

- JDBC = Java DataBase Connectivity
  - Java의 데이터베이스 접근을 위한 프로그래밍 API
  - 1995년에 만들어지고, 1996년 여름 release 됨
  - java.sql.* 패키지로 구성
  - 현재 JDBC 3.0 버전 표준 release

- JDBC의 특징
  - SQL을 기본으로한 DB 접근
  - 응용 프로그램 개발자는 어떤 드라이버를 사용하든지 JDBC 공통 인터페이스에 맞춰서 프로그램을 작성할 수 있음

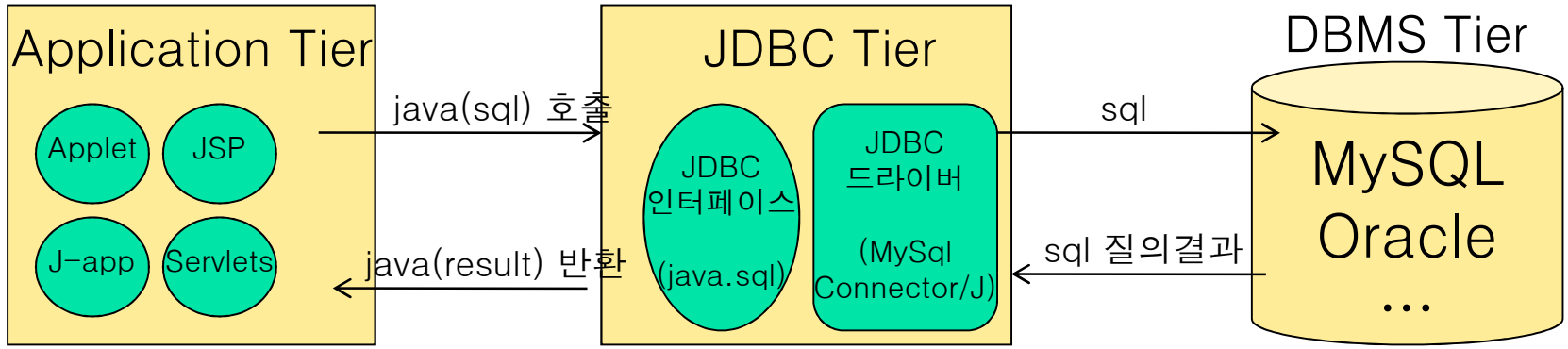
JDBC의 개요

- JDBC의 구성



JDBC의 개요

- JDBC의 역할



① 응용프로그램(Application Program)에서 DBMS가 알아볼 수 있는 SQL문을 String으로 만들어 JDBC의 클래스 인자로 전달

② JDBC는 인자로 넘어온 SQL문을 DBMS에 넘겨 실행

③ DBMS는 SQL문 처리 결과를 JDBC로 반환

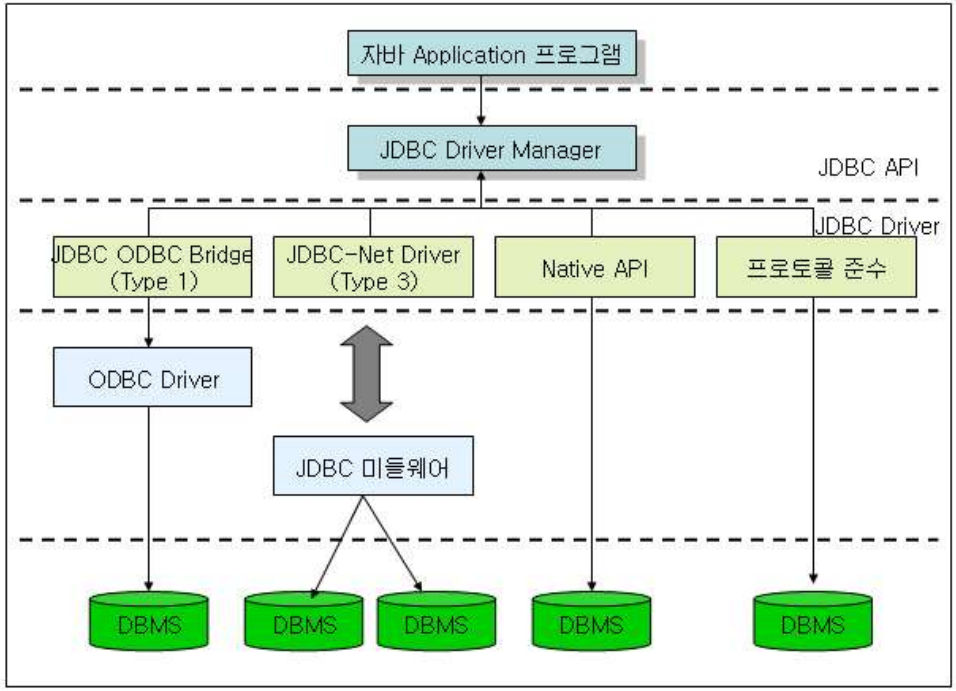
④ JDBC는 DBMS로 부터 받은 처리 결과를 자바 객체 형태로 응용프로그램에 반환

JDBC의 개요

- JDBC 드라이버의 종류
  - Type 1 : JDBC-ODBC Bridge Driver
  - Type 2 : Native-API Driver : OCI Driver (Partly Java Driver)
  - Type 3 : Net-Protocol Driver (All Java Driver)
  - Type 4 : Native-Protocol Driver(thin Driver)

