

# 6장. 데이터베이스를 이용한 프로그래밍 1

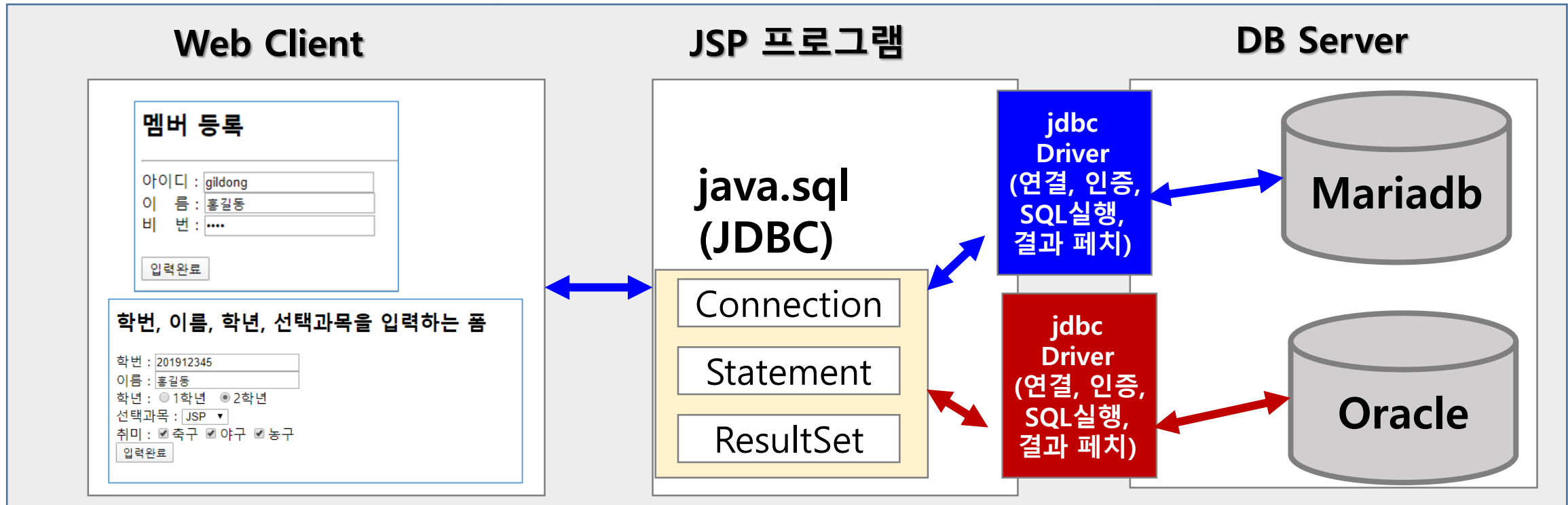
컴퓨터정보과  
김진숙

# 이 장에서 공부할 것

- JSP를 이용하여 데이터베이스를 사용하는 방법
- 학습내용
  - 데이터베이스 접속 및 종료
  - 데이터베이스의 내용을 변경하는 쿼리 실행
  - 데이터를 조회하는 쿼리 실행

# 1. JDBC를 사용한 JSP와 데이터베이스 연동

- JDBC(Java Database Connectivity) 라이브러리(Library)
  - 관계형 데이터베이스에 접근하고, SQL 실행하는 인터페이스
  - JDBC 라이브러리는 'java.sql' 패키지에 의해 구현
- JDBC Driver
  - 실제 해당 DB에 맞는 구동 코드를 가지고 있음
  - DB Server사에서 제공(mariadb 다운로드 파일에 첨부되어 있음)



