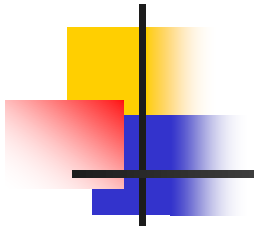
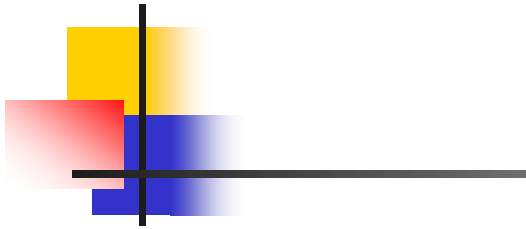
데이터베이스 프로그래밍

(소프트웨어 개발 트랙)

* 정보통신산업진흥원과 한국정보처리학회의 자료를 수정 보완함.



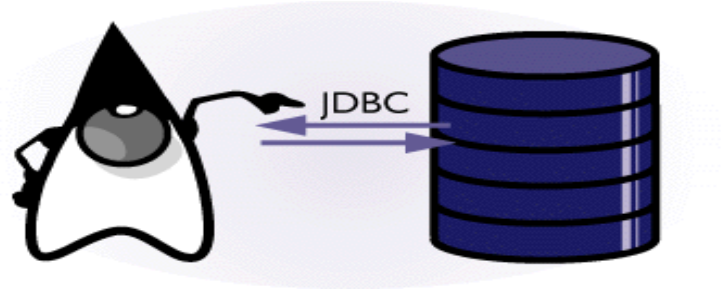
제 3부
데이터베이스
어플리케이션 프로그래밍(1)



JDBC 개념

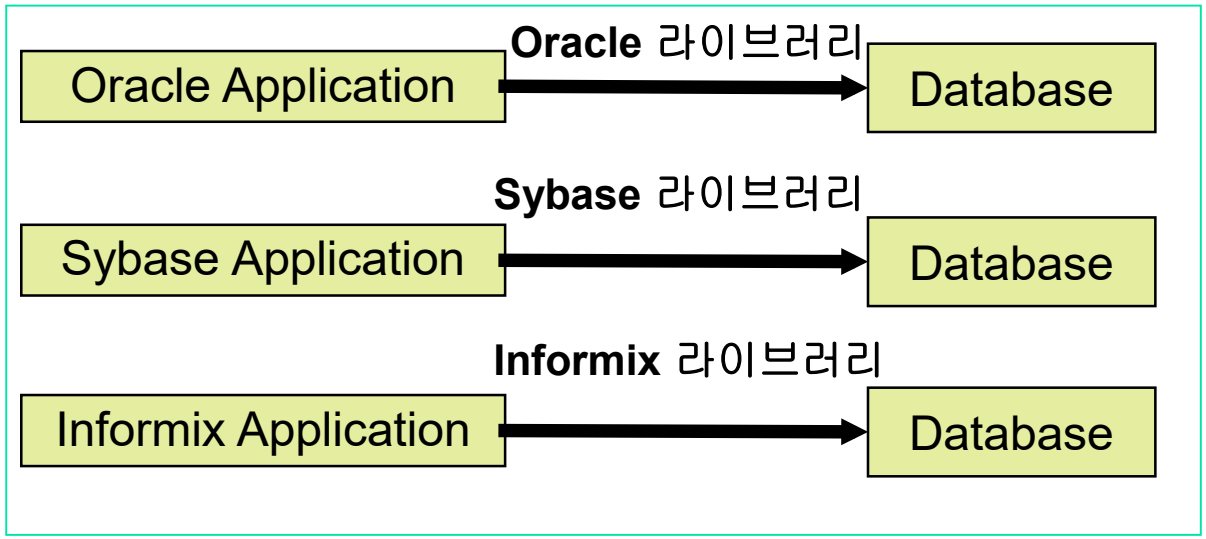
JDBC란

- JDBC(Java DataBase Connectivity)의 정의
  - 자바를 이용한 데이터베이스 접속과 SQL 문장의 실행, 그리고 실행 결과로 얻어진 데이터의 핸들링을 제공하는 방법과 절차에 관한 규약
  - 자바 프로그램 내에서 SQL 문을 실행하기 위한 자바 API
  - SQL과 프로그래밍 언어의 통합 접근 중 한 형태
- 개발자를 위한 표준 인터페이스인 JDBC API와 데이터베이스 벤더, 또는 기타 써드파티에서 제공하는 드라이버(driver)



JDBC 사용이유(1)

- 데이터베이스를 제공하는 제작사에서 클라이언트 어플리케이션을 독립시키기 위해서 만들어짐
- 다음과 같은 경우를 생각해 보자.



개발자는 각각의 데이터베이스에 종속될 라이브러리를 숙지하고 있어야 함

JDBC 사용이유(2)

- 하나의 어플리케이션에서 데이터베이스에 상관없이 단일화된 인터페이스(API)를 통해서 데이터를 이용할 수 있게 하기 위해 사용
- 각각의 데이터베이스에 종속되는 해당 어플리케이션을 따로 만들 필요가 없음

한번 작성하여 컴파일하면, 어디서나 사용할 수 있다.

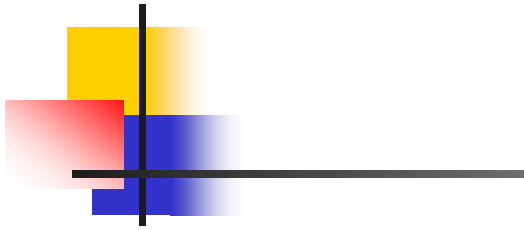
JDBC 드라이버의 교체로 같은 코드의 내용을
서로 다른 데이터베이스에서 활용 가능

JDBC 드라이버(1)

- 드라이버 벤더들은 무엇을 제공하는가?
  - 각 데이터베이스 벤더들은 JDBC API(interface) 클래스에 대한 구현(implementation)을 제공해야 하며, 이것을 드라이버(driver)라고 함
  - 프로그래머가 사용하는 JDBC API의 실제 오브젝트들이 바로 각 벤더들이 제공하는 드라이버에 있는 클래스의 오브젝트들이 되는 것
  - JDK1.2 이상의 버전에 들어있는 JDBC 관련 API는 인터페이스이며, 그 중에서 몇몇만이 클래스로 구현되어 있고, 구현된 클래스 중에서 DriverManager 클래스는 각각의 JDBC 드라이버를 연결하는 역할을 함

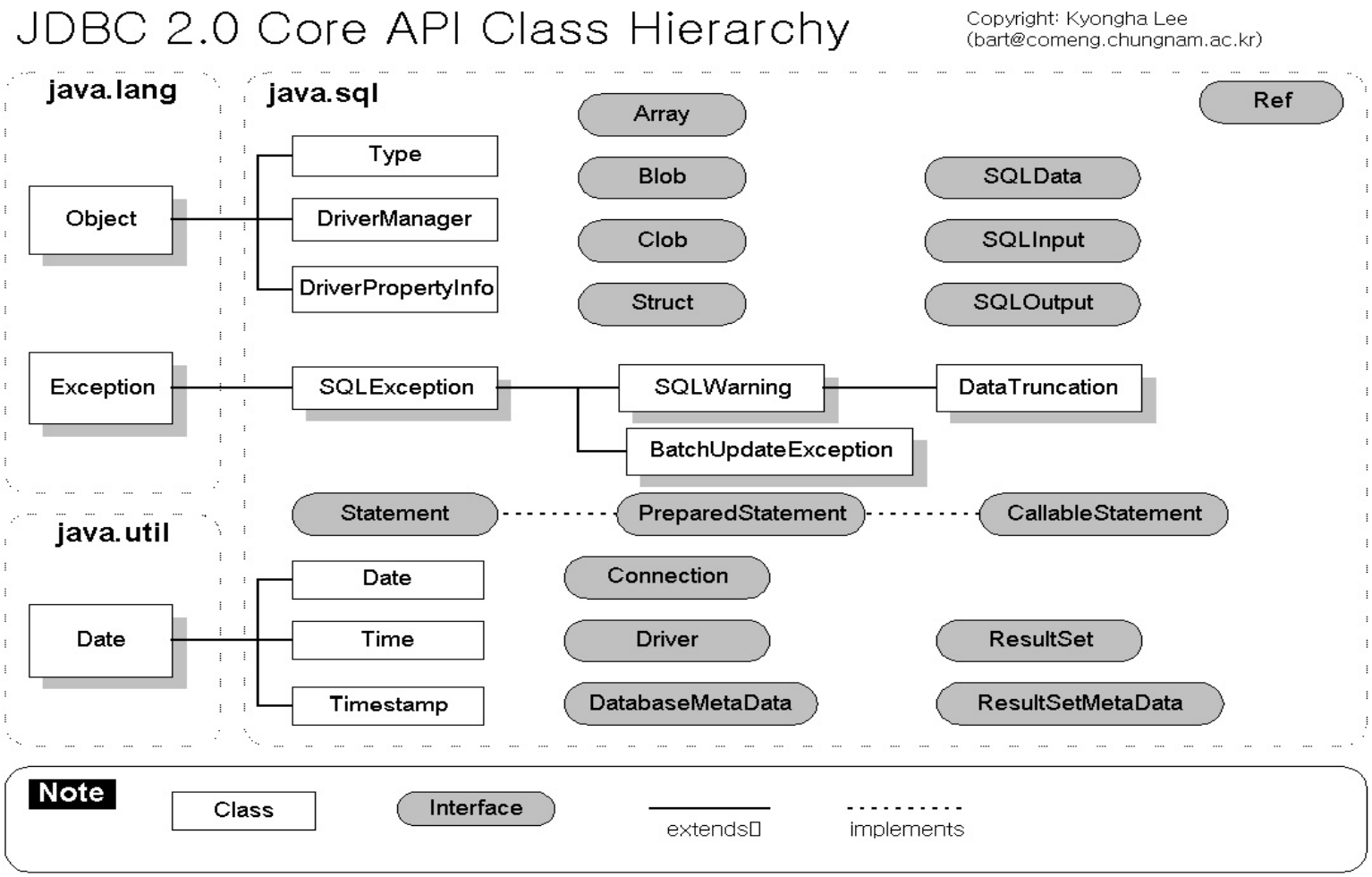
JDBC 드라이버(2)

- DriverManager는 클래스이면서, 다른 API는 인터페이스로 구현되는 이유는?
  - 데이터베이스 제품이 각 벤더마다 다르게 구성되어 있기 때문
  - JDBC 드라이버 :
    데이터베이스를 접속하고, 핸들링하기 위한 API는 인터페이스로 정해 놓고, 그 인터페이스를 각각의 데이터베이스 벤더들이 맞게 구현한 것
  - 데이터베이스에 맞는 JDBC 드라이버를 사용해야 함



JDBC를 이용한
데이터베이스 프로그래밍 개요

java.sql package 구조(1)

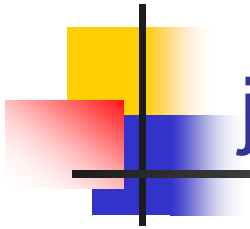


java.sql package 구조(2)

- JDBC Interfaces
  - Driver
  - Connection
  - Statement
  - ResultSet
  - PreparedStatement
  - CallableStatement
  - ResultSetMetaData
  - DatabaseMetaData
  - *Array, Blob, Clob, Ref*
  - *SQLData, SQLInput, SQLOutput, Struct*

java.sql package 구조(3)

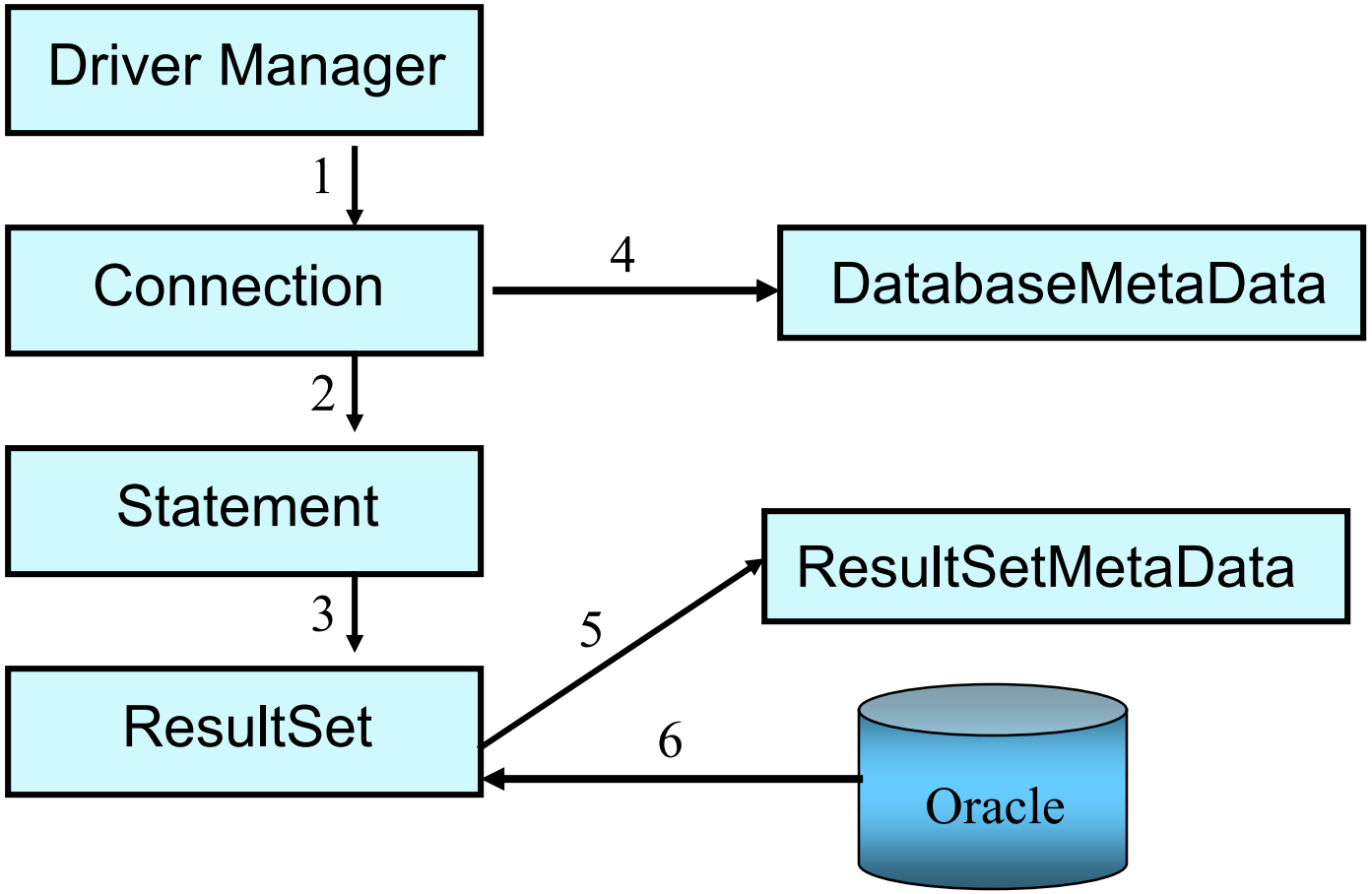
- JDBC Classes
  - DriverManager
  - Types
  - DriverPropertyInfo
  - Date
  - Time
  - Timestamp
- JDBC Exceptions
  - SQLException
  - SQLWarning
  - DataTruncation
  - *BatchUpdateException*



java.sql package 주요 요소

- java.sql.DriverManager
  - 드라이버 리스트를 저장하고 요청된 URL에 해당하는 드라이버를 어플리케이션에 제공
- java.sql.Connection
  - 하나의 논리적 데이터베이스 트랜잭션을 나타내는 클래스
- java.sql.Statement
  - SQL 구문의 실행
- java.sql.ResultSet
  - 데이터베이스 질의에 의해 얻어진 데이터의 행을 대표하는 오브젝트

JDBC를 이용한 연동단계(1)



JDBC를 이용한 연동단계(2)

- Step 1 : Driver Loading
  - Class.forName("driver-name");
- Step 2 : Obtain Connection
  - Connection con = DriverManager.getConnection("url");
- Step 3 : Create Statement
  - Statement stmt = con.createStatement();
- Step 4 : Execute Query
  - stmt.execute("query");
  - stmt.executeQuery("query");
  - stmt.executeUpdate("query");

