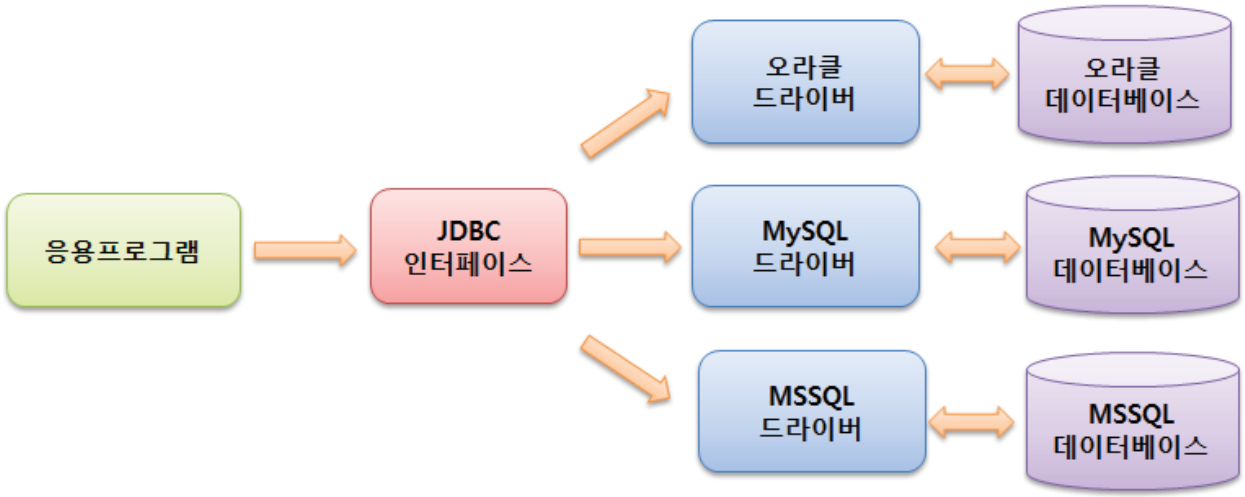
프로그래밍 언어 활용 강의안
(java ⟷ database 연동)

1. JDBC의 개요

JDBC(Java DataBase Connectivity)

- 자바/JSP 프로그램 내에서 데이터베이스와 관련된 작업을 처리할 수 있도록 도와주는 자바 표준 인터페이스
- 관계형 데이터베이스 시스템에 접근하여 SQL 문을 실행하기 위한 자바 API 또는 자바 라이브러리
- JDBC API를 사용하면 DBMS의 종류에 상관없이 데이터베이스 작업을 처리할 수 있음



1. JDBC의 개요

❖ JDBC를 사용한 JAVA와 데이터베이스의 연동

- ❶ java.sql.* 패키지 import
- ❷ JDBC 드라이버 로딩,
- ❸ 데이터베이스 접속을 위한 Connection 객체 생성,
- ❹ 쿼리문을 실행하기 위한 Statement/PreparedStatement/CallableStatement객체 생성,
- ❺ 쿼리 실행,
- ❻ 쿼리 실행의 결과 값(int, ResultSet) 사용,
- ❼ 사용된 객체 (ResultSet,Statement/PreparedStatement/CallableStatement, Connection) 종료

2. JDBC 드라이버 로딩 및 DBMS 접속

❖ 2.1 JDBC 드라이버 로딩하기

- JDBC 드라이버 로딩 단계에서는 드라이버 인터페이스를 구현하는 작업
- Class.forName( ) 메소드를 이용하여 JDBC 드라이버를 로딩

```
Class.forName(String className);
```

- JDBC 드라이버가 로딩되면 자동으로 객체가 생성
- DriverManager 클래스에 등록

[MySQL 드라이버 로딩 예]

```
<%
    try{
       Class.forName("com.mysql.jdbc.Driver");
    }catch(SQLException ex){
       //예외 발생 처리
    }
%>
```