# [참조] JDBC Driver 설치

- 자바 커넥터 파일 다운로드
  - mariadb-java-client-2.7.5.jar

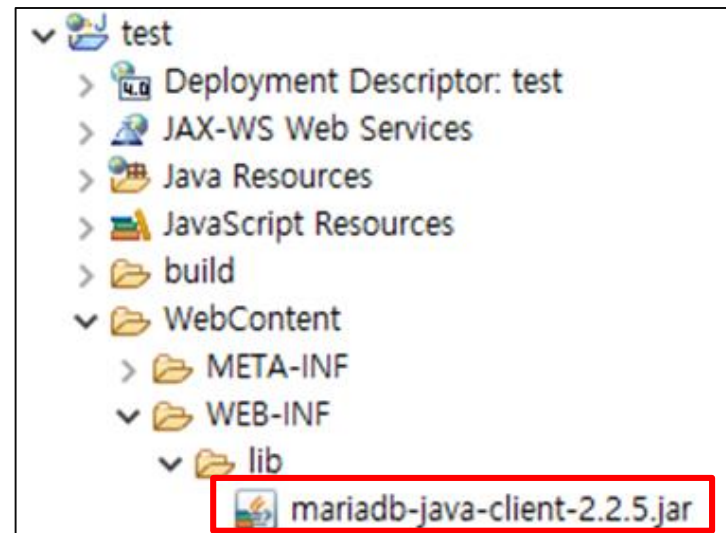
The screenshot shows the MariaDB Connectors website. At the top, there are navigation tabs: 'MariaDB Platform', 'apt/yum', 'Connectors' (highlighted with a red box), and 'Tools'. Below these are sub-tabs: 'Data Access' (highlighted), 'Data Ingestion', and 'Data Streaming'. The main content area features the text 'Lightweight, advanced connectors for high-performance data access and data streaming.' and a description of MariaDB Connector/J. A 'Product' dropdown menu is set to 'Java 8 connector' (highlighted with a red box). At the bottom, there is a table with columns for the connector name, download URL, size, and a 'Download' button (highlighted with a red box).

Product	Download URL	Size	Download
Java 8 connector	<a href="https://downloads.mariadb.com/Connectors/java/connector-java-2.4.2/mariadb-java-client-2.4.2.jar">https://downloads.mariadb.com/Connectors/java/connector-java-2.4.2/mariadb-java-client-2.4.2.jar</a>	573.93 KB	Download

# [참조] JDBC Driver 설치

## ■ 자바 커넥터 준비

- 데이터베이스를 사용하는 JSP 프로젝트에는 반드시 자바 커넥터(JDBC 드라이버라고도 한다)가 포함되어 있어야 함
- 지난 장에서 oracle DB를 위한 자바 커넥터를 다운로드 받아 보관에 두었음
- 이클립스의 Project Explorer에서, 프로젝트 > **WebContent > WEB-INF > lib** 폴더로 준비해 둔 자바 커넥터 파일을 드래그



# [참조] JDBC 관련 기본 클래스, 인터페이스, 메소드

The screenshot shows the Oracle Java API documentation for the `java.sql.Connection` interface. The browser address bar shows `docs.oracle.com/javase/8/docs/api/`. The left sidebar lists the package hierarchy, with `java.sql` and `Connection` highlighted. The main content area shows the `Connection` interface with tabs for `All Methods`, `Instance Methods`, and `Abstract Methods`. The `All Methods` tab is selected, displaying a table of methods.

Modifier and Type	Method and Description
void	<b>abort</b> ( <code>Executor</code> executor) Terminates an open connection.
void	<b>clearWarnings</b> () Clears all warnings reported for this Connection object.
void	<b>close</b> () Releases this Connection object's database and JDBC resources immediately instead of waiting for them to be automatically released.
void	<b>commit</b> () Makes all changes made since the previous commit/rollback permanent and releases any database locks currently held by this Connection object.
Array	<b>createArrayOf</b> ( <code>String</code> typeName, <code>Object[]</code> elements) Factory method for creating Array objects.
Blob	<b>createBlob</b> ()