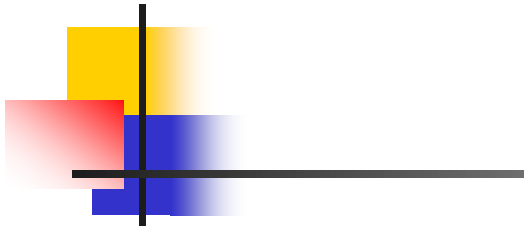
any SQL

SELECT

INSERT, UPDATE, DELETE

JDBC를 이용한 프로그램 사용 예

```
import java.sql.*;
                                                    ……
Connection myConn = null;
Statement stmt = null, mySQL = null;

String dburl  = “jdbc:oracle:thin:@210.94.199.20:1521:dblab”;
String user = “사용자 계정”;     String passwd=“패스워드”; // 비밀번호
String dbdriver = "oracle.jdbc.driver.OracleDriver”;

Class.forName(dbdriver);
myConn =  DriverManager.getConnection (dburl, user , passwd);
stmt = myConn.createStatement();

ResultSet myResultSet = stmt.executeQuery(“SELECT ename FROM emp”);

while ( myResultSet.next() )
      String ename = myResultSet.getString(“ename”);
stmt.close();
myConn.close();
```

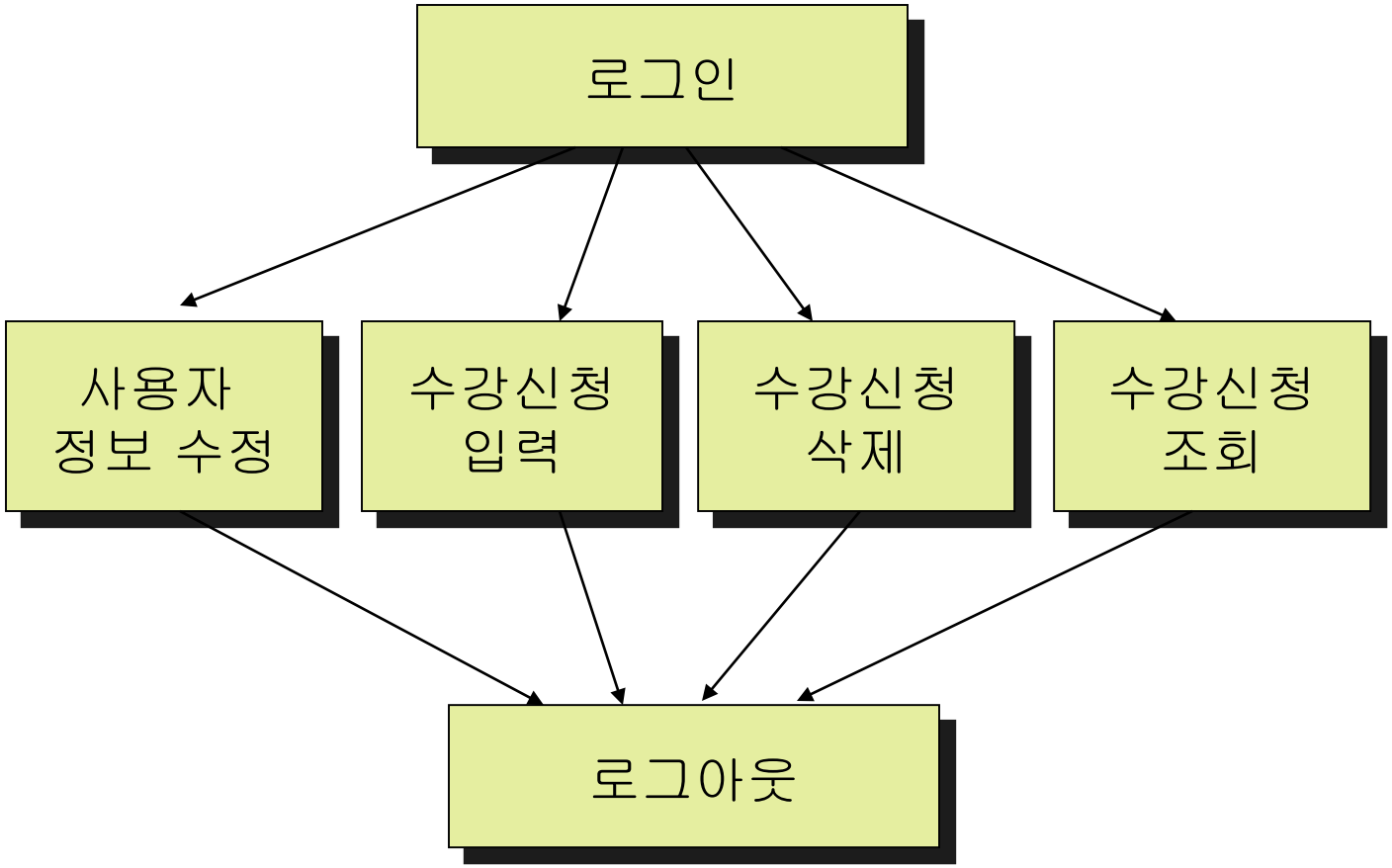


JDBC와 JSP를 이용한 데이터베이스 어플리케이션 프로그래밍 개요

수업을 통해 완성될 프로그램 개요

- 데이터베이스와 고급 SQL 활용
- 저장 프로시저 프로그래밍 활용
- JSP와 JDBC를 이용하여 수강신청 시스템의 일부를 프로그래밍

수강신청 어플리케이션 구성도(리뷰)



수강신청 어플리케이션 기능 설명(1)

- 로그인
  - 학번과 패스워드를 확인하여 수강신청 시스템을 사용할 수 있도록 함
  - 로그인이 정상적으로 이루어지지 않는 경우는 기타 기능을 사용할 수 없음
- 로그아웃
  - 수강신청 시스템 사용 종료
- 사용자 정보 수정
  - 학생의 주소와 패스워드 수정
  - 학생의 나머지 정보는 학생이 직접 수정이 불가능

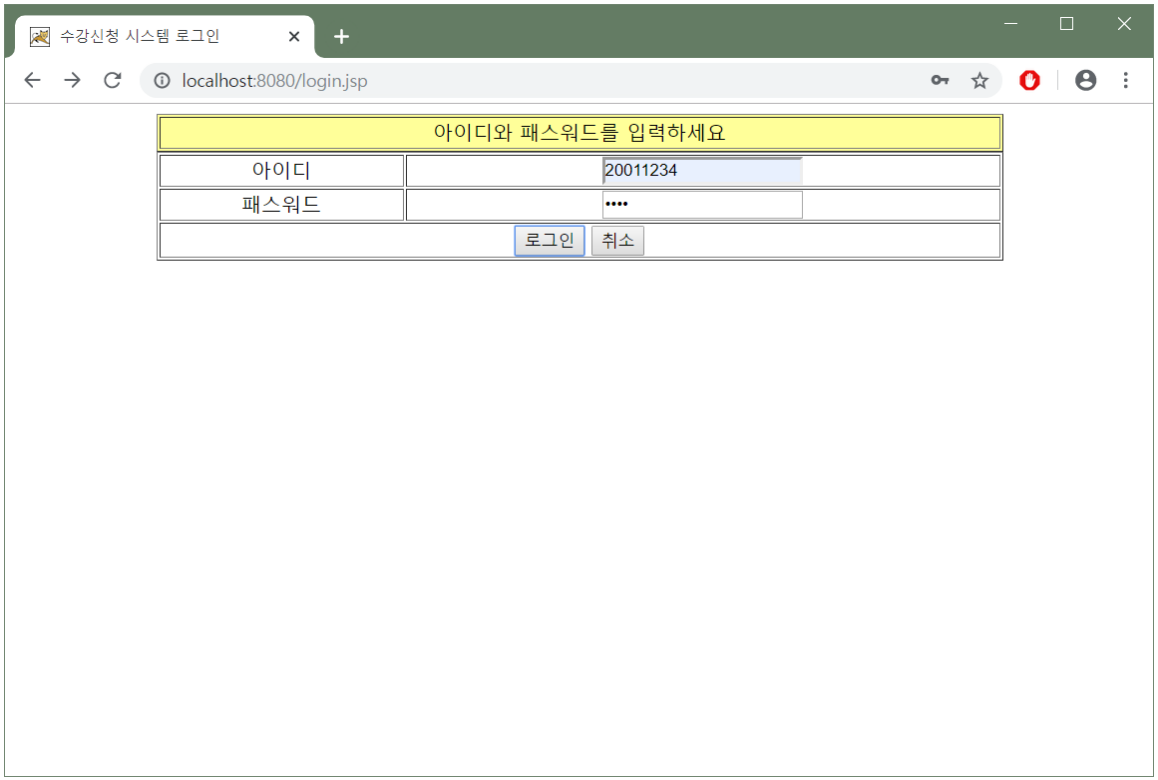
수강신청 어플리케이션 기능 설명(2)

- 수강신청 입력
  - 수강신청 가능 과목을 보여주고, 원하는 과목을 선택하면 해당 과목에 대한 수강신청 등록
- 수강신청 삭제
  - 현재 수강신청 과목으로 등록된 과목을 보여주고, 원하는 과목을 선택하면 해당 과목에 대한 수강신청 삭제
- 수강신청 조회
  - 현재 수강신청 과목으로 등록된 과목을 보여주고, 이들 과목에 대한 총 신청 과목수와 총 신청 학점을 보여줌
  - 현재 수강신청 가능한 년도와 학기 이외에도 원하는 년도와 학기를 지정하여 검색 가능

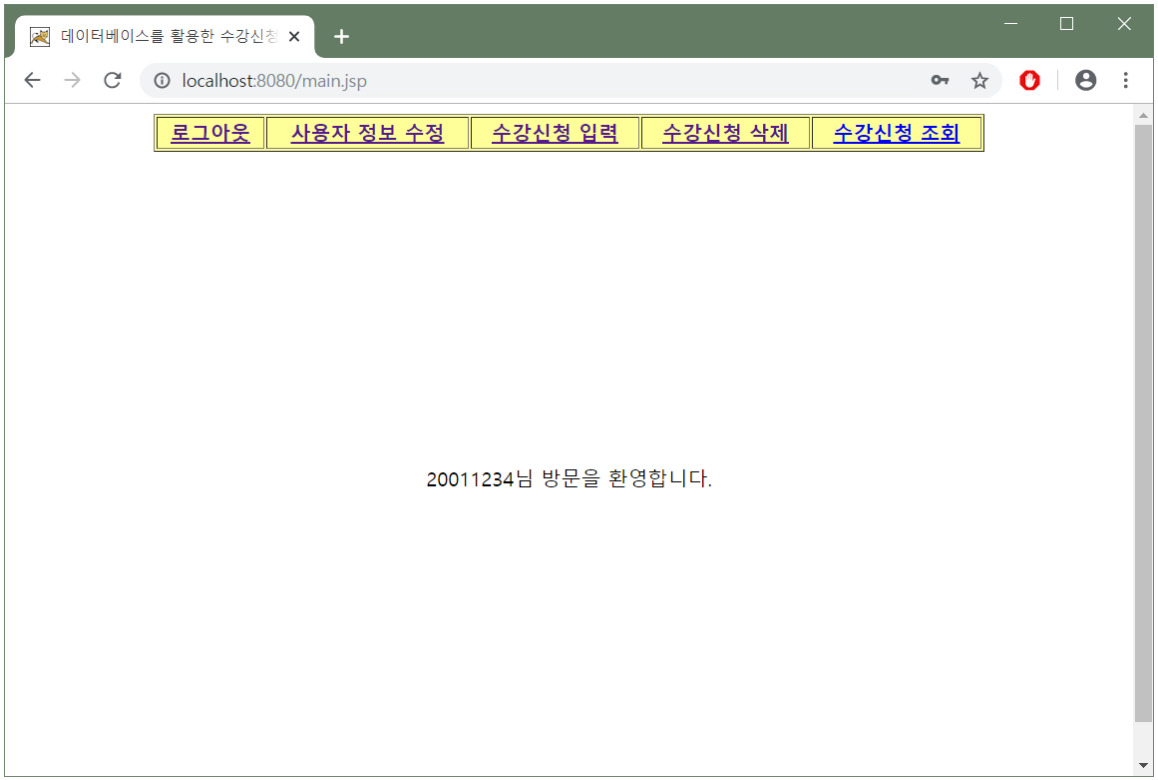
메인 화면


데이터베이스를 활용한 수강신청
localhost:8080/main.jsp
로그인
사용자 정보 수정
수강신청 입력
수강신청 삭제
수강신청 조회
로그인한 후 사용하세요.

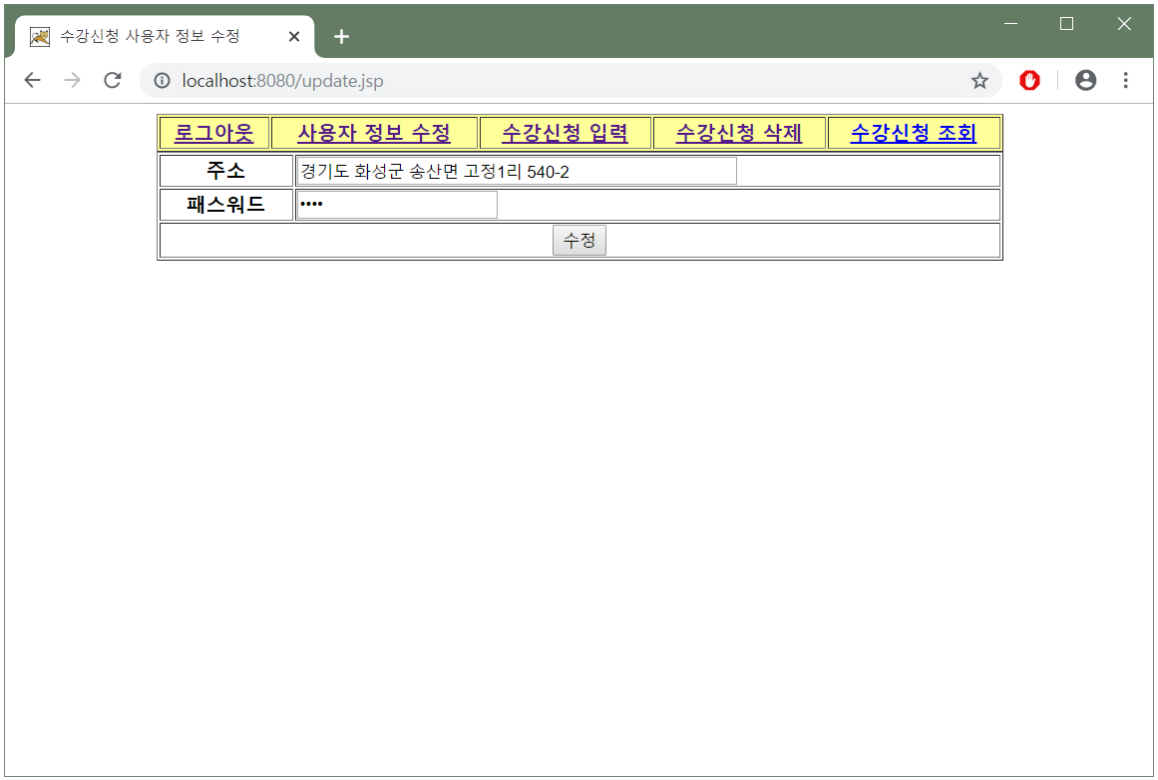
로그인 화면


수강신청 시스템 로그인
localhost:8080/login.jsp
아이디와 패스워드를 입력하세요
아이디
20011234
패스워드
로그인
취소

로그인 후의 화면


데이터베이스를 활용한 수강신청
localhost:8080/main.jsp
로그아웃
사용자 정보 수정
수강신청 입력
수강신청 삭제
수강신청 조회
20011234님 방문을 환영합니다.