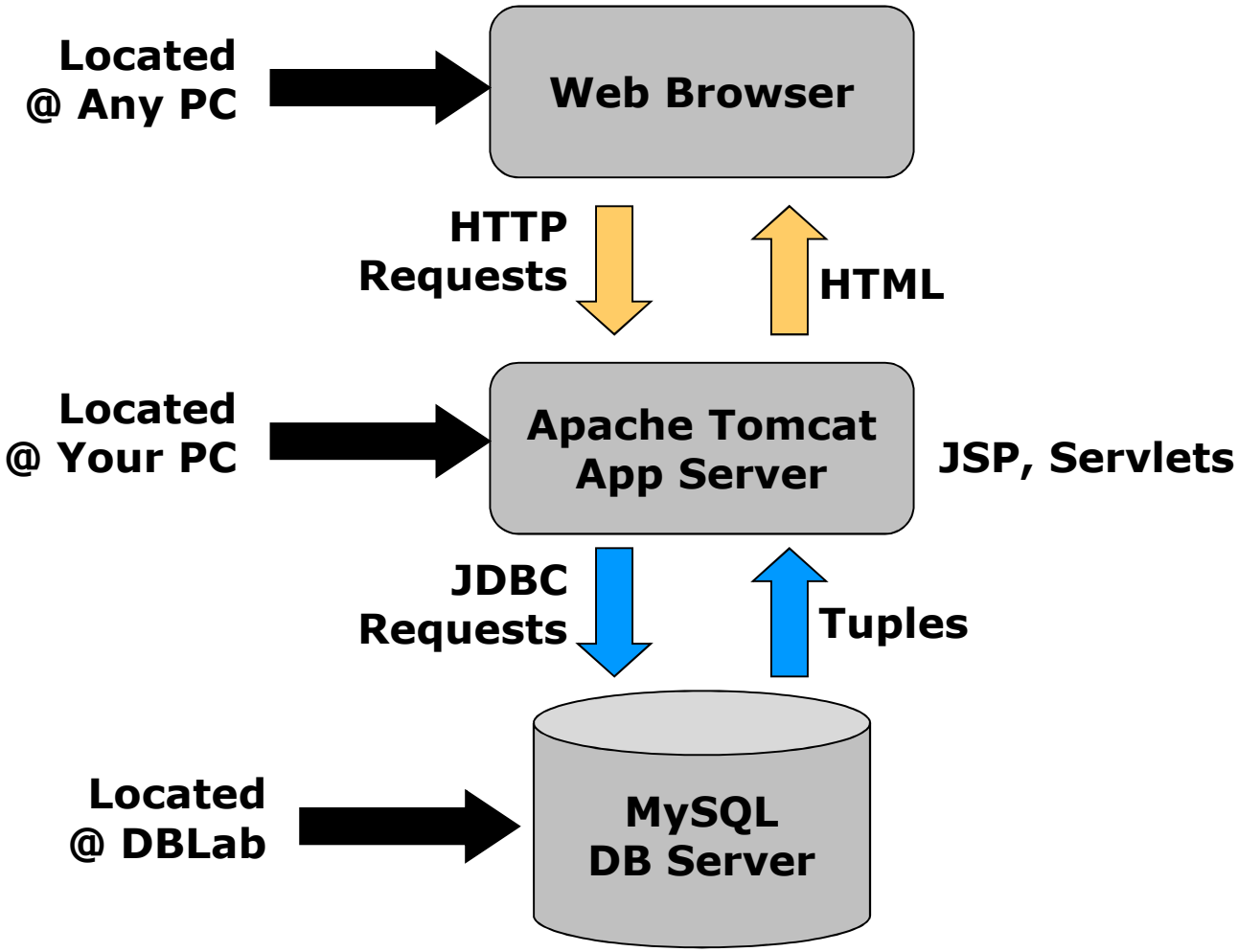
ODBC만 지원하는 DBMS에서 사용

가장 일반적 예: MySQL Connector/J



JDBC의 개요

- WebDB에서의 JDBC

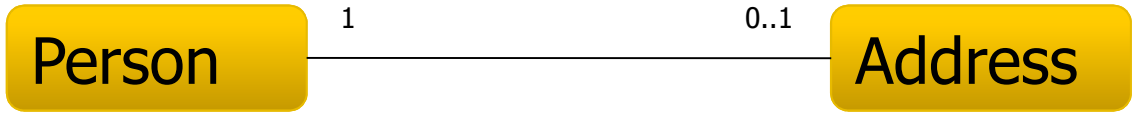


RDB의 장점과 단점

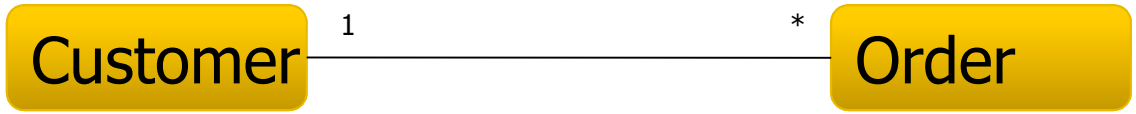
- RDB의 장점
  - 대용량 데이터에 대한 작업 용이성
    - 검색(searching), 정렬(sorting)
  - 데이터 집합에 대한 연산 지원
    - 조인(Joining), 집계(aggregating)
  - 데이터의 공유와 접근
    - 동시성제어(Concurrency)와 트랙잭션(Transactions)
    - 다수의 응용 프로그램 동시 접근 지원
  - 무결성
    - 제한자(Constraints)
    - 트랜잭션(Transaction isolation)