Java api 문서

<https://docs.oracle.com/javase/8/docs/api/>

# 1. JDBC를 사용한 JSP와 데이터베이스 연동

- JDBC 관련 기본 클래스, 인터페이스, 메소드

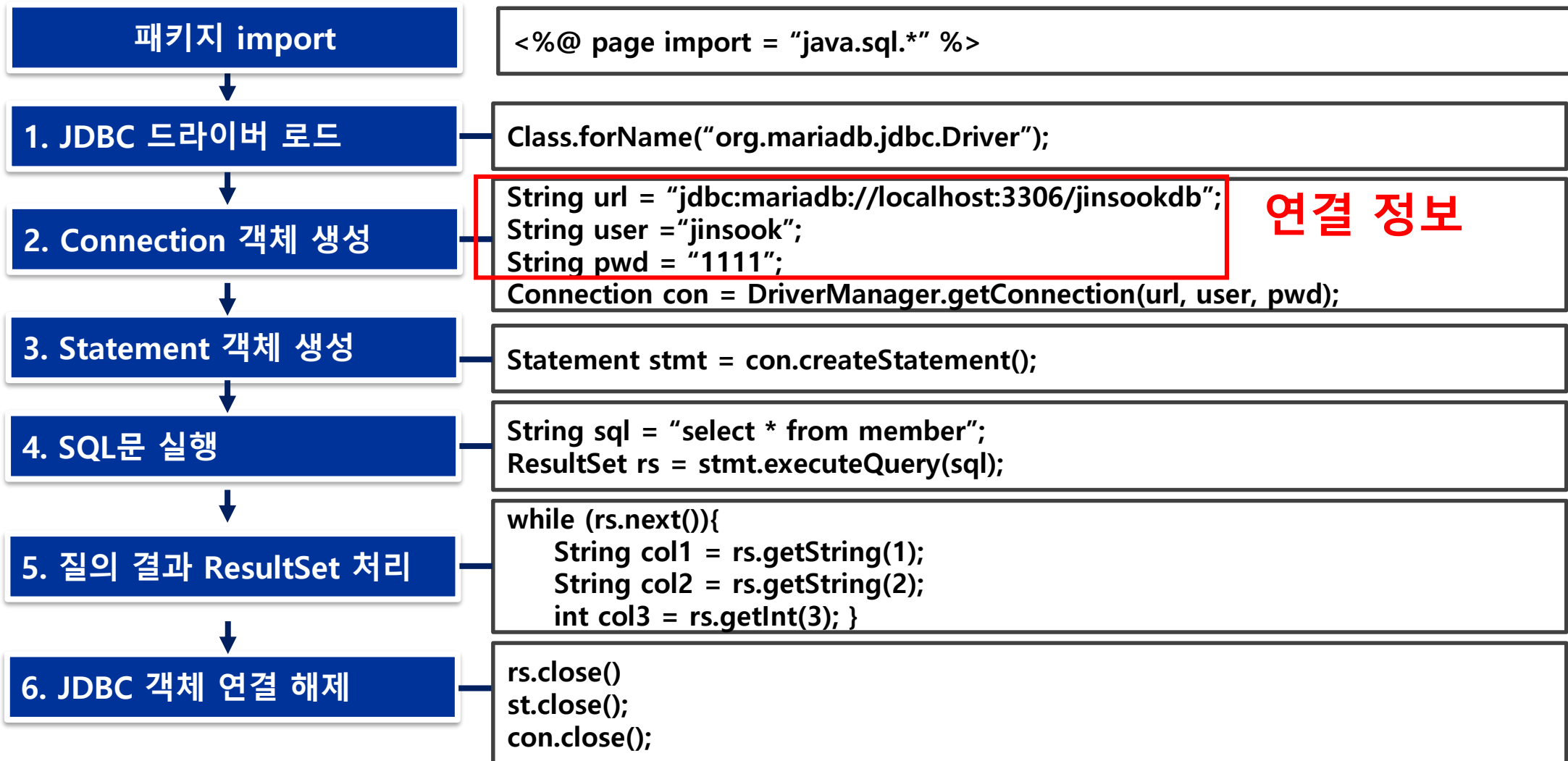
패키지	인터페이스 /클래스	클래스 용도	이용 메소드
java.lang	<b>Class</b> (클래스)	지정된 JDBC 드라이버를 실행시간 동안 메모리에 로드	<b>forName();</b>
java.sql	<b>DriverManager</b> (클래스)	여러 JDBC 드라이버를 관리하는 클래스로 데이터베이스를 접속하여 연결 객체 반환	<b>getConnection();</b>
	<b>Connection</b> (인터페이스)	특정한 데이터베이스 연결 상태를 표현하는 클래스로 질의할 문장 객체 반환	<b>createStatement(); close();</b>
	<b>Statement</b> (인터페이스)	데이터베이스에 SQL 질의 문장을 질의하여 그 결과인 결과 집합 객체를 반환	<b>executeQuery(); close();</b>
	<b>ResultSet</b> (인터페이스)	질의 결과의 자료를 저장하며 테이블 구조를 가짐	<b>next(); getString(); getInt(); close();</b>