사용자 정보 수정 화면


수강신청 사용자 정보 수정
localhost:8080/update.jsp
로그아웃
사용자 정보 수정
수강신청 입력
수강신청 삭제
수강신청 조회
주소
경기도 화성군 송산면 고정1리 540-2
패스워드
수정

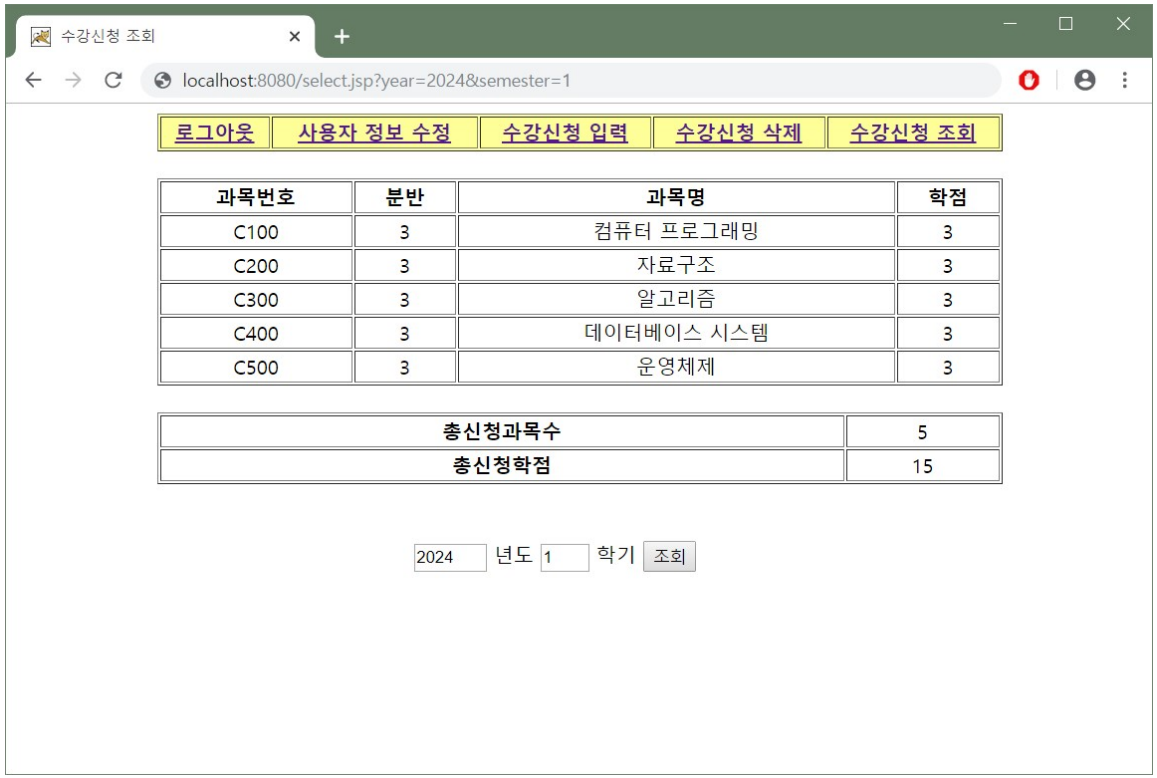
수강신청 입력 화면

과목번호	분반	과목명	학점	수강신청
C800	3	데이터베이스 프로그래밍	3	신청
C900	3	객체지향 윈도우즈 프로그래밍	3	신청
M100	3	멀티미디어 개론	3	신청
M200	3	선형대수	3	신청
M300	3	그래픽 활용	3	신청
M400	3	윈도우즈 프로그래밍	3	신청
M700	3	게임 프로그래밍	3	신청

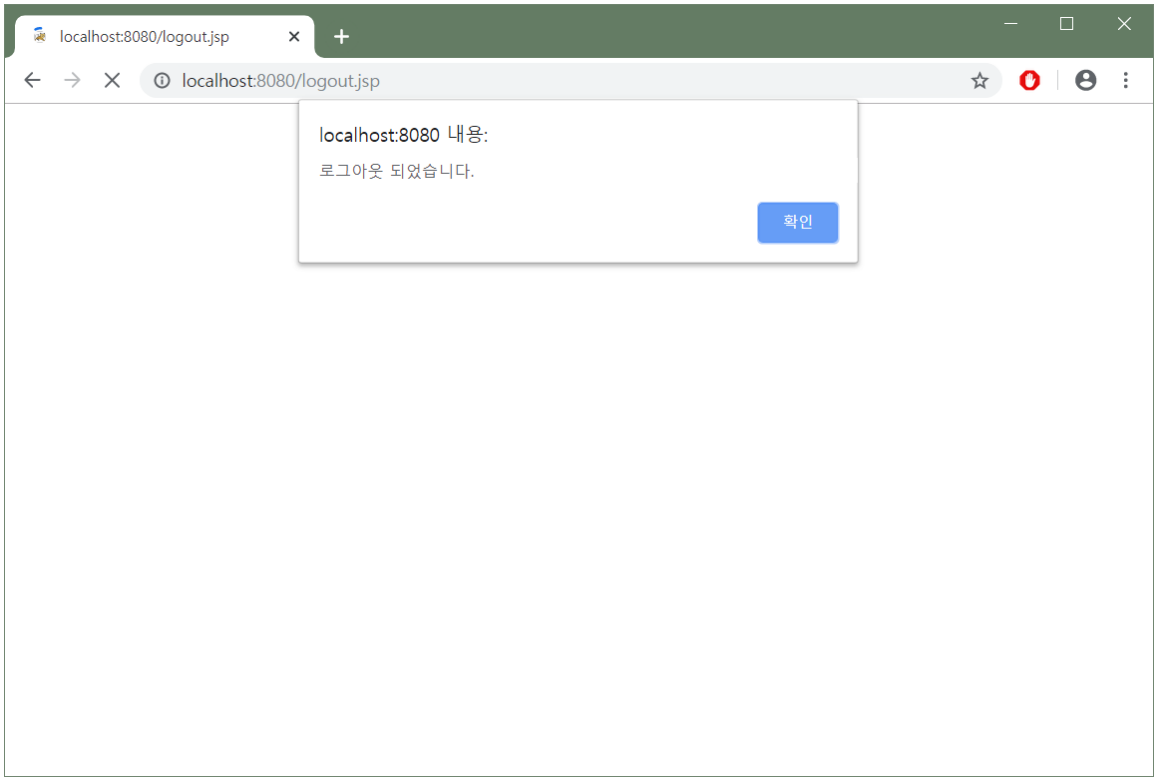
수강신청 삭제 화면


수강신청 삭제
localhost:8080/delete.jsp
로그아웃
사용자 정보 수정
수강신청 입력
수강신청 삭제
수강신청 조회
과목번호 분반 과목명 학점 수강신청
C100 3 컴퓨터 프로그래밍 3 삭제
C200 3 자료구조 3 삭제
C300 3 알고리즘 3 삭제
C400 3 데이터베이스 시스템 3 삭제
C500 3 운영체제 3 삭제

수강신청 조회 화면


수강신청 조회
localhost:8080/select.jsp?year=2024&semester=1
로그아웃
사용자 정보 수정
수강신청 입력
수강신청 삭제
수강신청 조회
과목번호
분반
과목명
학점
C100
3
컴퓨터 프로그래밍
3
C200
3
자료구조
3
C300
3
알고리즘
3
C400
3
데이터베이스 시스템
3
C500
3
운영체제
3
총신청과목수
5
총신청학점
15
2024
년도
1
학기
조회

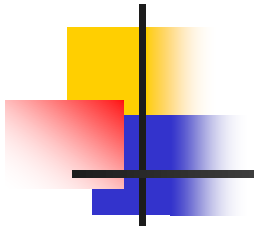
로그아웃 화면


localhost:8080/logout.jsp
localhost:8080/logout.jsp
localhost:8080 내용:
로그아웃 되었습니다.
확인

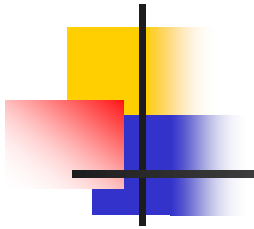
데이터베이스 프로그래밍

(소프트웨어 개발 트랙)

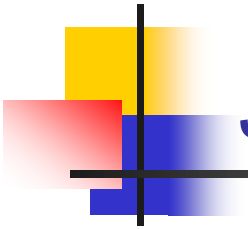
* 정보통신산업진흥원과 한국정보처리학회의 자료를 수정 보완함.



제 3부
데이터베이스
어플리케이션 프로그래밍(2)

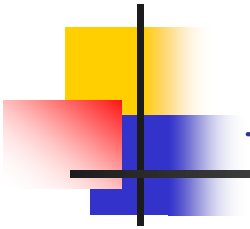


데이터베이스 연결



JDBC와 데이터베이스 연결

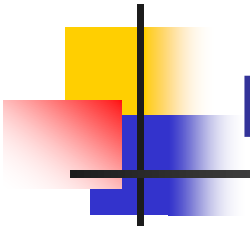
- JDBC를 이용하여 데이터베이스에 연결하는 방법
  - DriverManager
    - JDBC 1.0 버전부터 나온 데이터베이스 연결 방법
    - URL을 이용하여 특정 드라이버를 로딩하여 사용
  - DataSource
    - JDBC 2.0 Optional Package에서 등장한 방법
    - 어플리케이션에 투명한 개발 가능(연결 상세사항을 캡슐화)
      - ConnectionPoolDataSource



드라이버

- Driver 클래스
  - 모든 드라이버는 java.sql.Driver를 implement 한 Driver 클래스를 제공해야 함
  - DriverManager 클래스에 등록되며 관리됨

- Driver 클래스 예
  - Oracle : oracle.jdbc.driver.OracleDriver
  - MySQL : org.gjt.mm.mysql.Driver
  - Mini-SQL : com.imaginary.sql.msql.MsqlDriver
  - ODBC : sun.jdbc.odbc.JdbcOdbcDriver



DriverManager(1)

- DriverManager 클래스
  - 데이터베이스 드라이버(MySQL, Oracle 등)를 관리
  - Connection을 요구하는 URL을 처리할 수 있는 드라이버를 찾아서 Connection을 획득

DriverManager(2)

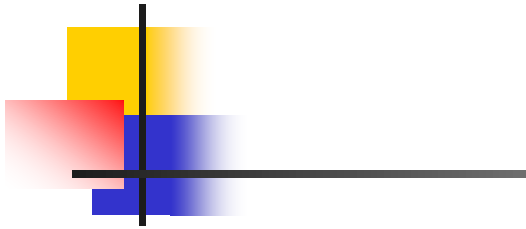
- DriverManager 클래스 중요 메소드
  - public static void registerDriver (Driver driver) throws SQLException
    - 사용 가능한 드라이버 세트에 드라이버를 등록하고, 드라이버가 로딩될 때 호출
  - public static Connection getConnection (String url, String user, String pw) throws SQLExcpetion
    - 클라이언트가 Connection을 구할 때 호출
    - DriverManager는 드라이버 세트에서 사용되는 해당 드라이버를 찾은 후에 그 드라이버의 Connection 객체를 리턴

DriverManager와 데이터베이스 연결(1)

- DriverManager 를 이용하여 데이터베이스에 연결하는 순서
  - 1) JDBC 드라이버 로딩
    - Class.forName("JDBC 드라이버명"); 또는
    - DriverManager.registerDriver("JDBC 드라이버명");
  - 2) JDBC 드라이버를 이용한 데이터베이스 연결
    - Connection con = DriverManager.getConnection ("JDBC URL", "사용자 계정", "비밀번호");
      - JDBC URL = jdbc:<subprotocol>:<subname>

DriverManager와 데이터베이스연결(2)

- 예 : 오라클의 경우
  - 1) JDBC 드라이버 로딩
    - Class.forName("oracle.jdbc.driver.OracleDriver");
    - 또는
      DriverManager.registerDriver(neworacle.jdbc.driver.OracleDriver());
  - 2) JDBC 드라이버를 이용한 데이터베이스 연결
    - thin 드라이버의 경우
    - Connection con = DriverManager.getConnection ("jdbc:oracle:thin:@serverName:1521:SID"," 사용자 계정", "비밀번호");
      - serverName : 예) 210.94.199.20
      - SID : 예) dblab



SQL 처리

Statement 객체로 SQL 명령문 실행(1)

- Statement 역할 : SQL 구문의 실행
- 생성
  - 전진과 읽기만 가능한 ResultSet 생성
    - Statement stmt = con.createStatement();
  - 스크롤과 수정 가능한 ResultSet 생성
    - createStatment 내에 옵션 지정
      - 스크롤 옵션
        - TYPE_FORWARD_ONLY(디폴트)
        - TYPE_SCROLL_INSENSITIVE
        - TYPE_SCROLL_SENSITIVE
      - 동시성 옵션
        - CONCUR_READ_ONLY
        - CONCUR_UPDATABLE :

Statement 객체로 SQL 명령문 실행(2)

- 실행
  - ResultSet executeQuery(String sql)
    - SQL문 중 SELECT문만 가능
    - SELECT를 통해 추출한 값들을 Resultset 형태로 리턴
    - 예

```
Statement stmt = con.createStatement();
String sql = “SELECT * FROM enroll”;
ResultSet rs = stmt.executeQuery(sql);
```

  - Int executeUpdate(String sql)
    - SQL문 중 SELECT문 이외의 SQL 명령문
    - 명령문이 영향을 끼친 row 개수(변경 개수)를 리턴
    - 예

```
Statement stmt = con.createStatement();
String sql = “INSERT INTO enroll (s_id,c_id,c_id_no,e_year,e_semester)
                    VALUES (‘123’,’C400’,3,2024,1)”;
Int res = stmt.executeUpdate(sql);
```

Statement 객체로 SQL 명령문 실행(3)

- 실행(Cont'd)
  - boolean execute(String sql)
    - 모든 종류의 SQL문
    - 첫 번째 결과가 ResultSet이면 true,
      결과가 없거나 변경 개수이면 false 리턴
    - 실행 후, 결과에 따라 다음 명령문 추가 실행
      - ResultSet getResultSet()
      - Int getUpdateCount()
    - 예

```
if (stmt.execute(query) == false) {
        int num = stmt.getUpdateCount();
        System.out.println(num + " rows affected");
} else {
        ResultSet rs = stmt.getResultSet();
        /* print rs */
}
```

ResultSet 객체로 데이터 가져오기(1)

- ResultSet 역할
  - 데이터베이스 질의에 의해 얻어진 데이터의 로우(row)를 대표하는 오브젝트
- 주요 메소드
  - 결과 값의 컬럼에 해당하는 데이터를 얻기 위해 일련의 메소드 제공
  - 데이터를 한 번에 하나씩 처리하므로 next()로 결과 값의 다음 행 지정

ResultSet 객체로 데이터 가져오기(2)

- 명시된 데이터 타입으로 컬럼 데이터 가져오기
  - xxx getXxx(String columnName)
    - 예: resultSet.getInt(“StoreID”)
  - xxx getXxx(int columnPosition)
    - columnPosition값은 0이 아니라 1로 시작
    - 예 : resultSet.getInt(1);

ResultSet 객체로 데이터 가져오기(3)

- 널(NULL)값 처리하기
  - 널 값은 주어진 컬럼에 대한 어떤 데이터도 정의되어 있지 않다는 것을 나타냄
  - 컬럼 데이터가 널 값인 경우에 getXXX() 메소드를 호출할 때 리턴 값
    - 오브젝트를 리턴하는 메소드 : java null
    - 숫자를 리턴하는 메소드 : 0
    - getBoolean() : false
  - 널 값 처리 방법 (널 값 유무 확인)
    - JDBC 메소드 사용 : wasNull() 메소드 호출
    - SQL문 사용 : 조건 절에서 널 값 유무 확인

ResultSet 객체로 데이터 가져오기(4)

- 자바 데이터 타입과 SQL 데이터 타입의 맵핑

SQL 타입	자바 타입	SQL 타입	자바 타입
BIT	boolean	DATE	java.sql.Date
TINYINT	byte	TIME	java.sql.Time
SMALLINT	short	TIMESTAMP	java.sql.Timestamp
INTEGER	int	BINARY	byte[]
BIGINT	long	VARBINARY	byte[]
REAL	float	LONGVARBINARY	byte[]
FLOAT	double	BLOB	java.sql.Blob
DOUBLE	double	CLOB	java.sql.Clob
NUMERIC	java.math.BigDecimal	ARRAY	java.sql.Array
CHAR	java.lang.String	REF	java.sql.Ref
VARCHAR	java.lang.String	STRUCT	java.sql.Struct
LONGVARCHAR	java.lang.String		

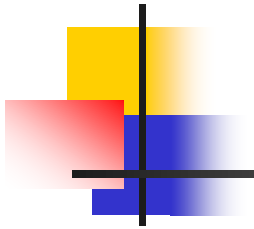
ResultSet 객체에 데이터 가져오기(5)

- ResultSet 객체 네비게이션
  - ResultSet 객체 생성시의 데이터 행 위치 : 처음 행의 바로 전
  - next() 메소드
    - 처음 호출하는 경우 : 포인터를 처음 행에 위치
    - 처음 호출하는 것이 아닌 경우 : ResultSet을 한 번에 한 행씩 전진 이동
  - 스크롤 가능한 ResultSet의 경우는 next() 이외에도 다양한 이동 방법이 존재
    - previous(), first(), last(), absolute(), relative()
    - moveToInsertRow(), moveToCurrentRow()