RDB 설계 하기– 관계 표현

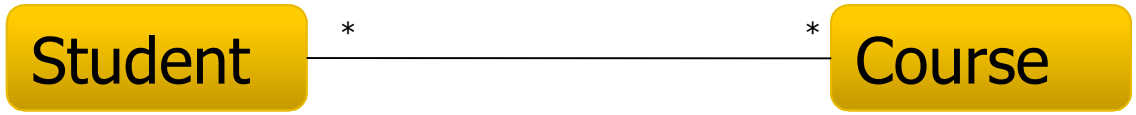
- One-to-One



- One-to-Many

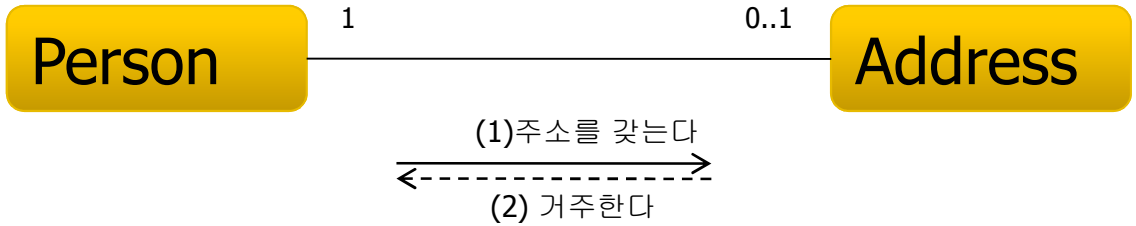


- Many-to-Many



RDB 설계 하기 : RDB 맵핑 – 1:1

- 1:1 관계의 데이터베이스 표현




```
create table person (
   personid  integer, -- PK
   name      varchar(32),
   phone     varchar(32),
   address_id   integer
);
```

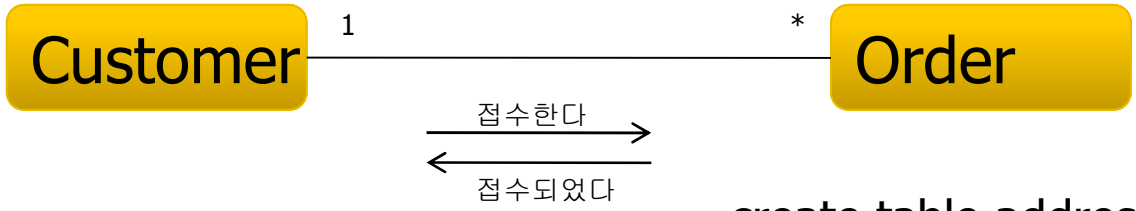
Foreign Key

```
create table address (
   addressid    integer, -- PK
   street       varchar(32),
   zip          integer,
   city         varchar(32)
);
```

- '남광우'의 주소를 검색하라(SQL ?)
- '군산'에 사는 모든 사람을 검색하라(SQL ?)

RDB 설계 하기 : RDB 맵핑 – 1:n

- 1:n 관계의 데이터베이스 표현



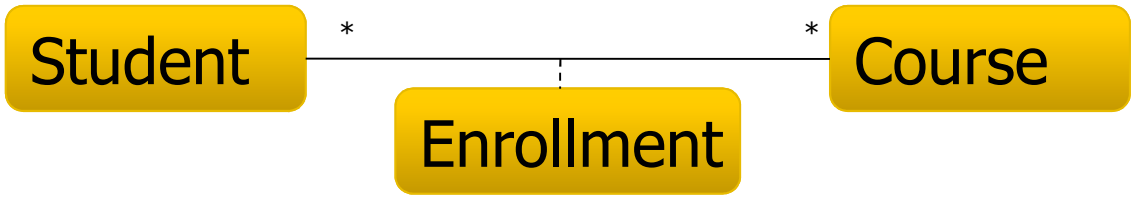

```
create table customer (
   customerid   integer,
   name         varchar(32),
   phone        varchar(32),
   address_id   integer
);
```

```
create table address (
   orderid      integer,
   customerid   integer,
   status       varchar(4),
   price        decimal(10,2)
);
```

- '남광우'의 접수한 주문을 모두 검색하라(SQL ?)
- '10만원 이상 주문한' 고객명을 검색하라(SQL ?)

RDB 설계 하기 : RDB 맵핑 – n:m

- n:m 관계의 데이터베이스 표현




```
create table student (
   studentid  integer,
   name   varchar(32),
   phone  varchar(32)
);
```

```
create table course (
   courseid    integer,
   name        varchar(32)
   teacher     varchar(32)
);
```

```
create table enrollment (
   studentid integer,
   courseid integer
   date timestamp
);
```

- '남광우'가 수강하는 모든 과목명을 검색하라(SQL ?)
- '웹DB' 를 수강하는 모든 학생을 검색하라 (SQL ?)

JDBC 프로그래밍의 기본

- Simple JDBC 프로그래밍의 절차
  - ① 드라이버를 로딩한다

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

  - ② DB와 연결하여 Connection을 획득한다

```
Connection conn
    = DriverManager.getConnection("jdbc:mysql://localhost/test?"
      + "user=monty&password=greatsqldb");
```

  - ③ Connection으로부터 SQL을 실행하기 위한 Statement 객체를 획득한다

```
stmt = conn.createStatement();
```

JDBC 프로그래밍의 기본

- Simple JDBC 프로그래밍의 절차(계속)

④ Statement 객체의 API를 이용하여 SQL을 실행한다

```
ResultSet rs = stmt.executeQuery("SELECT * FROM student");
```

⑤ ResultSet 객체에 있는 결과를 For문 등을 통해 처리한다

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

⑥ 사용한 Connection 객체와 Statement 객체를 Close한다

```
conn.close();
stmt.close();
```

JDBC 프로그래밍의 기본 – 예제

```
import java.sql.*;

class JdbcTest {

  public static void main (String args []) throws SQLException
  {
       // Load driver
       Class.forName("com.mysql.jdbc.Driver");
      // Connect to the local database
      Connection conn Connection conn =
       DriverManager.getConnection("jdbc:mysql://localhost/world?user=root&password=");
      // Query the student names
      Statement stmt = conn.createStatement ();
       ResultSet rs = stmt.executeQuery ("SELECT * FROM City");

      // Print out
      while (rs.next ())
                    System.out.println (rset.getString (2));

     //close the result set, statement, and the connection
      rs.close();
      stmt.close();
     conn.close();
}
```