Java api 문서

<https://docs.oracle.com/javase/8/docs/api/>

# 1. JDBC를 사용한 JSP와 데이터베이스 연동

## ■ JDBC프로그램의 작성단계(SELECT 문 기준)



연결 정보



# 실습1 – DB연동

- 준비사항
  - Maria DB
    - DB : 각자이름db(예 : jinsookdb)
    - 사용자 : 각자이름(예 : jinsook)
    - 비밀번호 : 1111
  - JDBC Driver 설치
    - [WebContent] – [WEB-INF] – [lib] 에 mariadb-java-client-2.7.5.jar 저장해 두기

# 실습1 - DB연동

데이터베이스를 사용하기 위해서  
java.sql 패키지를 импорт

```
1 <%@page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@page import="java.sql.DriverManager"%>
4 <%@page import="java.sql.Connection"%>
5 <%
6   //1. DB 연동 드라이버 로드
7   Class.forName("org.mariadb.jdbc.Driver");
8
9   //2. 연결 정보 설정
10  String url = "jdbc:mariadb://localhost:3306/jinsookdb";
11  String user = "jinsook";
12  String pwd = "1111";
13
14  //3. 연결 객체 생성
15  Connection con = DriverManager.getConnection(url, user, pwd);
16
17  out.println("연결 성공");
18
19  //4. 연결 객체 해제
20  con.close();
21 %>
22 <!DOCTYPE html>
23 <html>
24 <head>
25   <meta charset="UTF-8">
26   <title>DB 연동</title>
27 </head>
28 <body>
29
30 </body>
31 </html>
```

4월 26, 2020 9:57:36 오후 org.apache.catalina.startup.Catalina start  
정보: 서버가 [314] 밀리초 내에 시작되었습니다.  
java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver  
at org.apache.catalina.loader.WebappClassLoaderBase.loadC  
at org.apache.catalina.loader.WebappClassLoaderBase.loadC  
at org.apache.jasper.servlet.JasperLoader.loadClass(Jaspe  
at org.apache.jasper.servlet.JasperLoader.loadClass(Jaspe  
at java.lang.Class.forName0(Native Method)  
at java.lang.Class.forName(Unknown Source)

위의 예러는 대부분 jdbc 드라이버  
파일이 프로젝트에 등록되어  
있지 않을 때 발생한다.

# [참조] 데이터베이스 접속 및 종료 : try-with-resources

```
12: <%  
13:   Class.forName("org.mariadb.jdbc.Driver");  
14:   try (  
15:       Connection conn = DriverManager.getConnection(  
16:           "jdbc:mariadb://localhost:3306/jspdb", "jsp", "1234");  
17:   ) {  
18:  
19:       out.println("DB 접속 성공 !");  
20:  
21:   } catch(Exception e) {  
22:       e.printStackTrace();  
23:   }  
24: %>  
25:  
26: </body>  
27: </html>
```

자바 커넥터 로드

데이터베이스 접속

```
Connection conn = DriverManager.getConnection(  
    "jdbc:mariadb://서버주소:3306/사용할DB", "사용자ID", "비밀번호");
```

try-with-resources 문

try 다음의 괄호 사이에 데이터베이스 리소스(여기에서는 접속 정보)를 할당하는 문장을 써넣으면 try~catch 구문의 실행이 끝날 때 자동적으로 이 리소스를 해제(close) 해준다.

Java Bean 공부시 다룰 예정

## [참조] Try-with-resources

- try에 자원 객체를 전달하면, try 코드 블록이 끝나면 자동으로 자원을 종료해주는 기능
- 따로 finally 블록이나 모든 catch 블록에 종료 처리를 하지 않아도 됨
- 이 때, try에 전달할 수 있는 자원은 AutoCloseable 인터페이스의 구현체로 한정됨
- AutoCloseable은 JDK1.7부터 추가된 인터페이스

# 1단계 : JDBC 드라이버 로드

```
Class.forName("org.mariadb.jdbc.Driver");
```

- 인터페이스 드라이버(interface driver) 구현(implements) 작업
  - Mariadb Driver 클래스를 객체화
- forName() 메소드는 ClassNotFoundException 발생시키므로 반드시 예외처리 해야 함(Servlet에서)
- 자동으로 DriverManager클래스 메소드를 호출하여 인스턴스 등록
  - DriverManager클래스의 모든 메소드는 **static**이므로 반드시 객체를 생성시킬 필요 없음