- JDBC 드라이버 로딩은 프로그램 수행 시 한 번만 필요

2. JDBC 드라이버 로딩 및 DBMS 접속

❖ 2.2 MySQL JDBC 드라이버 Class

```
package com.mysql.cj.jdbc;

import java.sql.DriverManager;

public class Driver
  extends NonRegisteringDriver
  implements java.sql.Driver
{
  public Driver()
    throws SQLException
  {}

  static
  {
    try
    {
      DriverManager.registerDriver(new Driver());
    }
    catch (SQLException E)
    {
      throw new RuntimeException("Can't register driver!");
    }
  }
}
```

2. JDBC 드라이버 로딩 및 DBMS 접속

❖ 2.3 Connection 객체 생성하기

- JDBC 드라이버에서 데이터베이스와 연결된 커넥션을 가져오기 위해 DriverManager 클래스의 getConnection( ) 메소드를 사용
- DriverManager 클래스로 Connection 객체를 생성할 때 JDBC 드라이버를 검색하고, 검색된 드라이버를 이용하여 Connection 객체를 생성한 후 이를 반환

```
static Connection getConnection(String url)
static Connection getConnection(String url, String user, String password)
static Connection getConnection(String url, Properties info)
```

[Connection 객체 생성 예: getConnection(String url) 메소드 사용]

```
<%
    Connection conn = null;
    try {
       Class.forName("com.mysql.jdbc.Driver");
       conn = DriverManager.getConnection(
       "jdbc:mysql://localhost:3306/JSPBookDB?user=root&password=12341");
    } catch (SQLException ex) {
       //예외 발생 처리
    }
%>
```

2. JDBC 드라이버 로딩 및 DBMS 접속

[Connection 객체 생성 예: getConnection(String url, String user, String password) 메소드 사용]

```
<%
    Connection conn = null;
    try {
       Class.forName("com.mysql.jdbc.Driver");
       conn = DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/JSPBookDB", "root", "1234");
    } catch (SQLException ex) {
       //예외 발생 처리
    }
%>
```

[Connection 객체 생성 예: getConnection(String url, Properties info) 메소드 사용]

```
<%
    Connection conn = null;
    try {

       Class.forName("com.mysql.jdbc.Driver");
       Properties props = new Properties();
       props.put("user", "root");
       props.put("password", "1234");
       conn=DriverManager.getConnection(
           "jdbc:mysql://localhost:3306/JSPBookDB", props);
    } catch (SQLException ex) {
       //예외 발생 처리
    }
%>
```

2. JDBC 드라이버 로딩 및 DBMS 접속

2.4 데이터베이스 연결 닫기

- 데이터베이스 연결이 더 이상 필요하지 않으면 데이터베이스와 JDBC 리소스가 자동으로 닫힐 때까지 대기하는 것이 아니라 close( ) 메소드로 생성한 Connection 객체를 해제
- 일반적으로 데이터베이스 리소스를 사용하지 않기 위해 사용을 끝내자마자 리소스를 해제하는 것이 좋음

```
void close() throws SQLException
```

```
<%
    Connection conn = null;
    try{
       //JDBC 드라이버 로딩
       //Connection 객체 생성
    } catch(SQLException e){
       //예외 발생 처리
    } finally{
       if (conn != null) conn.close();
    }
%>
```

3. 데이터베이스 쿼리 실행

❖ 3.1 Statement 객체로 데이터 접근하기

- Statement 객체
  - 정적인 쿼리에 사용
  - 하나의 쿼리를 사용하고 나면 더는 사용할 수 없음
  - 하나의 쿼리를 끝내면 close( )를 사용하여 객체를 즉시 해제해야 함
  - close( )를 사용하여 객체를 즉시 해제하지 않으면 무시할 수 없는 공간이 필요하며 페이지가 다른 작업을 수행하는 동안 멈추지 않기 때문
  - 복잡하지 않은 간단한 쿼리문을 사용하는 경우에 좋음

```
Statement createStatement() throws SQLException
```

표 16-1 Statement 객체의 메소드 종류

메소드	반환 유형	설명
executeQuery(String sql)	ResultSet	SELECT 문을 실행할 때 사용합니다(ResultSet 객체 반환).
executeUpdate(String sql)	int	삽입, 수정, 삭제와 관련된 SQL 문 실행에 사용합니다.
close( ):	void	Statement 객체를 반환할 때 사용합니다.