JDBC 프로그래밍의 기본 – 예제

- 실습
  - MySQL의 샘플 World DB 설치
    - http://www.mysql.org/doc/ 의 Example Databases
    - DB 설치 방법 :
      - City, Country, CountryLanguage 테이블로 구성
      - http://dev.mysql.com/doc/world-setup/en/world-setup.html
    - Country 테이블의 스키마

```
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   |     |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int(11)  | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
```

  - 실습내용
    - JDBC를 이용하여 Country 테이블의 ID, Name, Population을 화면에 출력하시오

JDBC 클래스 – 드라이버 로딩

- JDBC 드라이버 로딩
  - 드라이버 인스턴스 및 DriverManager에 대한 드라이버 register 가 자동으로 수행됨

Exception
발생할 수 있음
: 드라이버가 없을 경우

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

// Notice, do not import com.mysql.jdbc.*
// or you will have problems!

public class LoadDriver {
    public static void main(String[] args) {
        try {
            // The newInstance() call is a work around for some
            // broken Java implementations

            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception ex) {
            // handle the error
        }
}
```

JDBC 클래스 – DB 연결

- Connection 객체 생성
  - 다음 두가지로 사용가능
    - DriverManager.getConnection(url, user, password);
    - DriverManager.getConnection(url);

SQLException
발생할 수 있음
: DB또는 패스워드오류

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

...
try {
    Connection conn =
       DriverManager.getConnection("jdbc:mysql://localhost/world?user=root&password=");

    // Do something with the Connection

   ...
} catch (SQLException ex) {
    // handle any errors
    System.out.println("SQLException: " + ex.getMessage());
    System.out.println("SQLState: " + ex.getSQLState());
    System.out.println("VendorError: " + ex.getErrorCode());
}
```

<Another Example>
String DRIVER = "com.mysql.jdbc.Driver";
String URL = "jdbc:ysql://localhost/test";
String USER = "monty"
String PASSWORD = "greatesqldb";

Class.forName(DRIVER);
Connection conn = DriverManager.getConnection
(URL, USER, PASSWORD);

JDBC 클래스 – DB 연결

- Connection의 대표적인 메소드
  - createStatement()
    - SQL문을 Database에 전달하기 위한 Statement객체 생성
  - prepareStatement(String Sql)
    - 파라미터가 포함된 SQL문을 Database에 전달하기 위한 PreparedStatement객체를 생성
  - prepareCall(String sql)
    - 데이터베이스의 Stored Procedure를 호출하기 위해 CallableStatement객체를 생성
  - close()
    - 현재의 Connection객체에 할당된 System Resource를 반환

JDBC 클래스 – SQL 실행

- Statement 객체 생성 및 실행
  - executeQuery() : SELECT문 실행
    - 질의 결과를 포함한 ResultSet 객체를 반환
  - executeUpdate() : INSERT/UPDATE/DELETE문 실행
    - 갱신된 row의 개수를 반환

```
// JDBC connection 생성된 것으로 간주
Statement stmt = null;
ResultSet   rs = null;

stmt = conn.createStatement();
    rs = stmt.executeQuery("SELECT * FROM city");
```

```
<Another Example>
 if (stmt.execute("SELECT foo FROM bar")) {
       rs = stmt.getResultSet();
   }
```

```
// JDBC connection 생성된 것으로 간주
Statement stmt = null;
ResultSet rs = null;

stmt = conn.createStatement();
    rs = stmt.executeUpdate("DELETE FROM city WHERE name = 'Seoul'");
```

JDBC 클래스 – SQL 실행

- Statement 객체의 장단점
  - 장점
    - 사용이 편리함
      - SQL문을 자유롭게 생성할 수 있으며, 원하는 Query문을 직접 넘겨주기 때문에 직관적이고 쉬운 방법으로 Query를 할 수 있음
  - 단점
    - 디버깅이 어렵다.
      - 잘못된 SQL문의 에러는 런타임(Runtime)시에만 알 수 있다. 따라서 프로그램 디버깅이 힘들다.
    - Overhead가 많다.
      - Statement의 메소드들은 String으로 SQL문을 직관적으로 구성할 수는 있지만 그렇기 때문에 매번 SQL문을 실행할 때마다 그 구문을 해석하는 Overhead가 따른다.
    - SQL문을 재사용하기 힘들다.
      - 같은 구조임에도 불구하고 조건이 조금씩 다른 SQL문일 경우 Statement의 메소드들은 SQL문을 재사용할 수 없고 다시 코드로 작성해 주어야 한다

JDBC 클래스 – SQL 실행

- PreparedStatement 객체의 생성 및 실행
  - 미리 생성된 질의 Statement를 재활용하여 사용할 수 있음
  - PreparedStatement와 Statement의 비교
    - Statement가 항상 String 결합(+)을 사용하여야 하는데 비하여 PreparedStatement는 고정형태를 사용할 수 있음

<PreparedStatement>

```
String val = "abc";
PreparedStatement pstmt =  conn.prepareStatement("select * from R where A=?");
pstmt.setString(1, val);
ResultSet rs =  pstmt.executeQuery();
```

<Statement>

```
String val = "abc";
Statement stmt =  conn.createStatement( );
ResultSet rs =  stmt.executeQuery("select * from R where A=" + val);
```