# DBMS 종류에 따른 JDBC 드라이버

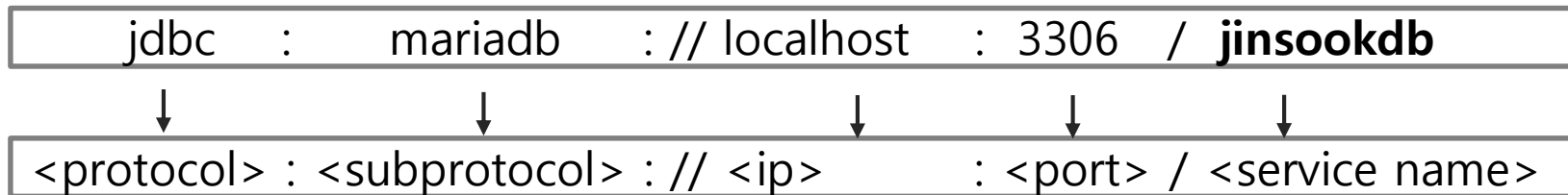
DBMS	JDBC 드라이버 로드 문장
mariadb	<code>Class.forName("org.mariadb.jdbc.Driver");</code>
oracle	<code>Class.forName("oracle.jdbc.driver.OracleDriver");</code>
mySQL	<code>Class.forName("com.mysql.jdbc.Driver");</code>
MS SQL	<code>Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");</code>
ODBC	<code>Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");</code>
mSQL	<code>Class.forName("com.imaginary.sql.mssql.MssqlDriver");</code>

## 2단계 : Connection 객체 생성

- 지정한 데이터베이스 **연결 요구** 단계
- DriverManager클래스의 getConnection()을 호출하면 등록된 드라이버 중 주어진 URL로 DB에 연결할 수 있는 드라이버를 찾아 Connection객체 반환
- Connection객체는 close()가 호출될 때까지 데이터베이스와 애플리케이션과의 연결을 지속시킴

```
String url = "jdbc:mariadb://localhost:3306/jinsookdb";  
Connection con = DriverManager.getConnection(url, "user", "passwd");
```

- URL 문자열의 의미



# DBMS 종류에 따른 URL

DBMS	JDBC URL
<b>mariadb</b>	<b>"jdbc:mariadb://localhost:3306/dbname"</b>
Oracle	"jdbc:oracle:thin:@//localhost:1521/XEPDB1"
mySQL	"jdbc:mysql://localhost:3306/dbname"
MS SQL	"jdbc:microsoft:sqlserver://localhost:1433"
ODBC	"jdbc:odbc:basicjsp"
mSQL	"jdbc:msql://localhost:1114/dbname"



## 3단계 : Statement 객체 생성

- Connection객체로 SQL문을 실행하고 결과를 반환 받을 수 있는 Statement 객체 생성 단계
  - Statement 인터페이스
  - PreparedStatement 인터페이스
  - CallableStatement 인터페이스
- 위의 객체들은 자바 코드 내에서는 반드시 try~catch문장을 사용하여 예외 처리해야 함

### 참고

#### CallableStatement 인터페이스

- 저장함수(stored procedure) 호출 시 사용.
- 미리 저장된 쿼리를 사용하므로 수행 속도 빠름

## • Statement 인터페이스

- Statement 객체는 Statement 인터페이스를 구현한 객체
- Connection 객체의 **createStatement()** 호출로 생성
- 단순한 SQL문장을 보낼 때 사용되며 성능, 효율성 면에서 가장 기능이 떨어짐

```
Statement stmt = con.createStatement();
```

## • PreparedStatement 인터페이스

- 각 인수에 대해 위치 홀더(placeholder : "?" )를 사용하여 SQL문을 작성
- 동일한 질의문을 특정 값만 바꾸어 여러 번 실행해야 할 때 유용
- Statement 객체의 SQL은 실행될 때 매번 서버에서 분석되어야 하는 반면 PreparedStatement는 한번 분석되면 재사용
- 컴파일 시 에러 체크를 하기 때문에 효율적이고 처리속도 빠름