트랜잭션

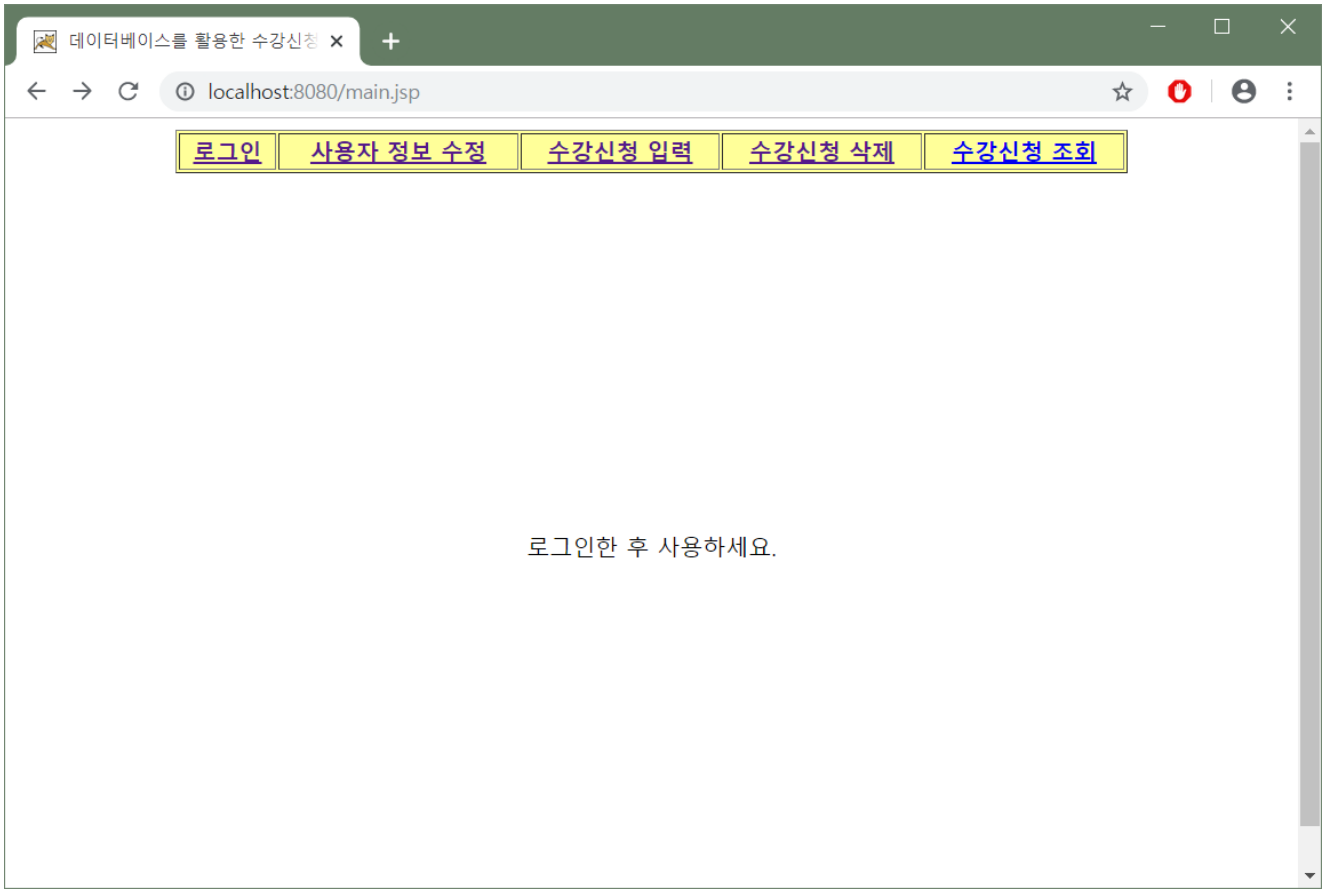
- JDBC에서의 트랜잭션
  - 데이터베이스 처리의 논리적 단위로 함께 실행되는 하나 이상의 Statement
- JDBC에서는 어떤 방식으로 Connection 객체를 생성하더라도 AutoCommit 모드를 on 시킴
  - AutoCommit 모드(디폴트)
    - 각 SQL 문장이 트랜잭션으로 간주됨
    - 각 트랜잭션은 실행된 후 바로 커밋됨
  - 2개 이상의 statement를 하나의 트랜잭션으로 그룹핑하는 방법
    - AutoCommit모드를 off로 셋팅한 후 사용
    - 예)

```
Connection conn = DriverManger.getConnection();
Conn.setAutoCommit(false);
…
If (<executed-transaction---successfully>)  conn.commit();
else                                        conn.rollback();
```

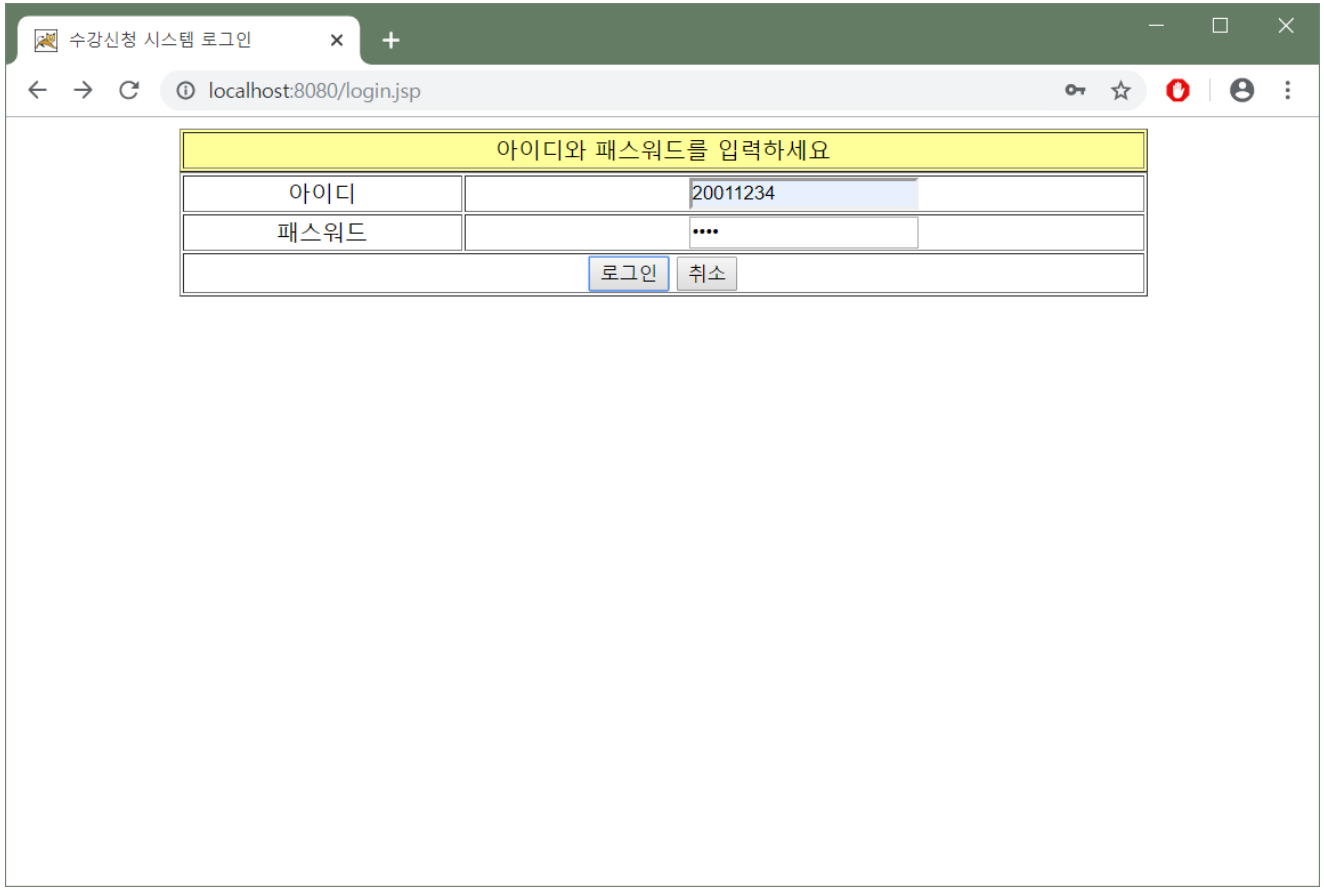


데이터베이스 연결 및 SQL 처리 실습
: 수강신청 시스템
(로그인, 사용자 정보 수정, 로그아웃)