JDBC 클래스 – SQL 실행

- PreparedStatement 객체의 생성 및 실행(계속)
  - PreparedStatement를 이용한 SELECT 질의 수행

```
PreparedStatement    pstmt;
ResultSet            rs;

String queryStr = "SELECT * FROM city" +
                  "WHERE district = ? and population > ?";

pstmt = con.prepareStatement(queryStr);

pstmt.setString(1, “califonia");
pstmt.setInt(2, 100000);

rs = pstmt.executeQuery();

pstmt.setString(1, “califonia");
pstmt.setInt(2, 500000);

rs = pstmt.executeQuery();
```

<주의!!>
다음과 같이 테이블명에 대해서는 ?를 사용할 수 없음!
컬럼명과 값에 대해서만 가능

```
PreparedStatement pstmt =  conn.prepareStatement("select * from ?");(X)

pstmt.setString(1, myFavoriteTableString); (X)
```

JDBC 클래스 – SQL 실행

- PreparedStatement 객체의 생성 및 실행(계속)
  - PreparedStatement를 이용한 UPDATE 질의 수행

```
String queryStr = “DELETE FROM city" +
                  "WHERE district = ? and population > ?";

PreparedStatement pstmt = conn.prepareStatement(deleteStr);

pstmt.setString(1, “California");
pstmt.setDouble(2, 10000);

int delnum = pstmt.executeUpdate();
```

JDBC 클래스 – SQL 실행

- PreparedStatement의 장점
  - 작은 따옴표를 사용하지 않고 간편하게 사용 가능
    - Statement를 사용할 경우

```
String    name;
Statement   stmt = conn.createStatement();
rs = stmt.executeQuery( “SELECT * FROM city WHERE district =‘”+ name + “’”);
```

    - PreparedStatement를 사용할 경우

```
PreparedStatement   pstmt = conn.prepareStatement
                                    ( “SELECT * FROM city WHERE district = ? ”);
pstmt.setString( 1, name );
rs = pstmt.executeQuery();
```

  - SQL 질의 스트링을 재사용할 수 있음

JDBC 클래스 – 질의 결과의 사용

- ResultSet의 사용
  - ResultSet은 SQL 쿼리 실행 결과를 추상화 한 Interface로서 Database의 Table과 비슷한 형식의 Row와 Column으로 접근

```
ResultSet rs = stmt.executeQuery ("SELECT name, district, population FROM City");

while (rs.next ())
        System.out.println (rset.getString (2));
```

  - next()
    - 결과를 얻을 수 있는 행으로 이동하면서 결과의 획득 여부를 boolean타입으로 리턴한다.
  - close()
    - 현재 객체에 할당된 시스템 자원을 즉시 반환한다.
  - getString(int n)
    현재 행에서 n번째 컬럼의 데이터를 String값으로 얻어온다.

JDBC 클래스 – 질의 결과의 사용

- ResultSet의 사용

  - getString(String columnName)
    현재 행에서 column명이 columnName인 데이터를 String값으로 얻어온다.

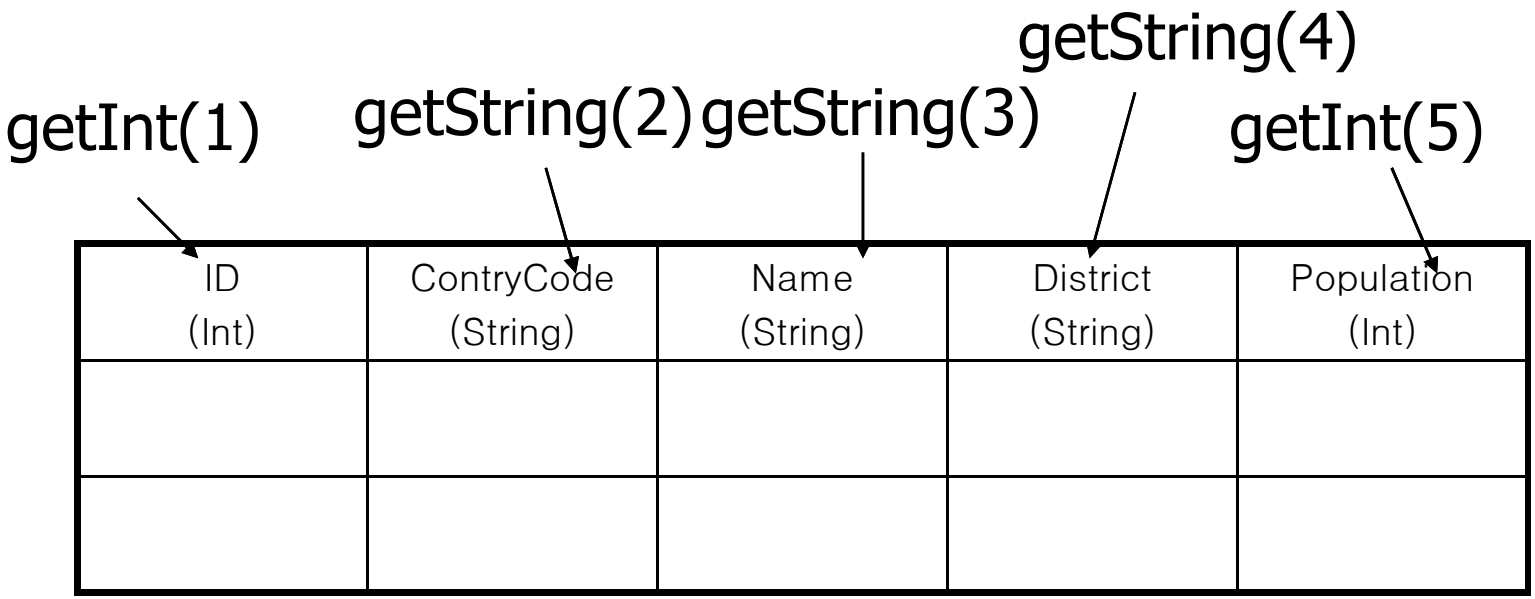
  - getInt(int n)
    현재 행에서 n번째 컬럼의 int값을 얻어온다.