3. 데이터베이스 쿼리 실행

❖ 3.1 Statement 객체로 데이터 접근하기

- executeQuery() 메소드로 데이터 조회하기

```
ResultSet executeQuery(String sql) throws SQLException
```

[executeQuery() 메소드 사용 예: SELECT 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    Statement stmt = conn.createStatement();
    String sql = "SELECT * FROM Member WHERE id ='1'";
    ResultSet rs = stmt.executeQuery(sql);
    stmt.close();
%>
```

3. 데이터베이스 쿼리 실행

❖ 3.1 Statement 객체로 데이터 접근하기

- executeUpdate() 메소드로 데이터 삽입, 수정, 삭제하기
  - executeUpdate( ) 메소드는 INSERT, UPDATE, SELECT 쿼리문을 통해 데이터를 삽입, 수정, 삭제하는 데 사용

```
int executeUpdate(String sql) throws SQLException
```

[executeUpdate() 메소드 사용 예(삽입): INSERT 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    Statement stmt = conn.createStatement();
    String sql = "INSERT INTO Member(id, name, passwd) VALUES ('1', '홍길순',
    '1234')";
    int rs = stmt.executeUpdate(sql);
%>
```

3. 데이터베이스 쿼리 실행

[executeUpdate() 메소드 사용 예(수정): UPDATE 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    Statement stmt = conn.createStatement();
    String sql = "UPDATE Member SET name = '관리자' WHERE id = '1'";
    int rs = stmt.executeUpdate(sql);
%>
```

[executeUpdate() 메소드 사용 예(삭제): DELETE 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    Statement stmt = conn.createStatement();
    String sql = "DELETE FROM Member WHERE id = '1'";
    int rs = stmt.executeUpdate(sql);
%>
```

3. 데이터베이스 쿼리 실행

❖ 3.2 PreparedStatement 객체로 데이터 접근하기

- PreparedStatement 객체
  - 동적인 쿼리에 사용
  - Prepared Statement 객체는 하나의 객체로 여러 번의 쿼리를 실행할 수 있으며, 동일한 쿼리문을 특정 값만 바꾸어서 여러 번 실행해야 할 때, 매개변수가 많아서 쿼리문을 정리해야 할 때 유용

```
PreparedStatement prepareStatement(String sql) throws SQLException
```

3. 데이터베이스 쿼리 실행

표 16-2 setXxx() 메소드의 종류

메소드	반환 유형	설명
setString(int parameterIndex, String x)	void	필드 유형이 문자열인 경우
setInt(int parameterIndex, int x)	void	필드 유형이 정수형인 경우
setLong(int parameterIndex, long x)	void	필드 유형이 정수형인 경우
setDouble(int parameterIndex, double x)	void	필드 유형이 실수형인 경우
setFloat(int parameterIndex, float x)	void	필드 유형이 실수형인 경우
setObject(int parameterIndex, Object x)	void	필드 유형이 객체형인 경우
setDate(int parameterIndex, Date x)	void	필드 유형이 날짜형인 경우
setTimestamp(int parameterIndex, Timestamp x)	void	필드 유형이 시간형인 경우

표 16-3 PreparedStatement 객체의 메소드 종류

메소드	반환 유형	설명
executeQuery( )	ResultSet	SELECT 문을 실행할 때 사용합니다(ResultSet 객체 반환).
executeUpdate( )	int	삽입, 수정, 삭제와 관련된 SQL 문 실행에 사용합니다.
close( ):	void	Statement 객체를 반환할 때 사용합니다.