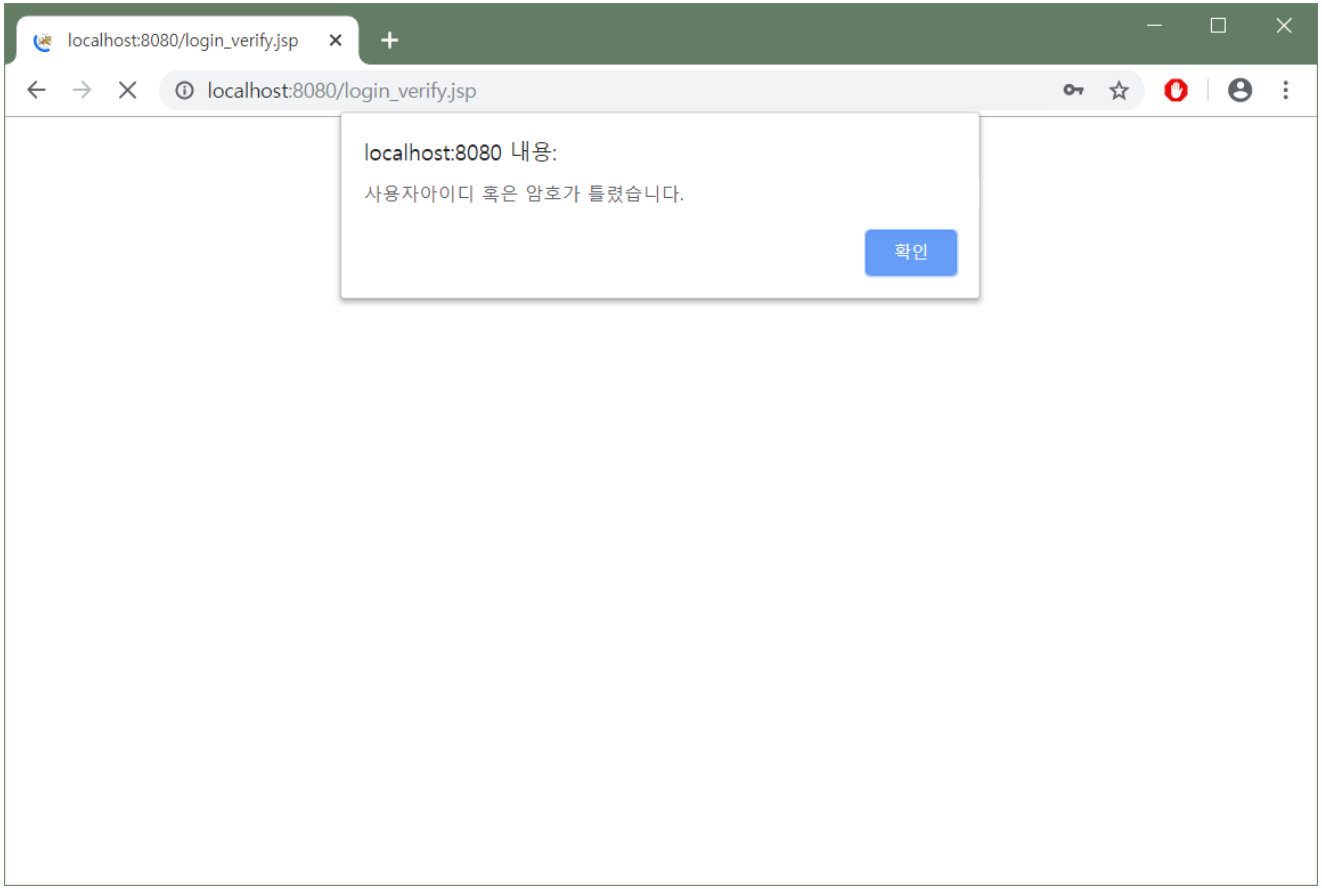
메인 화면 : main.jsp



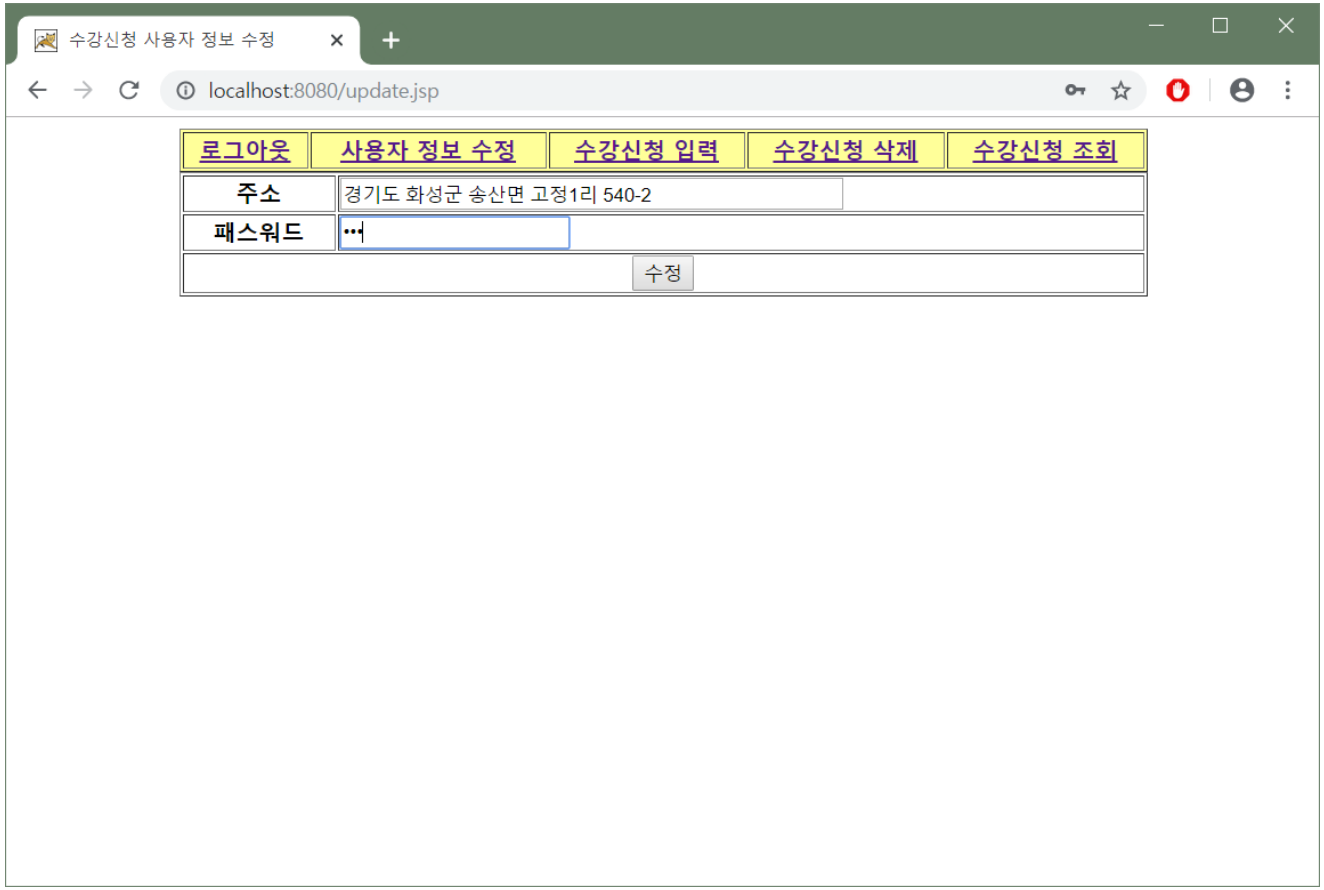
로그인 화면 : login.jsp



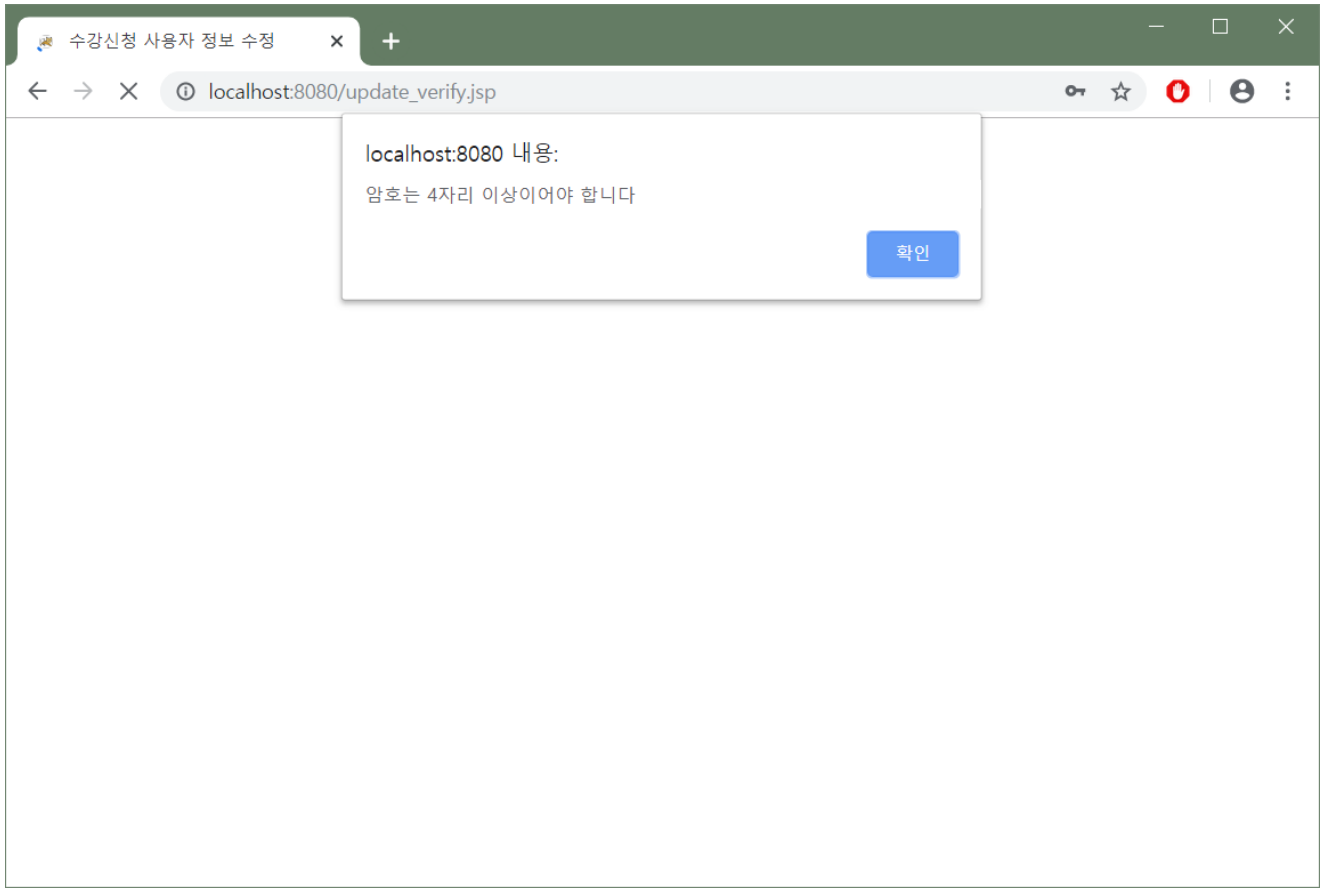
로그인 정보 확인 화면 : login_verify.jsp



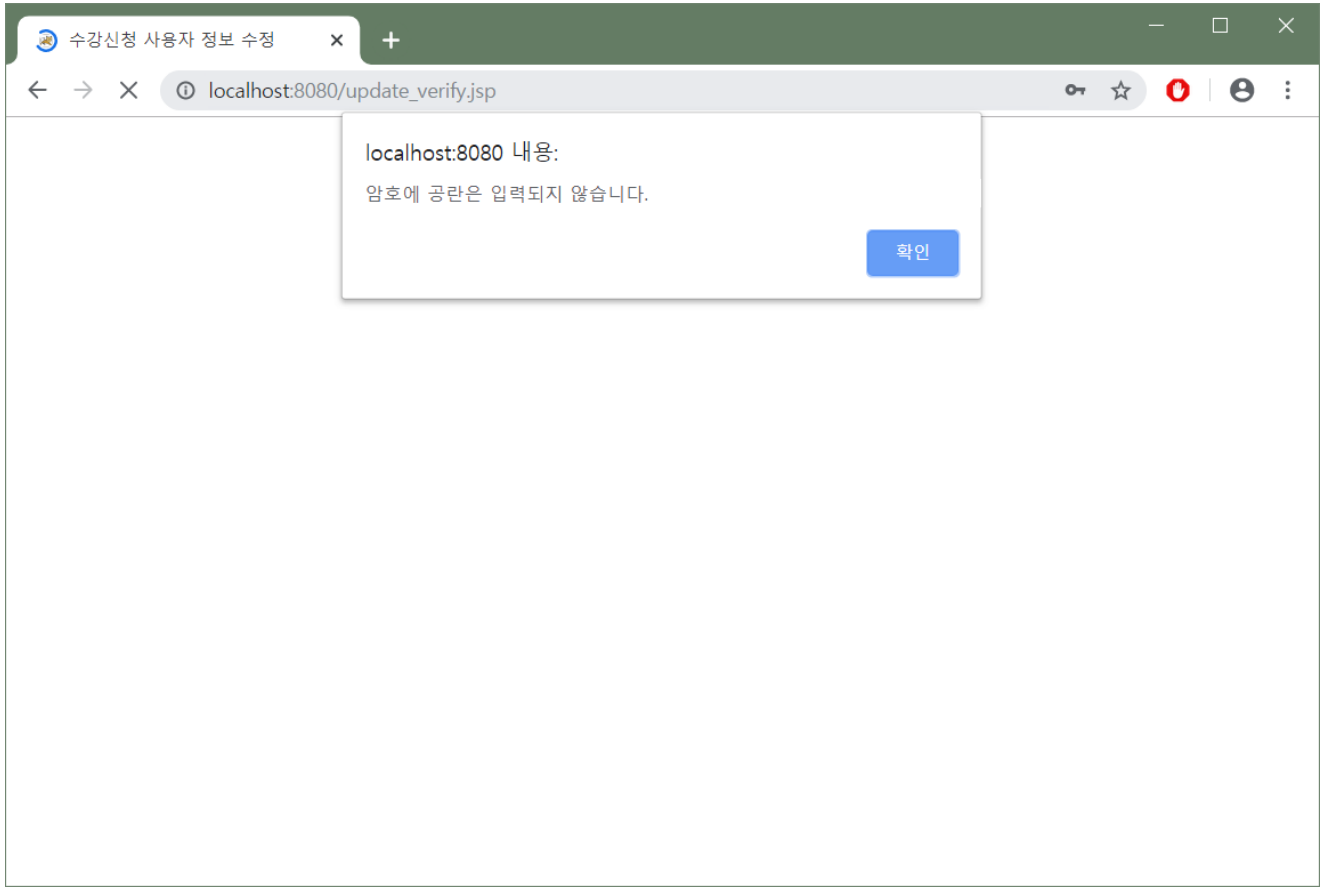
사용자 정보 수정 화면 : update.jsp



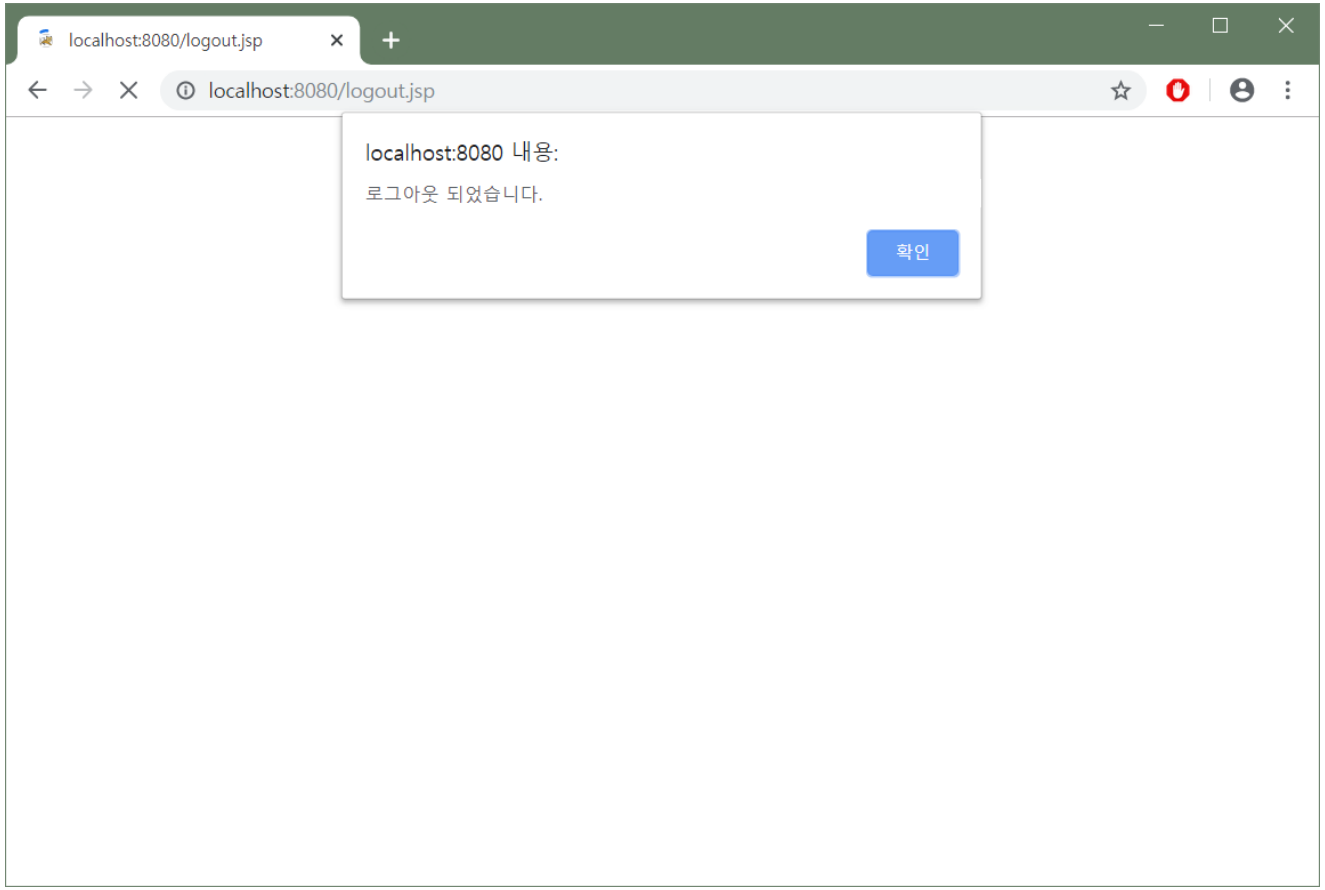
사용자 정보 수정 확인 화면(1) : update_verify.jsp

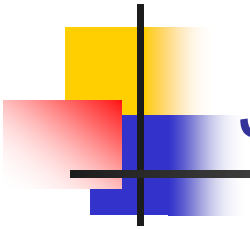


사용자 정보 수정 확인 화면(2) : update_verify.jsp



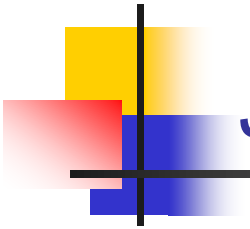
로그 아웃 화면 : logout.jsp





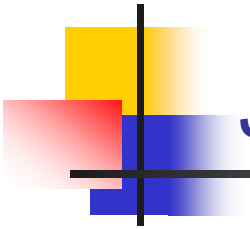
JSP(1)

- main.jsp
  - 메인 화면을 보여줌
  - top.jsp 포함
- top.jsp
  - 학번을 이용한 세션 관리
  - 메뉴 관리
- login.jsp
  - 로그인에 필요한 화면 구성
  - 아이디와 패스워드 입력한 후 올바른지 여부는 login_veriy.jsp가 담당



JSP(2)

- login_verify.jsp
  - 로그인 시 입력된 아이디와 패스워드가 맞는지 확인
    - 데이터베이스 연결
    - 질의 수행(Statement 사용)
  - 아이디와 패스워드가 올바른 경우
    - 기타 기능을 사용할 수 있도록 함
    - main.jsp로 리턴
  - 아이디와 패스워드가 올바르지 않은 경우
    - login.jsp로 리턴



JSP(3)

- update.jsp
  - 사용자 정보 수정 화면
  - 로그인 정보를 통해 사용자 정보를 조회하여 화면에 보여줌
    - 데이터베이스 연결
    - 질의 수행
  - 수정을 요청하는 경우, update_verify.jsp 호출
- update_verify.jsp
  - update.jsp에서 입력된 값을 수정
    - 데이터베이스 연결
    - 테이블 수정 : 트리거 활용
- logout.jsp
  - 세션 해제
  - main 화면으로 리턴

Cont.

main.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<html><head>
<title>데이터베이스를 활용한 수강신청 시스템입니다.</title></head>
<body>
<%@ include file="top.jsp" %>
<table width="75%" align="center" height="100%">
<%  if (session_id != null)  {  %>
<tr>
    <td align="center"><%=session_id%>님 방문을 환영합니다.</td>
</tr>
<% } else {  %>
<tr>
    <td align="center">로그인한 후 사용하세요.</td>
</tr>
<%
}
%>
</table>
</body>
</html>
```

Cont.

top.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%
    String session_id = (String)session.getAttribute("user");
                    세션 이용
    String log;
    if (session_id==null)  log="<a href=login.jsp>로그인</a>";
    else                   log="<a href=logout.jsp>로그아웃</a>";
%>

<table width="75%" align="center" bgcolor="#FFFF99" border>
  <tr>
   <td align="center"><b><%=log%></b></td>
   <td align="center"><b><a href="update.jsp">사용자 정보 수정</b></td>
   <td align="center"><b><a href="insert.jsp">수강신청 입력</b></td>
   <td align="center"><b><a href="delete.jsp">수강신청 삭제</b></td>
   <td align="center"><b><a href="select.jsp">수강신청 조회</b></td>
  </tr>
</table>
```

Cont.

login.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<HTML><head><title>수강신청 시스템 로그인</title></head>
<BODY>
<table width="75%"  align="center" bgcolor="#FFFF99" border>
  <tr><td><div align="center"> 아이디와 패스워드를 입력하세요</table>
<table width="75%" align="center" border>
<FORM method="post" action="login_verify.jsp" >
  <tr><td><div align="center">아이디</div></td>
       <td><div align="center"><input type="text" name="userID"></div></td>
  </tr>
  <tr><td><div align="center">패스워드</div></td>
       <td><div align="center"><input type="password" name="userPassword">
       </div></td>
  </tr>
  <tr>
    <td colspan=2><div align="center">
      <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="로그인">
      <INPUT TYPE="RESET" VALUE="취소">
    </div></td>
  </tr>
</table>
</FORM>
</BODY> </HTML>
```

Cont.

login_verify.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*" %>
<%
  String userID = request.getParameter("userID");
  String userPassword = request.getParameter("userPassword");

  Connection myConn = null;
  Statement stmt = null;
  String mySQL = null;

  String dburl = "jdbc:oracle:thin:@210.94.199.20:1521:dblab";
  String user = "사용자 계정";
  String passwd = "비밀번호";
  String dbdriver = "oracle.jdbc.driver.OracleDriver";

  Class.forName(dbdriver);
  myConn =  DriverManager.getConnection (dburl, user, passwd);
```

Cont.

```
 stmt = myConn.createStatement();

 mySQL = “SELECT s_id FROM student WHERE s_id='" + userID + "' AND
    s_pwd='" + userPassword + "'";

 ResultSet myResultSet = stmt.executeQuery(mySQL);
 if (myResultSet.next()) {
    session.setAttribute("user", userID);
    response.sendRedirect("main.jsp");
 } else {
%>
<script>
    alert("사용자아이디 혹은 암호가 틀렸습니다");
    location.href = "login.jsp";
</script>
<%
 }
 stmt.close();
 myConn.close();
%>
```

Cont.

update.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*"  %>
<html>
<head><title>수강신청 사용자 정보 수정</title></head>
<body>
<%@ include file="top.jsp" %>
<%
      if (session_id==null) response.sendRedirect("login.jsp");

      Connection myConn = null;     Statement stmt = null;
      ResultSet myResultSet = null; String mySQL = "";

      String dburl  = "jdbc:oracle:thin:@210.94.199.20:1521:dblab";
      String user = "사용자 계정";   String passwd = "비밀번호";
      String dbdriver = "oracle.jdbc.driver.OracleDriver";

      try {
          Class.forName(dbdriver);
          myConn =  DriverManager.getConnection (dburl, user, passwd);
          stmt = myConn.createStatement();
      } catch(SQLException ex) {
           System.err.println("SQLException: " + ex.getMessage());
      }
```

```
    mySQL = “SELECT s_addr,s_pwd FROM student WHERE s_id='" + session_id + "'" ;
    myResultSet = stmt.executeQuery(mySQL);
    if (myResultSet.next()) {
                String s_addr = myResultSet.getString("s_addr");
                String s_pwd = myResultSet.getString("s_pwd");
%>

<form method="post" action="update_verify.jsp">
  <input type="hidden" name="s_id" size="30" value="<%= session_id %>">
  <table width="75%" align="center" border>
    <tr><th>주소</th>
       <td><input type="text" name="s_addr" size="50" value="<%= s_addr %>">
       </td></tr>
     <tr><th>패스워드</th>
          <td><input type="password" name="s_pwd" size="20"
                value="<%= s_pwd %>"></td></tr>

<%
      }
      stmt.close();  myConn.close();
%>
<tr><td colspan="2" align="center"><input type="submit" value="수정"></td> </tr>
</table></form></body></html>
```

update_verify.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*" %>
<html>
<head><title> 수강신청 사용자 정보 수정 </title></head>
<body>
<%
    String s_id = request.getParameter("s_id");

    String s_addr = new String(request.getParameter("s_addr").getBytes("8859_1"),"utf-8");
    String s_pwd = new String(request.getParameter("s_pwd"));

    Connection myConn = null;  Statement stmt = null;  String mySQL = "";

    String dburl  = "jdbc:oracle:thin:@210.94.199.20:1521:dblab";
    String user = "사용자 계정";  String passwd = "비밀번호";
    String dbdriver = "oracle.jdbc.driver.OracleDriver";

    try {
       Class.forName(dbdriver);
       myConn =  DriverManager.getConnection (dburl, user, passwd);
       stmt = myConn.createStatement();
    } catch(SQLException ex) {
      System.err.println("SQLException: " + ex.getMessage());
    }
```

```
      mySQL = “UPDATE student ";
      mySQL = mySQL + " SET s_addr ='" + s_addr + "' , " ;
      mySQL = mySQL + " s_pwd ='" + s_pwd + "' WHERE s_id='" + s_id + "' ";
      try  {
        stmt.executeQuery(mySQL);
%>
<script>
  alert("학생 정보가 수정되었습니다.");       location.href="update.jsp";
</script>
<%
    } catch(SQLException ex) {
        String sMessage;
        if (ex.getErrorCode() ==  20002) sMessage="암호는 4자리 이상이어야 합니다";
        else if (ex.getErrorCode() == 20003) sMessage="암호에 공란은 입력되지 않습니다.";
        else sMessage="잠시 후 다시 시도하십시오";
%>
<script>
      alert("<%= sMessage %>");    location.href = "update.jsp";
</script>
<% } finally {
         if (stmt != null)   try { stmt.close();   myConn.close(); }
                             catch(SQLException ex) { }
   }
%>
</body></html>
```

2부에서 작성한 트리거에 의한 결과

Cont.