  - getInt(String columnName)
    현재 행에서 column명이 columnName에 해당하는 셀의 데이터를 Int값으로 얻어온다.

  - 이 밖에, getDecimal(), getDouble() 또는 getDate() 사용가능

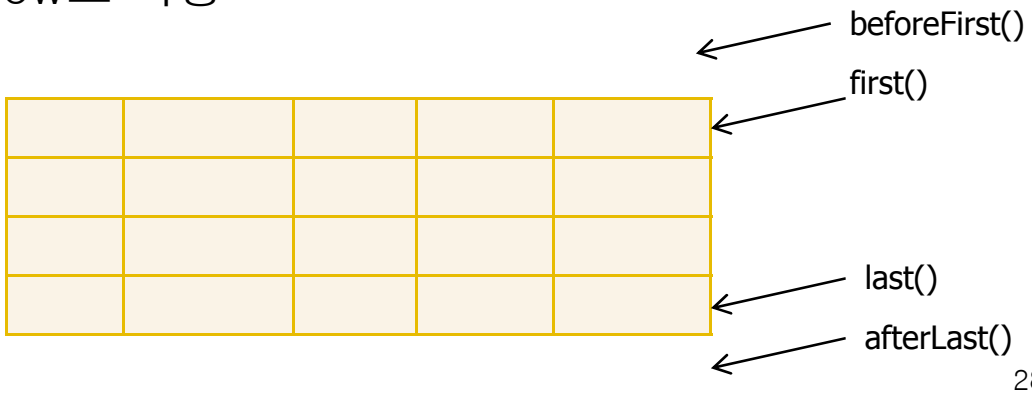
JDBC 클래스 – 질의 결과의 사용

- ResultSet의 사용



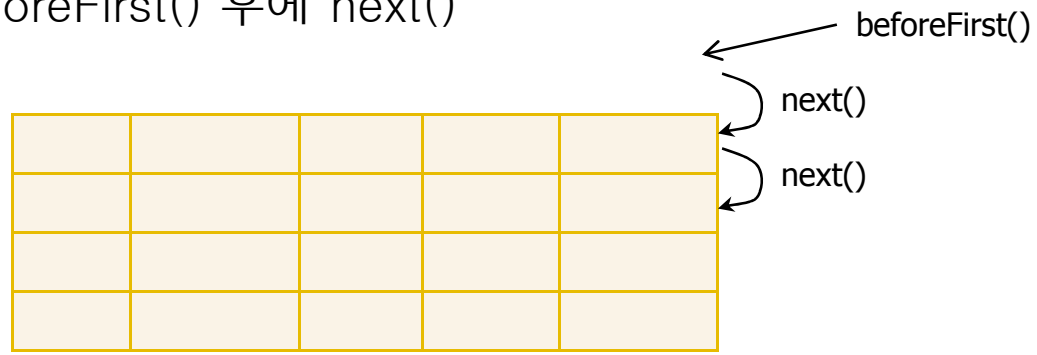
JDBC 클래스 – 질의 결과의 사용

- Scrollable ResultSet의 사용
  - beforeFirst()
    - Resultset의 제일 처음 결과 row의 앞으로 이동
  - first()
    - Resultset의 제일 처음 결과 row로 이동
  - last()
    - Resultset의 제일 마지막 결과 row로 이동
  - afterLast()
    - Resultset의 제일 마지막 결과 row의 다음으로 이동
  - next()
    - Resultset의 다음 row로 이동

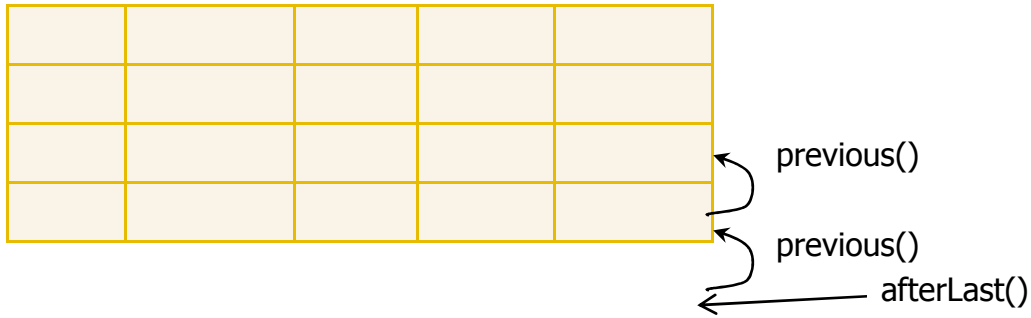


JDBC 클래스 – 질의 결과의 사용

- Scrollable ResultSet의 사용
  - 결과를 제일 처음부터 끝까지 보기
    - beforeFirst() 후에 next()



  - 결과를 마지막에서부터 역순으로 보기
    - afterLast() 후에 previous()



JDBC 클래스 – 질의 결과의 사용

- Scrollable ResultSet의 사용
  - ResultSet을 Scroll하며 데이터를 검색할 수 있음
    - 드라이버에서 지원하는 경우만 사용가능

```
stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_READ_ONLY);

 ResultSet srs = stmt.executeQuery("SELECT * FROM city");

 srs.absolute(4);
 int rowNum = srs.getRow(); // rowNum should be 4
 System.out.println("rowNum should be 4 " + rowNum);

 srs.relative(-3);
 rowNum = srs.getRow(); // rowNum should be 1
 System.out.println("rowNum should be 1 " + rowNum);

 srs.afterLast();
 while (srs.previous()) {
   String name = srs.getString("NAME");
   String district = srs.getFloat("DISTRICT");
   System.out.println(name + "     " + district );
 }
```