3. 데이터베이스 쿼리 실행

❖ 3.2 PreparedStatement 객체로 데이터 접근하기

- executeQuery() 메소드로 데이터 조회하기
  - executeQuery( ) 메소드는 동적인 SELECT 쿼리문을 통해 데이터를 검색하는 데 사용

```
int executeQuery() throws SQLException
```

[executeQuery() 메소드 사용 예: SELECT 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    String sql = "SELECT * FROM Member WHERE id =? ";
    PreparedStatement  pstmt = conn.prepareStatement(sql);
    pstmt.setString(1,"1");
    ResultSet rs = pstmt.executeQuery(sql);
    …(생략)…
    pstmt .close();
%>
```

3. 데이터베이스 쿼리 실행

❖ 3.2 PreparedStatement 객체로 데이터 접근하기

- executeUpdate() 메소드로 데이터 삽입, 수정, 삭제하기
  - executeUpdate( ) 메소드는 INSERT, UPDATE, SELECT 쿼리문을 통해 데이터를 삽입, 수정, 삭제하는 데 사용

```
int executeUpdate() throws SQLException
```

[executeUpdate() 메소드 사용 예(삽입): INSERT 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    String sql = "INSERT INTO Member(id, name, passwd) VALUES (?,?,?)";
    PreparedStatement  pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "1");
    pstmt.setString(2, "홍길순");
    pstmt.setString(3, "1234");
    pstmt.executeUpdate();
    …(생략)…
    pstmt .close();
%>
```

3. 데이터베이스 쿼리 실행

[executeUpdate() 메소드 사용 예(수정): UPDATE 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    String sql = "UPDATE Member SET name = ?' WHERE id = ?";
    PreparedStatement  pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "1");
    pstmt.setString(2, "관리자");
    pstmt.executeUpdate();
    …(생략)…
    pstmt .close();
%>
```

[executeUpdate() 메소드 사용 예(삭제): DELETE 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    String sql = "DELETE FROM Member WHERE id = ?";
    PreparedStatement  pstmt = conn.prepareStatement(sql);
    pstmt.setString(1, "1");
    pstmt.executeUpdate();
    …(생략)…
    pstmt .close();
%>
```

4. 쿼리문 실행 결과 값 가져오기

❖ ResultSet 객체

- Statement 또는 PreparedStatement 객체로 SELECT 문을 사용하여 얻어온 레코드 값을 테이블 형태로 가진 객체

[Statement 객체를 사용하는 경우]

```
ResultSet executeQuery(String sql) throws SQLException
```

[PreparedStatement 객체를 사용하는 경우]

```
ResultSet executeQuery() throws SQLException
```

4. 쿼리문 실행 결과 값 가져오기

표 16-4 ResultSet 객체의 메소드 종류

메소드	반환 유형	설명
getXxx(int ColumnIndex)	XXX	설정한 ColumnIndex(필드 순번)의 필드 값을 설정한 XXX 형으로 가져옵니다.
getXxx(String ColumnName)	XXX	설정한 ColumnName(필드 순번)의 필드 값을 설정한 XXX 형으로 가져옵니다.
absolute(int row)	boolean	설정한 row 행으로 커서를 이동합니다.
beforeFirst( )	void	첫 번째 행의 이전으로 커서를 이동합니다.
afterLast( )	void	마지막 행의 다음으로 커서를 이동합니다.
first( )	void	첫 번째 행으로 커서를 이동합니다.
last( )	void	마지막 행으로 커서를 이동합니다.
next( )	boolean	다음 행으로 커서를 이동합니다.
previous( )	boolean	현재 행의 이전 행으로 커서를 이동합니다.
close( ):	void	ResultSet 객체를 반환할 때 사용합니다.

4. 쿼리문 실행 결과 값 가져오기

[executeQuery() 메소드 사용 예: SELECT 쿼리문]

```
<%
    Connection conn = null;
    …(생략)…
    Statement stmt = conn.createStatement();
    String sql = "SELECT * FROM Member WHERE id ='1";
    ResultSet rs = stmt.executeQuery(sql);

    while (rs.next()) {
       out.println(rs.getString(2) + ", " + rs.getString(3) + "<br/>");
    }
    rs.close();
    stmt.close();
%>
```

Thank You