**PreparedStatement 객체 제공  
setXxx(num, var)메소드**

```
try{ String sql = "insert into member values(?, ?)";  
    pstmt = con.prepareStatement(sql);  
    pstmt.setString(1, id);  
    pstmt.setString(2, passwd);  
} catch(SQLException e){  
    e.printStackTrace();  
}
```

```
setString(int parameterIndex, String x)  
setInt(int parameterIndex, int x)  
setLong(int parameterIndex, long x)  
setObject(int parameterIndex, Object x)  
setDate(int parameterIndex, Date x)  
setTimestamp(int parameterIndex, Timestamp x)  
setDouble(int parameterIndex, double x)  
setFloat(int parameterIndex, float x)
```

## 4단계 : Query 수행

- Statement 객체가 생성되면 Statement 객체의 **executeQuery()** 메소드나 **executeUpdate()** 메소드를 사용해서 쿼리 처리

```
ResultSet rs = stmt.executeQuery("select * from 소속기관");
```

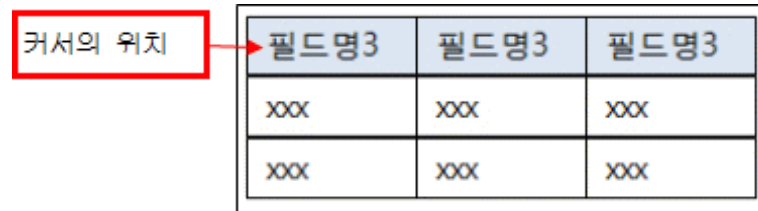
```
int rowCount = stmt.executeUpdate("delete from student where name='홍길동'");
```

- 질의 메소드 종류

종류	반환 유형	특징
<b>executeQuery()</b>	<b>ResultSet</b>	주로 select와 같이 DB에 변경을 주지 않는 SQL문 실행 시 사용 결과로 ResultSet 객체 반환
<b>executeUpdate()</b>	<b>int</b>	DB값이나 구조를 변경하는 insert, update, delete, create, drop 구문 시 주로 사용 질의 수행 후 영향을 받은 행 수를 반환 DDL의 경우에는 0를 반환
<b>execute()</b>	<b>boolean</b>	SQL문이 어떤 것인지 모를 경우 유용

## 5단계 : ResultSet으로 질의 결과 처리(검색 경우)

- executeQuery()메소드는 결과로 ResultSet 반환
- ResultSet 으로부터 원하는 데이터를 추출하는 과정
  - ResultSet객체는 '커서(cursor)'를 사용해 ResultSet 객체의 특정 레코드 참조
  - 커서는 초기에 첫 번째 레코드(행)의 직전 즉, 필드명이 위치한 곳을 가리킴



필드명3	필드명3	필드명3
xxx	xxx	xxx
xxx	xxx	xxx

- 문자열 데이터를 갖는 필드는 rs.getString("name") 혹은 rs.getString(1)로 사용.
  - ResultSet의 첫 번째 필드는 1부터 시작.
  - rs.getString("name")과 같이 필드명을 사용하는 것이 권장 형태.

```
while (result.next()){  
    String col1 = result.getString(1);  
    String col2 = result.getString(2);  
    int col3 = result.getInt(3)  
}
```

```
while (result.next ()) {  
    <%=result.getString("id")%>  
    <%=result.getString("passwd")%>  
    <%=result.getInt("age")%>  
}
```

# ResultSet에서 사용되는 주요 메소드

메소드	반환 유형	기능
<b>absolute(int row)</b> <b>next()</b> <b>previous()</b> <b>first()</b> <b>last()</b> <b>afterLast()</b> <b>beforeFirst()</b>	<b>boolean</b>	커서 이동 메소드 이동하면 true, 이동 못하면 false
<b>getDate()</b> <b>getString()</b> <b>getInt()</b> <b>getLong()</b> <b>getFloat()</b> <b>getDouble()</b>	<b>Date</b> <b>String</b> <b>int</b> <b>long</b> <b>float</b> <b>double</b>	컬럼 자료 값 반환
<b>close()</b>	<b>void</b>	ResultSet 객체의 연결 해제

## 6단계 : JDBC 객체 연결 해제

- 사용했던 JDBC 객체의 연결 해제

```
result.close();  
pstmt.close();  
con.close();
```

- 데이터베이스 연결을 의미하는 Connection 객체는 메모리와 서버의 부하 관점에서 객체 연결 해제가 특히 중요

## 6.3 데이터를 조회하는 쿼리 실행(교재 p.120~123)

- 데이터 조회를 위해 select 쿼리를 실행하는 형식

```
Statement stmt = conn.createStatement();  
ResultSet rs = stmt.executeQuery(쿼리);  
while (rs.next()) {  
    rs.getInt(정수_필드)  
    rs.getString(문자열_필드)  
}
```