상대적 또는 절대적 위치 이동

```
relative( int index )

absolute( int index )
```

JDBC 클래스 – 질의 결과의 사용

- ResultSet의 Delete
  - ResultSet에서 특정 Row를 삭제할 수 있음

```
String query = "select * from city";
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
rs = stmt.executeQuery(query);
while (rs.next()) {
    String id = rs.getString(1);
    String name = rs.getString(2);
    System.out.println("id=" + id + "  name=" + name);
}
rs.first();
rs.deleteRow();
rs.beforeFirst();
while (rs.next()) {
    String id = rs.getString(1);
    String name = rs.getString(2);
    System.out.println("id=" + id + "  name=" + name);
}
```

JDBC 클래스 – 질의 결과의 사용

- ResultSet의 Insert
  - ResultSet에서 특정 Row를 삽입할 수 있음

```
String query = "select * from city";
stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
rs = stmt.executeQuery(query);
while (rs.next()) {
    String id = rs.getString(1);
    String name = rs.getString(2);
    System.out.println("id=" + id + "  name=" + name);
}
// Move cursor to the "insert row"
rs.moveToInsertRow();
// Set values for the new row.
rs.updateString("name", "Kunsan");
rs.updateString("district", "Jeonbuk");
rs.updateString("population", "3000000");
// Insert the new row
rs.insertRow();
// scroll from the top again
rs.beforeFirst();
while (rs.next()) {
  String id = rs.getString(1);
  String name = rs.getString(2);
  System.out.println("id=" + id + "  name=" + name);
}
```

JDBC 클래스 – 메타 데이터 사용

- MetaData의 생성과 접근
  - 프로그래머는 Database의 Table구조를 모르더라도 메타데이터를 통해 동적으로 SQL데이터타입을 얻어내어 적절하게 자바 프로그램에서 Database를 사용할 수 있음
  - JDBC는 Database에 대한 정보를 얻어오는 System 수준의 DatabaseMetaData인터페이스와, 테이블의 칼럼에 대한 정보를 얻어오기 위해 ResultSetMetaData라는 인터페이스 제공

시스템 수준 Metadata

```
DatabaseMetaData  dbMeta = conn.getMetaData();
```

테이블 수준 Metadata

```
ResultSet rs = stmt.executeQuery( sql );
ResultSetMetaData = rs.getMetaData();
```

JDBC 클래스 – 메타 데이터 사용

- DatabaseMetaData의 사용
  - DB 내의 테이블들에 대한 정보 검색
    - getTables(String catalog, String schemaPattern, String tableNamePattern, String[ ] types)
  - 테이블들의 각 컬럼별 데이터 타입 등 정보 검색
    - ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String columnNamePattern)
  - 데이터베이스 이름 보기
    - String getDatabaseProductName()
  - 기타
    - String getDatabaseProductVersion()
    - String getDriverVersion()

JDBC 클래스 – 메타 데이터 사용

- DB의 Table 정보 보기

```
public void showTables( Connection conn ) throws SQLException {
   DatabaseMetaData metadata = null;

  metadata = conn.getMetaData();
  String[] names = {"TABLE"};

  ResultSet tableNames = metadata.getTables(null,"%", "%", names);
  while (tableNames.next()) {
     System.out.println( tableNames.getString("TABLE_NAME"));
   }
}
```

- Table의 컬럼 보기

```
public void readTableColumns(DatabaseMetaData meta, String tableName) throws SQLException {
    ResultSet columns = meta.getColumns(null, "%", tableName, "%");
    while (columns.next()) {
       String columnName = columns.getString("COLUMN_NAME");
       String datatype = columns.getString("TYPE_NAME");
       int datasize = columns.getInt("COLUMN_SIZE");
       int digits = columns.getInt("DECIMAL_DIGITS");
       int nullable = columns.getInt("NULLABLE");
       boolean isNull = (nullable == 1);
       // 화면에 프린드
   }
 }
```

JDBC 클래스 – 메타 데이터 사용

- ResultSetMeta 데이터의 사용
  - ResultSet의 Column 개수 반환
    - int getColumnCount()
  - 지정 컬럼의 컬럼명을 반환
    - int getColumnName(int column)
  - 지정 컬럼의 데이터 타입을 반환
    - String getColumnTypeName(int column)
  - 지정 컬럼의 Null 여부를 반환
    - int isNullable(int column)

JDBC 클래스 – 메타 데이터 사용

- ResultSetMeta 데이터의 사용
  - 컬럼별로 화면 출력하기

```
public void  showTable( ResultSet  rs )
{
      ResultSetMetaData rsmd = rs.getMetaData();
      int numberOfColumns = rsmd.getColumnCount();
      int rowCount = 1;
      while (rs.next()) {
        System.out.println("Row " + rowCount + ":  ");
        for (int i = 1; i <= numberOfColumns; i++) {
          System.out.print("    Column " + i + ":  ");
          System.out.println(rs.getString(i));
        }
        System.out.println("");
        rowCount++;
      }
}
```

JDBC 클래스 – 실습

- 간단 DB Admin 프로그램
  - City 테이블을 이용하여 다음의 기능을 하는 프로그램을 작성

```
C:> java simpleAdmin

다음기능중에 하나를 선택하세요
1. DB내의 Table들 보기
2. 특정 Table의 컬럼 정보 보기
3. Table의 내용 보기

선택>
```