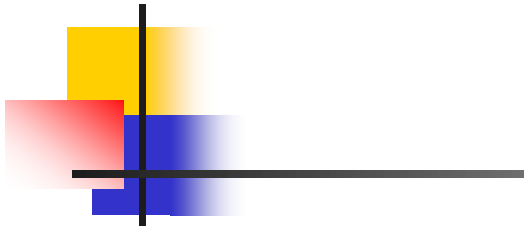
logout.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<% session.invalidate(); %>

<script>
  alert("로그아웃 되었습니다.");
  location.href="main.jsp";
</script>
```

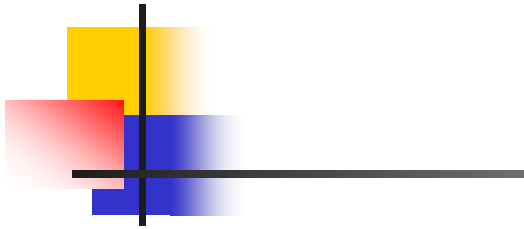
데이터베이스 프로그래밍

(소프트웨어 개발 트랙)

* 정보통신산업진흥원과 한국정보처리학회의 자료를 수정 보완함.



제 3부
데이터베이스
어플리케이션 프로그래밍(3)



미리 컴파일된 SQL문 호출

PreparedStatement 개요

- Statement 객체
  - SQL 명령문을 DBMS에 보낼 때마다 DBMS는 매번 아래의 작업을 반복함
    - SQL 문자열 컴파일
    - 컴파일된 SQL 명령문 실행
    - 실행 결과를 호출자에게 리턴
- PreparedStatement 객체
  - 특정 테이블에 데이터를 입력할 경우, SQL 명령문은 대부분 동일하고, 입력되는 데이터 값만 다르므로 재사용하는 것이 바람직함
  - PreparedStatement 인터페이스는 이런 상황에서 사용하기 위해 개발됨

PreparedStatement 필요성

- 반복되는 질의 계획 없앰
  - 데이터베이스로 보낸 각각의 SQL 명령문은 그때마다 컴파일되어 이에 해당하는 새로운 질의 계획 구축
  - 그러나, 동일한 질의 계획을 갖는 명령문이 존재하는 경우 이를 반복 수행하는 것은 불필요
  - PreparedStatement를 사용하게 될 때의 효과
    - 실제 SQL이 수행되기 전에 해당 SQL 호출의 질의 계획을 생성한 후, SQL을 실행할 시간이 되면 준비된 질의 계획으로 SQL 실행
- 프로그램 가독성(readability)에 좋음
  - 프로그래머 입장에서, 해당되는 SQL문을 String 결합 방법을 통해 구하는 것보다 알아보기 쉬움

PreparedStatement 사용(1)

- 생성
  - Connection 메소드인 prepareStatment() 사용
  - 예

```
Connection conn = DriverManager.getConnection();
PreparedStatement pstmt = conn.prepareStatement
("SELECT * FROM emp WHERE id=? AND name=?");
```

PreparedStatement 사용(2)

- PreparedStatment 파라미터에 대한 값 지정
  - '?'자리에 들어갈 값 제공
  - setXXX()
  - 예

```
pstmt.setString(1, "123");

pstmt.setString(2, "dbman");
```

Connection.prepareStatment 메소드

- 나중에 데이터를 셋팅하기 위한 '?'를 가짐
- '?'는 1부터 시작하는 인덱스를 사용
- '?'는 데이터가 전달되는 위치에서만 사용될 수 있고,
  테이블이나 컬럼 이름과 같은 위치에는 사용될 수 없음
- 잘못된 예 : DELETE FROM ? WHERE ? = 1;
- '?'에 데이터를 셋팅해주는 메소드의 순서는 상관이 없지만,
  모든 데이터를 셋팅해주어야 함(하지 않으면, SQLException() 발생)
- '?'에 데이터를 한번 셋팅해주면 execute…() 호출 후에도 그 값이
  계속 유지됨

PreparedStatement 객체 활용 예 : 질의 효율성

- Statement 객체 생성
  - Statement stmt = conn.createStatement();

```
for (int i=1; i<=10; i++) {
    rs=stmt.executeQuery("SELECT * FROM emp WHERE no="+i+
    " AND name='"+name[i]+"'");
    ....
}
```

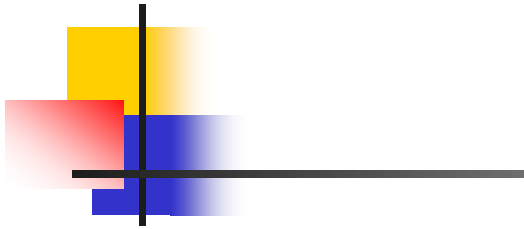
- PreparedStatement 객체 생성
  - PreparedStatement pstmt = conn.prepareStatement ("SELECT * FROM emp WHERE no=? AND name=?");

```
for (int i=1; i<=10; i++) {
    pstmt.setInt(1,i);    pstmt.setString(2,name[i]);
    rs = pstmt.executeQuery();
    ....
}
```

PreparedStatement 객체 활용 예 : 프로그램 가독성

- Statement 객체 생성
  - rs=stmt.executeQuery
    (“SELECT * FROM emp WHERE id=‘”+id+“’ AND name=‘”+name+“’”);
    - “, +, ‘ 등의 잦은 사용으로 인해 가독성 저하

- PreparedStatment 객체 생성
  - PreparedStatement pstmt = conn.prepareStatement
    (“SELEST * FROM emp WHERE id=? AND name=?”);
    pstmt.setString(1,id);
    pstmt.setString(2,name);
    - ‘?’ 자리에 필요한 컬럼 값을 쉽게 읽을 수 있음



저장 프로시저 호출

JDBC에서 저장 프로시저 사용

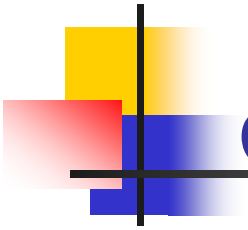
- 장점
  - 데이터베이스에 저장되어 있는 저장 프로시저에 대한 장점을 그대로 수용
  - 자바 개발자는 해당 저장 프로시저의 이름과 입력 및 출력만 알고 있으면 됨
    - 정보 캡슐화
    - 기능의 재사용
- JDBC에서 저장 프로시저 사용 방법
  - CallableStatment 인터페이스 사용

CallableStatement 객체 생성(1)

- 생성
  - Connection 메소드인 prepareCall() 사용
  - 프로시저 호출
    - CallableStatement pstmt = conn.prepareCall
      (“{call procedure-name }”);
    - CallableStatement pstmt = conn.prepareCall
      (“{call procedure-name(?,?,…,?) }”);
  - 함수 호출
    - CallableStatement pstmt = conn.prepareCall
      (“{? = call procedure-name(?,?,…,?) }”);
  - 중괄호(‘{’와 ‘}’)를 사용하는 이유는 DBMS에 적당한 호출형식으로 변환하도록 지시하는 것

CallableStatement 객체 생성(2)

- CallableStatment 파라미터 처리
  - 저장 프로시저의 IN형 파라미터
    - void setXXX(int pIndex, xxx)
  - 저장 프로시저의 OUT형 파라미터
    - void registerOutParameter(int pIndex, int jdbcType)
    - 참고) jdbcType : java.sql.Types.XXX
  - 저장 프로시저의 INOUT형 파라미터
    - IN형이기도 하고 OUT형이기도 하므로, 입력 값을 셋팅 해주고 리턴 값을 위한 Java 데이터 타입을 등록해주어야 함
    - void setXXX(int pIndex, xxx)
      void registerOutParameter(int pIndex, int jdbcType)



CallableStatement 객체로부터 데이터 얻어오기

- CallableStatment 객체가 가지고 있는 SQL 명령어를 실행
- getXXX()메소드를 통해, OUT/INOUT 파라미터의 값을 얻어옴

CallableStatement 객체 활용 예(1)

- IN 파라미터 만 있는 프로시저 처리
- 예

```
String sql = “{ call setPrice(?,?) }”;
CallableStatment stmt = conn.prepareCall(sql);
stmt.setInt(1, 1000);
stmt.setDouble(2,11.99);
stmt.execute();
```

CallableStatement 객체 활용 예(2)

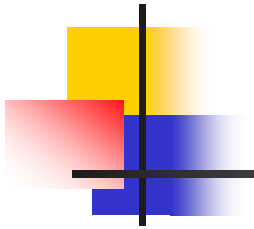
- OUT 파라미터가 존재하는 프로시저 처리
- 예

```
String sql = “{call InsertEnroll(?,?,?,?)}”;

CallableStatement cstmt = myConn.prepareCall(sql);
cstmt.setString(1, s_id);
cstmt.setString(2, c_id);
cstmt.setInt(3,c_id_no);
cstmt.registerOutParameter(4, java.sql.Types.VARCHAR);
cstmt.execute();
String result = cstmt.getString(4);
```

CallableStatement 객체 활용 예(3)

- 리턴값이 있는 함수 처리
- 예