- select는 create, insert, delete, update 등과는 달리 값을 꺼내 오는 쿼리
  - 쿼리를 실행할 때도 executeQuery()라는 별도의 메서드를 사용
  - 쿼리의 결과를 어디엔가 담아놓고 사용해야 하므로 반환 값도 받아두어야 한다.
    - executeQuery() 메서드는 ResultSet 클래스 형태의 객체를 반환
    - 이 객체는 쿼리의 결과를 담고 있다
    - 쿼리 결과 객체 안에는 테이블 형태의 데이터가 들어가 있으며, 여기에서 각각의 데이터를 꺼내기 위해서는 추가적인 작업이 필요하다.

## 6.3 데이터를 조회하는 쿼리 실행

- `rs.next()`
  - `rs`가 담고 있는 쿼리의 결과 값(테이블 형태)에서 레코드 포인터를 다음 레코드로 옮기고 `true`를 반환
  - 레코드 포인터란, 현재 관심있는 레코드에 표시를 해둔 것
  - 처음으로 `next()`가 호출되면 첫 번째 레코드를, 다시 `next()`가 호출되면 두 번째, 다시 `next()`가 호출되면 세 번째...와 같은 식으로 `next()`가 호출될 때마다 다음 레코드를 가리키게 된다.
  - 다음 레코드가 남아있지 않으면 `false`를 리턴한다.
  - 따라서 `while`의 조건식이 `true`가 되어 `while` 반복문 안으로 진입했다는 것은 지금 레코드 포인터가 어느 특정한 레코드를 가리키고 있다는 뜻
- 이때 그 레코드에서 특정 필드의 값을 읽어내려면 다음과 같은 메소드를 사용
  - `rs.getInt(정수_필드)`
  - `rs.getString(문자열_필드)`
    - `rs.getInt("num")`은 현재 레코드 포인터가 가리키는 레코드의 `num` 필드 값이 정수인데, 그 값을 읽어 달라는 의미
    - `rs.getString("name")`은 그 레코드의 `name` 필드 값이 문자열인데 그 값을 읽어달라는 의미



## 6.3 데이터를 조회하는 쿼리 실행

[예제 6-5] 데이터 출력 예제 (6-5.jsp)

```
1: <%@ page language="java" contentType="text/html; charset=UTF-8"
2:    pageEncoding="UTF-8"%>
3: <%@ page import="java.sql.*" %>
4:
5: <!DOCTYPE html>
6: <html>
7: <head>
8:    <meta charset="UTF-8">
9:    <style>
10:        table { width: 400px; text-align: center; }
11:        th   { background-color: cyan; }
12:    </style>
13: </head>
14: <body>
15:
16: <table>
17:    <tr>
18:        <th>번호</th> <th>이름</th>
19:        <th>국어</th> <th>영어</th> <th>수학</th>
20:        <th>총점</th> <th>평균</th>
21:    </tr>
```

## 6.3 데이터를 조회하는 쿼리 실행

```
22: <%
23:   Class.forName("org.mariadb.jdbc.Driver");
24:   try (
25:       Connection conn = DriverManager.getConnection(
26:           "jdbc:mariadb://localhost:3306/jspdb", "jsp", "1234");
27:       Statement stmt = conn.createStatement();
28:       ResultSet rs = stmt.executeQuery("select * from score");
29:   ) {
30:       while (rs.next()) {
31:           int sum = rs.getInt("kor") + rs.getInt("eng") +
32:               rs.getInt("math");
33: %>
34:       <tr>
35:           <td><%=rs.getInt  ("num" )%> </td>
36:           <td><%=rs.getString("name")%> </td>
37:           <td><%=rs.getInt  ("kor" )%> </td>
38:           <td><%=rs.getInt  ("eng" )%> </td>
39:           <td><%=rs.getInt  ("math")%> </td>
40:           <td><%=sum%> </td>
41:           <td><%=String.format("%.2f", (float)sum / 3)%> </td>
42:       </tr>
```

## 6.3 데이터를 조회하는 쿼리 실행

```
43: <%  
44:     }  
45:  
46: } catch(Exception e) {  
47:     e.printStackTrace();  
48: }  
49: %>  
50: </table>  
51:  
52: </body>  
53: </html>
```