    - Table 들 보기
      - DB 안의 모든 테이블들을 리스트
    - 특정 Table의 컬럼 정보 보기
      - 특정 테이블 이름을 입력하면 화면에 테이블의 컬럼 정보 리스트
    - Table의 내용보기
      - 특정 테이블의 이름을 입력하면 테이블의 내용을 리스트
      - 단 10개씩 끊어서 “엔터”키가 입력될 때 다음 페이지 보이기

JSP로 DB 내용 보기

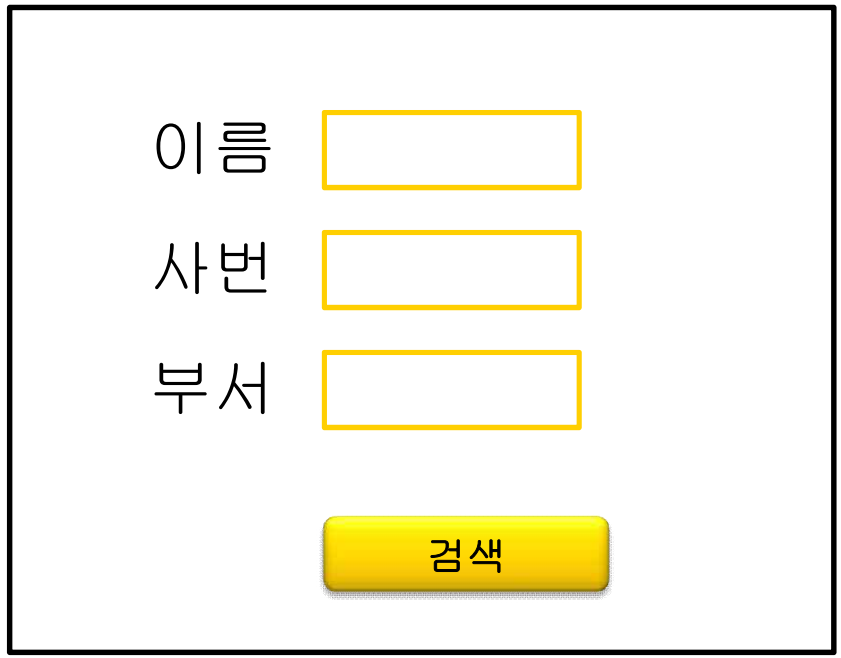
- JSP를 이용한 Table 그리기
  - Table 9x9단 그리기

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>Insert title here</title>
</head>
<body>
<center><h3>JSP를 이용한 테이블</h3></center>
<%int row =9;int col =9;%>
<table border=1>
        <%for(int i=2; i <= row; i++){ %>
                <tr>
                        <%for(int j=1; j <= col; j++) { %>
                        <td>
                        <%out.println("("+(i + " * " + j + "=" + i * j ) +")"); %>
                        </td>
                        <% } %>
                </tr>
        <% } %>
</table>
</body>
</html>
```

2 * 1= 2	3 * 1= 3	4 * 1= 4	5 * 1= 5	6 * 1= 6	7 * 1= 7	8 * 1= 8	9 * 1= 9
2 * 2= 4	3 * 2= 6	4 * 2= 8	5 * 2= 10	6 * 2= 12	7 * 2= 14	8 * 2= 16	9 * 2= 18
2 * 3= 6	3 * 3= 9	4 * 3= 12	5 * 3= 15	6 * 3= 18	7 * 3= 21	8 * 3= 24	9 * 3= 27
2 * 4= 8	3 * 4= 12	4 * 4= 16	5 * 4= 20	6 * 4= 24	7 * 4= 28	8 * 4= 32	9 * 4= 36
2 * 5= 10	3 * 5= 15	4 * 5= 20	5 * 5= 25	6 * 5= 30	7 * 5= 35	8 * 5= 40	9 * 5= 45
2 * 6= 12	3 * 6= 18	4 * 6= 24	5 * 6= 30	6 * 6= 36	7 * 6= 42	8 * 6= 48	9 * 6= 54
2 * 7= 14	3 * 7= 21	4 * 7= 28	5 * 7= 35	6 * 7= 42	7 * 7= 49	8 * 7= 56	9 * 7= 63
2 * 8= 16	3 * 8= 24	4 * 8= 32	5 * 8= 40	6 * 8= 48	7 * 8= 56	8 * 8= 64	9 * 8= 72
2 * 9= 18	3 * 9= 27	4 * 9= 36	5 * 9= 45	6 * 9= 54	7 * 9= 63	8 * 9= 72	9 * 9= 81

JSP로 DB 내용 보기

- 실습 : 사원 테이블 검색
  - DB
    - employee = {empno, ename, job, hiredate}
  - 검색 화면



JSP로 DB 내용 보기

- 실습 : 사원 테이블 검색 jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.List, module.IbatisModule , ex1.vo.EmpVO,ex1.vo.DeptVO" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">

<%! //선언부 초기화하지 못한다
   String code;
   String deptno;
   String empno;
   List list;
%>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>
       <%
         list = request.getParameter("result");
       %>
       <table border="1" width="500">
          <thead>
             <tr>
                <th>사원번호</th>
                <th>이름</th>
                <th>직원</th>
                <th>입사일</th>
             </tr>
          </thead>
```

JSP로 DB 내용 보기

- 실습 : 사원 테이블 검색

```
<%
        for(EmpVO e : (List<EmpVO>)list){
    %>
            <tr align="center">
                <td><%= e.getEmpno()%></td>
                <td><%= e.getEname()%></td>
                <td><%= e.getJob()%></td>
                <td><%= e.getHiredate()%></td>
            </tr>
    <%
     }//for문 끝
    %>
        </tbody>
     </table>
    <%
    }//end if
    %>
   </body>
</html>
```

[출처] JSP 연습문제: 사원번호, 부서번호를 검색하여 결과를 DB에서 가져와 table로 나타내기|**작성자** paran_java