```
String sql = “{? = call Date2EnrollYear(SYSDATE)}”;
cstmt = conn.prepareCall(sql);
cstmt.registerOutParameter(1,java.sql.Types.INTEGER);
cstmt.execute();
int nYear = cstmt.getInt(1);
```

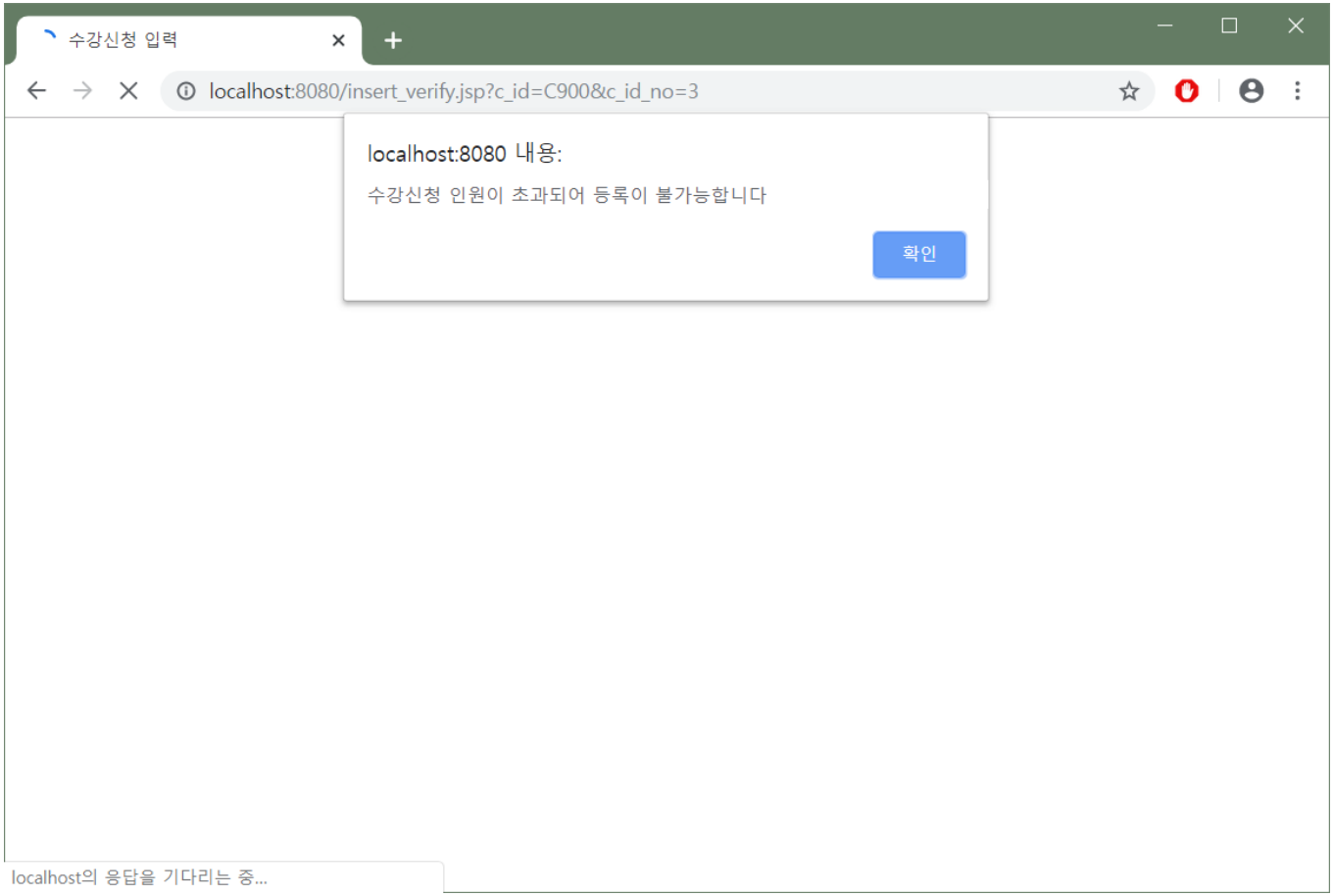


저장 프로시저 호출 실습
: 수강신청 시스템
(수강신청 입력)

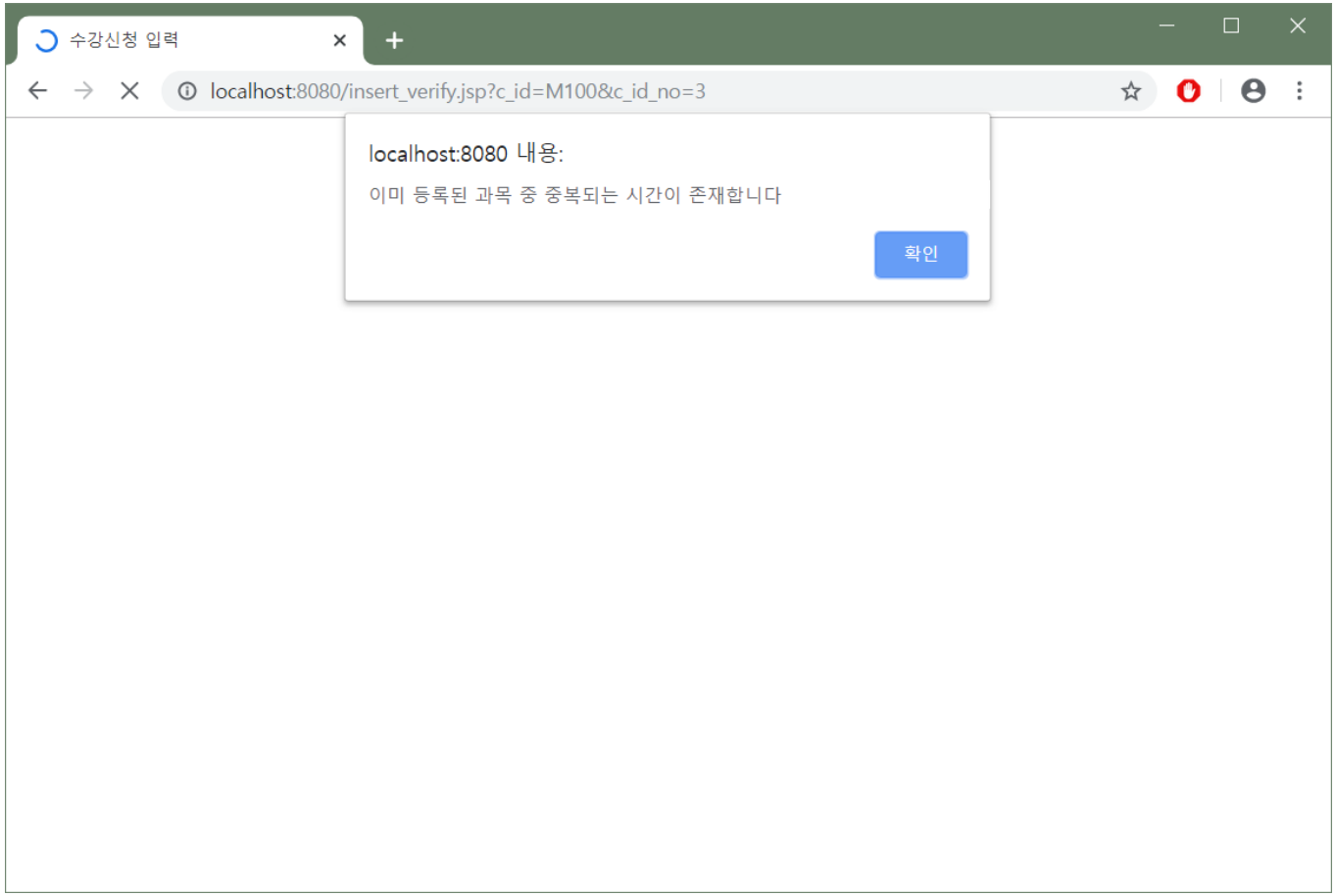
수강신청 입력 화면 : insert.jsp

과목번호	분반	과목명	학점	수강신청
C800	3	데이터베이스 프로그래밍	3	신청
C900	3	객체지향 윈도우즈 프로그래밍	3	신청
M100	3	멀티미디어 개론	3	신청
M200	3	선형대수	3	신청
M300	3	그래픽 활용	3	신청
M400	3	윈도우즈 프로그래밍	3	신청
M700	3	게임 프로그래밍	3	신청

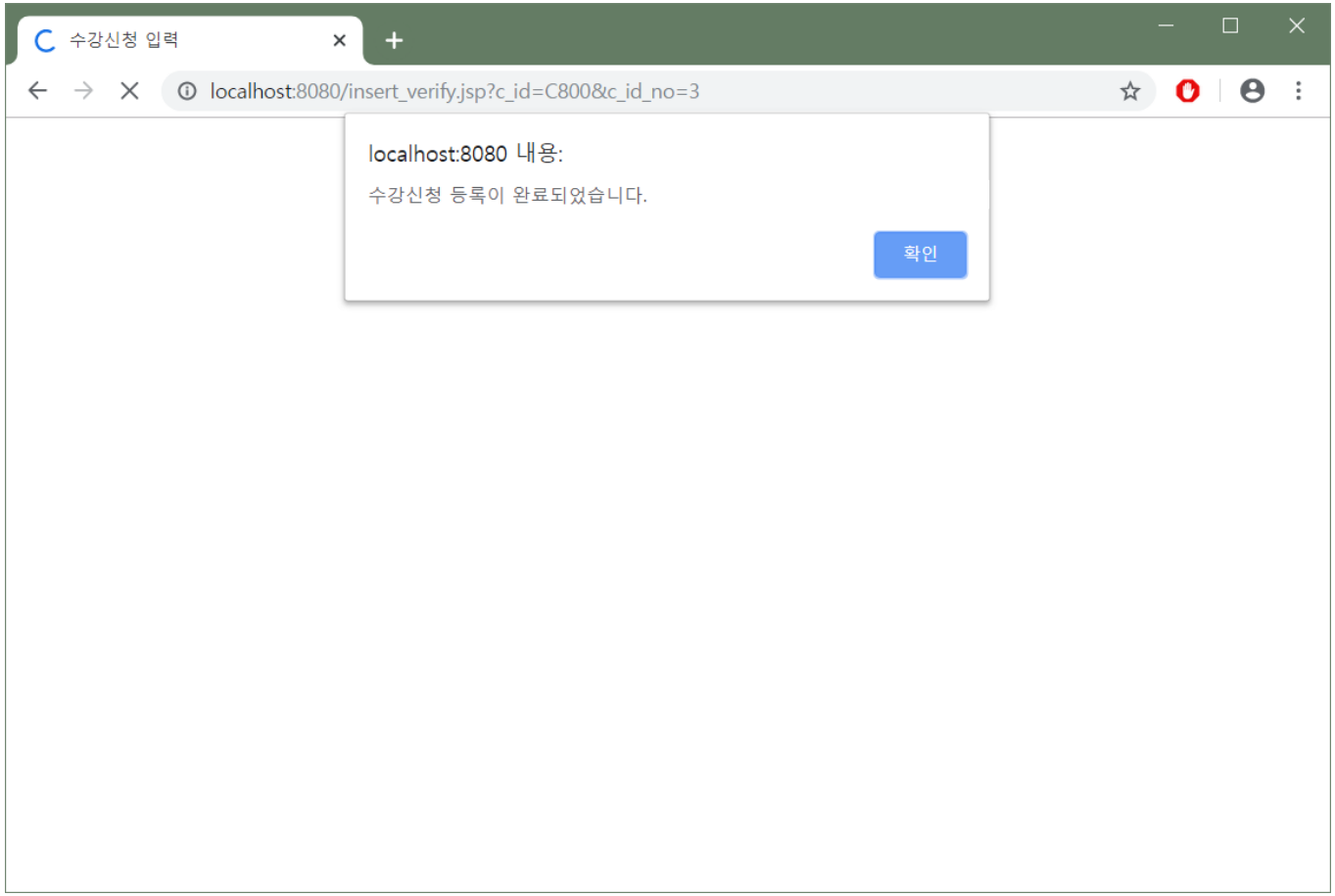
수강신청 입력 화면 : insert_verify.jsp (과목번호 'C900' 신청 후)



수강신청 입력 화면 : insert_verify.jsp (과목번호 'M100' 신청 후)



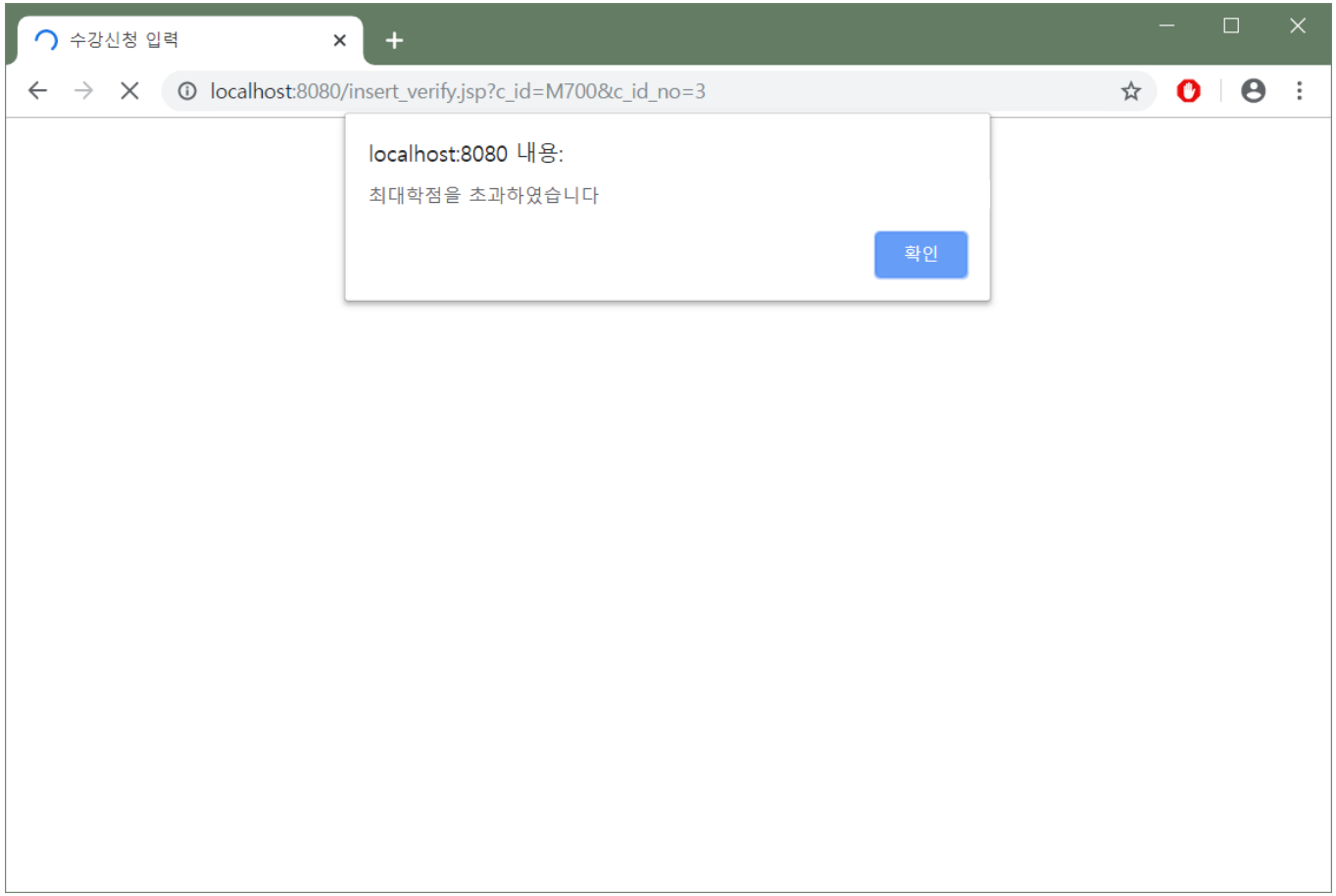
수강신청 입력 화면 : insert_verify.jsp (과목번호 'C800' 신청 후)(1)

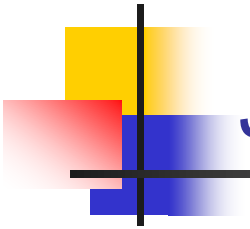


수강신청 입력 화면 : insert_verify.jsp (과목번호 'C800' 신청 후)(2)

과목번호	분반	과목명	학점	수강신청
C900	3	객체지향 윈도우즈 프로그래밍	3	신청
M100	3	멀티미디어 개론	3	신청
M200	3	선형대수	3	신청
M300	3	그래픽 활용	3	신청
M400	3	윈도우즈 프로그래밍	3	신청
M700	3	게임 프로그래밍	3	신청

수강신청 입력 화면 : insert_verify.jsp (과목번호 'M700' 신청 후)





JSP

- insert.jsp
  - 데이터베이스 연결
  - 수강 신청 가능한 과목 조회
    - Statement, ResultSet 사용
  - 수강신청 입력
    - insert_verify.jsp 호출
- insert_veify.jsp
  - 데이터베이스 연결
  - 2부에서 작성한 저장 프로시저 InsertEnroll 호출
    - CallableStatment 사용

insert.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*"  %>
<html><head><title>수강신청 입력</title></head>
<body>
<%@ include file="top.jsp" %>
<%   if (session_id==null) response.sendRedirect("login.jsp");  %>

<table width="75%" align="center" border>
<br>
<tr><th>과목번호</th><th>분반</th><th>과목명</th><th>학점</th>
     <th>수강신청</th></tr>
<%
      Connection myConn = null;      Statement stmt = null;
      ResultSet myResultSet = null;   String mySQL = "";
      String dburl  = "jdbc:oracle:thin:@210.94.199.20:1521:dblab";
      String user= " 사용자 계정 " ;     String passwd="비밀번호";
      String dbdriver = "oracle.jdbc.driver.OracleDriver";

      try {
                Class.forName(dbdriver);
                myConn =  DriverManager.getConnection (dburl, user, passwd);
                stmt = myConn.createStatement();
      } catch(SQLException ex) {
                System.err.println("SQLException: " + ex.getMessage());
      }
```

```
mySQL = “SELECT c_id, c_id_no, c_name, c_unit FROM course WHERE
        c_id NOT IN (SELECT c_id FROM enroll WHERE s_id='" + session_id + “‘
        AND  e_year=‘2024’ AND e_semester=‘1’ ) AND c_id IN (SELECT c_id
        FROM teach WHERE t_year=‘2024’ AND t_semester=‘1’)";

myResultSet = stmt.executeQuery(mySQL);

if (myResultSet != null) {
     while (myResultSet.next()) {
            String c_id = myResultSet.getString("c_id");
            int c_id_no = myResultSet.getInt("c_id_no");
            String c_name = myResultSet.getString("c_name");
            int c_unit = myResultSet.getInt("c_unit");
%>
<tr>
  <td align="center"><%= c_id %></td> <td align="center"><%= c_id_no %></td>
  <td align="center"><%= c_name %></td><td align="center"><%= c_unit %></td>
  <td align="center"><a href="insert_verify.jsp?c_id=<%= c_id %>&
                        c_id_no=<%= c_id_no %>">신청</a></td>
</tr>
<%
     }
}
stmt.close();  myConn.close();
%>
</table></body></html>
```

insert_verify.jsp

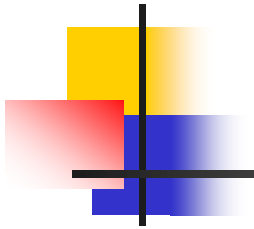
```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*" %>

<html><head><title> 수강신청 입력 </title></head>
<body>

<%
    String s_id = (String)session.getAttribute("user");
    String c_id = request.getParameter("c_id");
    int c_id_no = Integer.parseInt(request.getParameter("c_id_no"));
%>
<%
    Connection myConn = null;    String      result = null;
    String dburl  = "jdbc:oracle:thin:@210.94.199.20:1521:dblab";
    String user="사용자 계정";   String passwd= " 비밀번호 " ;
    String dbdriver =  " oracle.jdbc.driver.OracleDriver " ;

    try {
        Class.forName(dbdriver);
        myConn =  DriverManager.getConnection (dburl, user, passwd);
    } catch(SQLException ex) {
        System.err.println("SQLException: " + ex.getMessage());
    }
```

```
    CallableStatement cstmt = myConn.prepareCall("{call InsertEnroll(?,?,?,?)}");

     cstmt.setString(1, s_id);                  2부에서 작성한 저장 프로시저 사용
     cstmt.setString(2, c_id);                  : CallableStatement 사용
     cstmt.setInt(3,c_id_no);
     cstmt.registerOutParameter(4, java.sql.Types.VARCHAR);
     try  {
          cstmt.execute();
          result = cstmt.getString(4);
%>
<script>                              !!! 저장 프로시저에서 작성된
     alert("<%= result %>");          복잡한 입력 규칙 검사를 다시 작성하지 않아도 됨
     location.href="insert.jsp";
</script>
<%
     } catch(SQLException ex) {
          System.err.println("SQLException: " + ex.getMessage());
     }
     finally {
          if (cstmt != null)
             try { myConn.commit(); cstmt.close();  myConn.close(); }
             catch(SQLException ex) { }
    }
%>
</form></body></html>
```

데이터베이스 프로그래밍

(소프트웨어 개발 트랙)

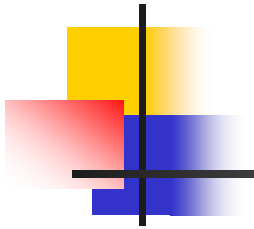
* 정보통신산업진흥원과 한국정보처리학회의 자료를 수정 보완함.