번호	이름	국어	영어	수학	총점	평균
1	홍길동	50	60	70	180	60.00
2	이순신	65	75	85	225	75.00
3	강감찬	60	80	70	210	70.00

## 6.3 데이터를 조회하는 쿼리 실행

- 쿼리의 종류에 따른 데이터베이스 접근 코드의 형태 정리

구분		코드 형식
DB 접속		<pre>Connection conn = DriverManager.getConnection(     "jdbc:mariadb://서버주소:3306/사용할DB", "사용자ID", "비밀번호");</pre>
쿼리 문장 객체 준비		<pre>Statement stmt = conn.createStatement();</pre>
쿼리 실행	select 외의 쿼리	<pre>stmt.executeUpdate(쿼리);</pre>
	select 쿼리	<pre>ResultSet rs = stmt.executeQuery(쿼리); while (rs.next()) {     // rs.getInt(정수_필드)     // rs.getString(문자열_필드) }</pre>

## 6.2 데이터베이스의 내용을 변경하는 쿼리 실행 (교재 p.116~119)

- 추가적인 처리가 필요 없는 쿼리들을 실행하는 방법
  - create, drop, insert, delete, update와 같이 데이터베이스의 내용을 변경하는 쿼리  

```
Statement stmt = conn.createStatement();  
stmt.executeUpdate(쿼리);
```
  - 첫번째 줄은 쿼리 문장 객체를 stmt라는 이름으로 만든다.
  - 두 번째 줄에서는 이 쿼리 문장 객체를 이용해 원하는 쿼리를 실행한다.

## 6.2 데이터베이스의 내용을 변경하는 쿼리 실행

[예제 6-3] 테이블 생성 예제 (6-3.jsp)

```
1: <%@ page language="java" contentType="text/html; charset=UTF-8"  
2:    pageEncoding="UTF-8"%>  
3: <%@ page import="java.sql.*" %>  
4:  
5: <!DOCTYPE html>  
6: <html>  
7: <head>  
8:    <meta charset="UTF-8">  
9: </head>  
10: <body>  
11:  
12: <%  
13:    Class.forName("org.mariadb.jdbc.Driver");  
14:    try (  
15:        Connection conn = DriverManager.getConnection(  
16:            "jdbc:mariadb://localhost:3306/jspdb", "jsp", "1234");  
17:        Statement stmt = conn.createStatement();  
18:    ) {  
19:
```

## 6.2 데이터베이스의 내용을 변경하는 쿼리 실행

```
20:      String sql =
21:          "create table score (" +
22:          "    num int          primary key," +
23:          "    name varchar(20),      " +
24:          "    kor int,              " +
25:          "    eng int,              " +
26:          "    math int              " +
27:          ")";
28:
29:      stmt.executeUpdate(sql);
30:      out.println("성적 테이블 생성 성공 !");
31:
32:  } catch(Exception e) {
33:      e.printStackTrace();
34:  }
35: %>
36:
37: </body>
38: </html>
```

## 6.2 데이터베이스의 내용을 변경하는 쿼리 실행

[예제 6-4] 레코드 추가 예제 (6-4.jsp)

```
1: <%@ page language="java" contentType="text/html; charset=UTF-8"
2:    pageEncoding="UTF-8"%>
3: <%@ page import="java.sql.*" %>
4:
5: <!DOCTYPE html>
6: <html>
7: <head>
8:    <meta charset="UTF-8">
9: </head>
10: <body>
11:
12: <%
13:    Class.forName("org.mariadb.jdbc.Driver");
14:    try (
15:        Connection conn = DriverManager.getConnection(
16:            "jdbc:mariadb://localhost:3306/jspdb", "jsp", "1234");
17:        Statement stmt = conn.createStatement();
18:    ) {
19:
```



# [과제]

- 팀별로 설계한 **게시판** CRUD(Create, Read, Update, Delete) 구현하기
  - 팀원 모두가 자신의 깃허브에도 올려둘 것
- 팀별로 설계한 **회원관리** CRUD 구현하기

마감일은 항상 과제가 나간 날에서 일주일 후 밤 12:00까지 각 자의 깃허브에 올려둘 것!!!  
마감 기한을 지나면 하루에 20%씩 점수 깎임