제 3부

데이터베이스

어플리케이션 프로그래밍(4)

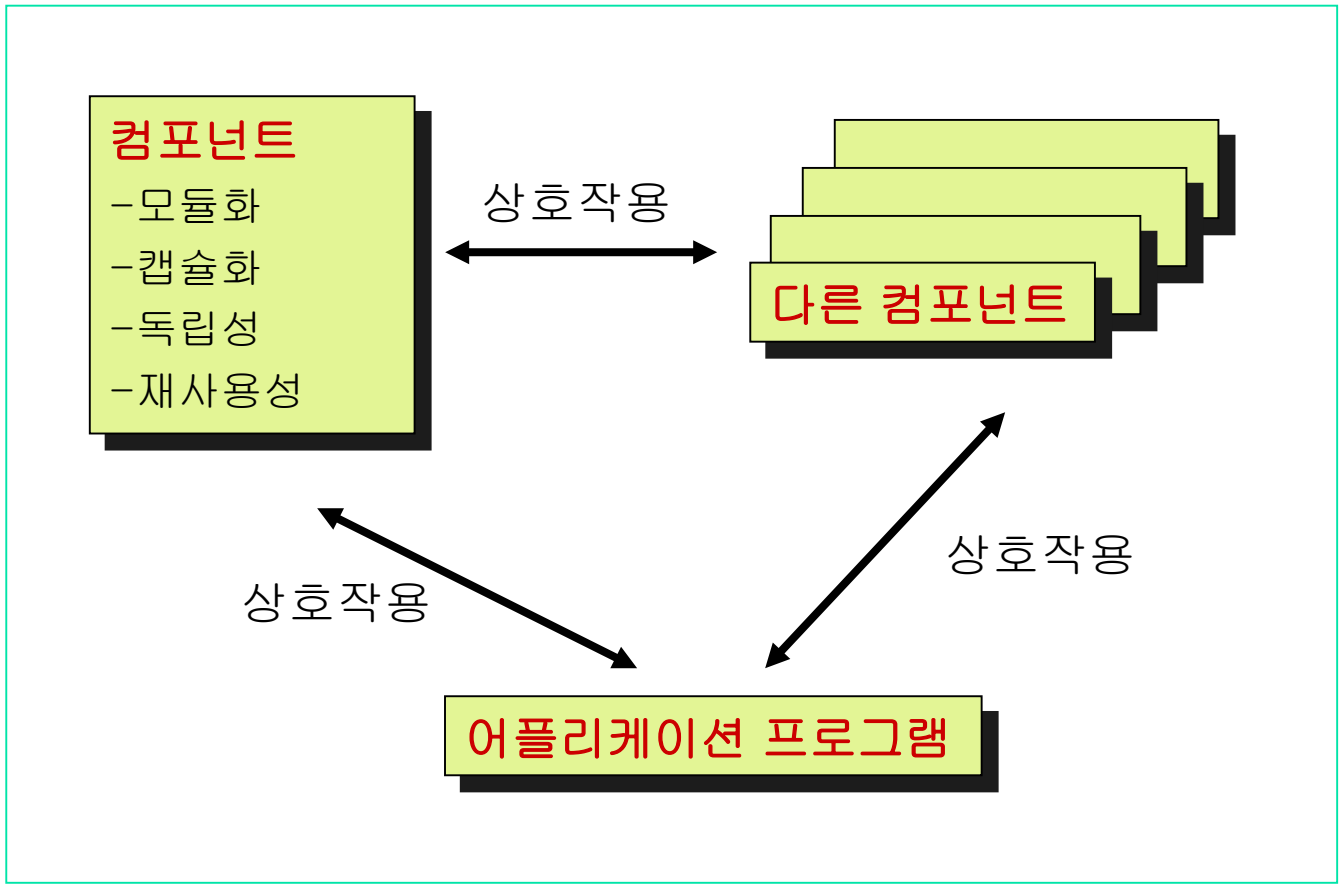


자바 빈즈를 사용한 데이터베이스 프로그래밍

소프트웨어 컴포넌트(1)

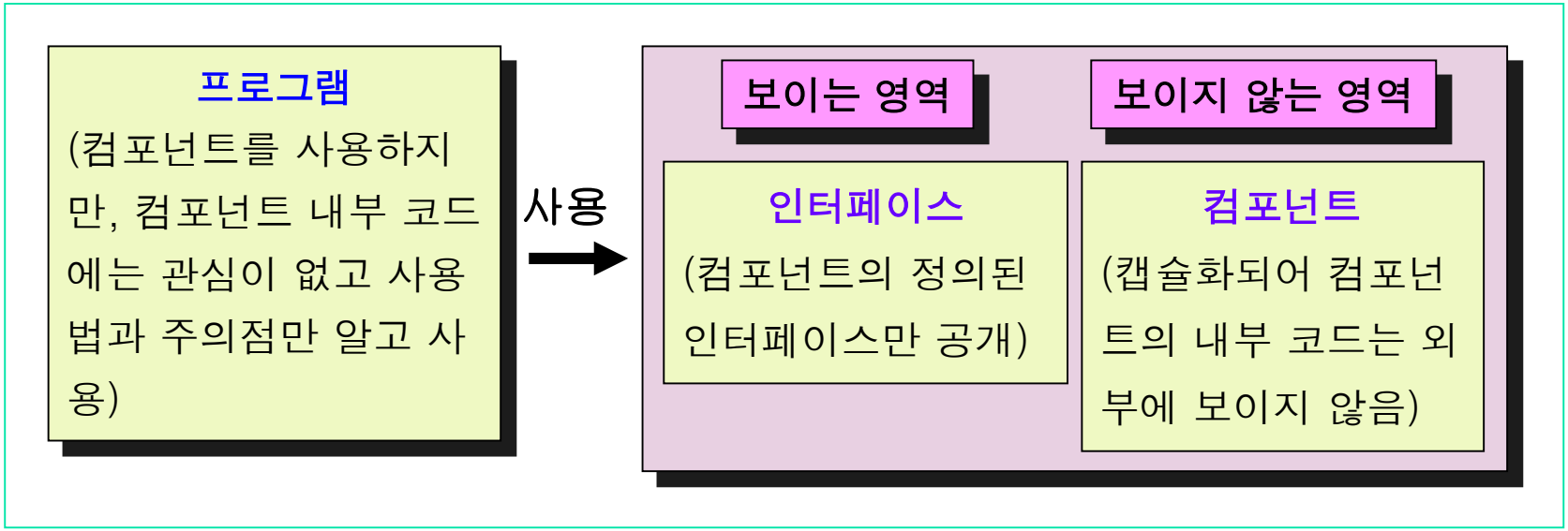
- 모듈화되고 재사용이 가능하며, 내부의 데이터와 로직이 캡슐화되어 있어서 컴포넌트를 사용하는 프로그램이 컴포넌트의 내부 실행 조직을 알지 못해도 실행 시 상호 작동할 수 있는 객체
- 특징
  - 소프트웨어 개발의 복잡성을 단순화시키고, 재사용이 가능하게 함으로써 개발 시간을 단축시키는 효과

소프트웨어 컴포넌트(2)



소프트웨어 컴포넌트(3)

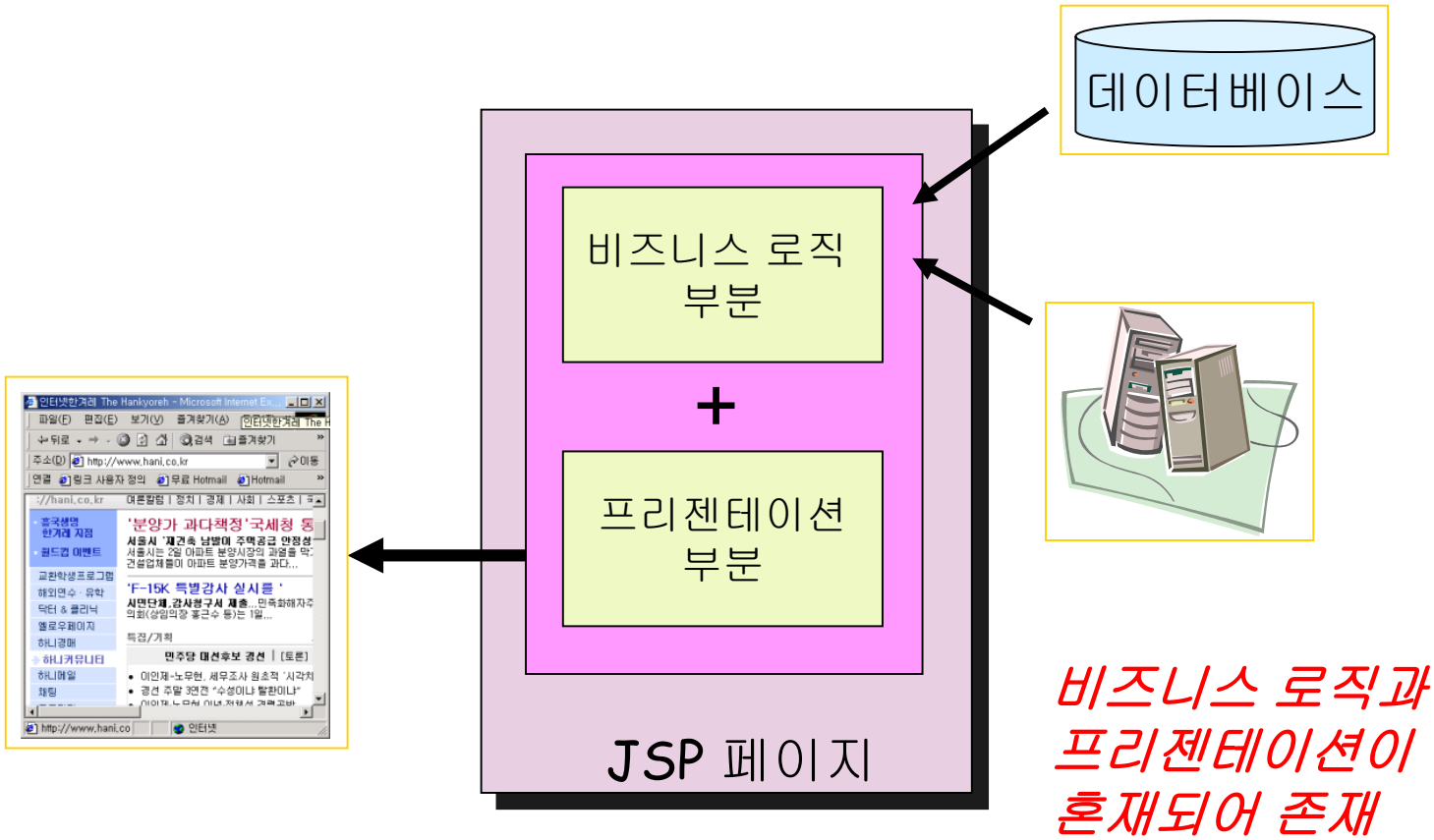
- 소프트웨어 컴포넌트 구조의 장점
  - 컴포넌트는 캡슐화되어 있음
  - 컴포넌트는 재사용 가능
  - 컴포넌트는 어플리케이션에 독립적



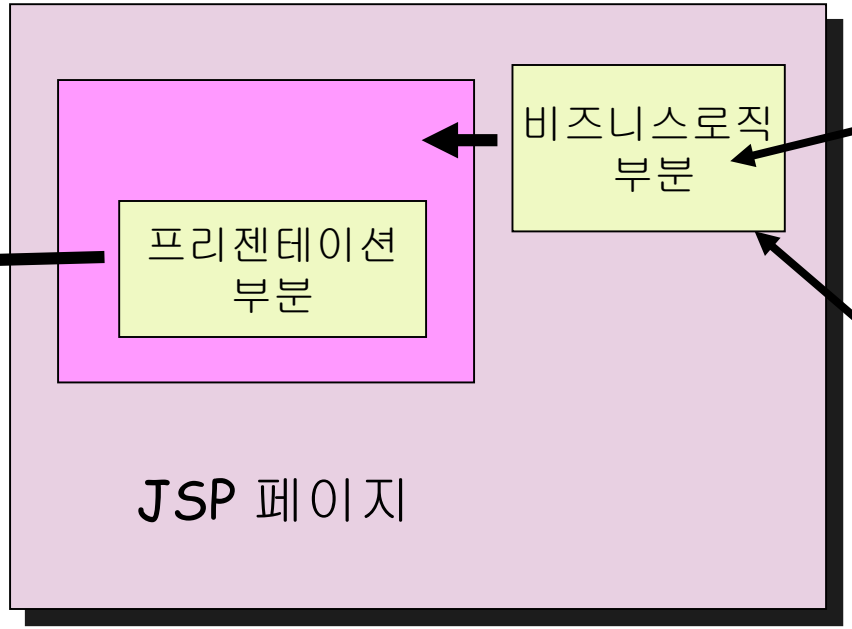
자바 빈즈와 소프트웨어 컴포넌트

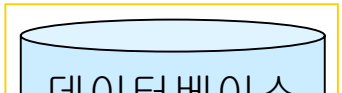
- 자바 빈즈(Java Beans) 의 이해
  - 자바 빈즈는 자바로 만들어진 소프트웨어 컴포넌트이며, 자바 빈즈 API규약을 따르는 컴포넌트를 빈즈(Beans)이라고 함
  - 자바 빈즈는 자바 클래스이며 단지 자바 빈즈 규약에 정의된 명명 규칙과 디자인 형식들을 따르기만 하면 됨

자바 빈을 사용하지 않은 JSP와 JDBC를 이용한 데이터베이스 어플리케이션

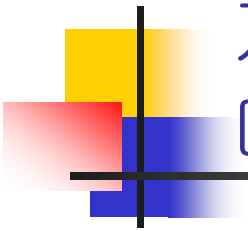


자바 빈을 사용한 JSP와 JDBC를 이용한 데이터베이스 어플리케이션





비즈니스 로직과 프리젠테이션이 분리됨



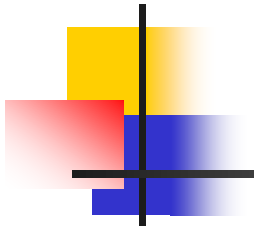
자바 빈즈를 사용한 데이터베이스 어플리케이션으로의 진화

자바 빈즈를 고려하지 않은

데이터베이스 어플리케이션으로부터

자바 빈즈를 이용한 데이터베이스 어플리케이션으로

발전시키자



연결 풀링을 사용한 데이터베이스 연결 기능 향상

연결 풀링의 필요성(1)

- 자원의 풀링
  - 프로그래밍에서 자주 사용되는 자원 관리 기법 중 하나
    - 예 : 자바에서의 스레드 풀링(thread pooling) 기법
  - 사용할 때마다 자원을 생성하는 것이 아니라, 먼저 고정된 수의 자원을 만들어 풀에 넣은 후 다음 사용 시 풀로부터 자원을 가져와서 사용

연결 풀링의 필요성(2)

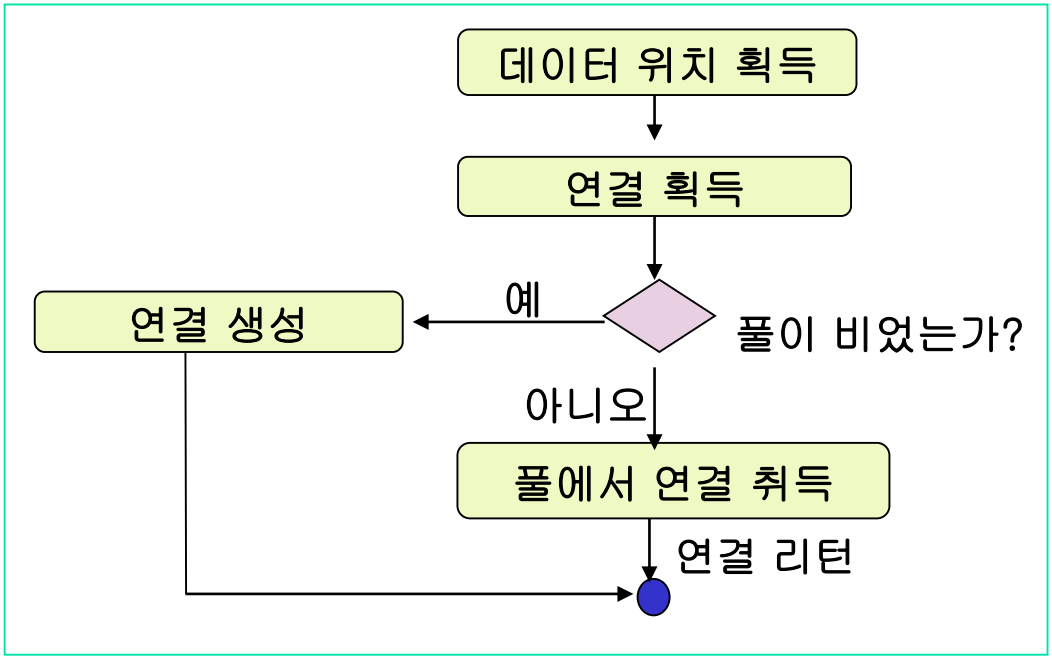
- 기존 데이터베이스 연결 방법의 문제점
  - 데이터베이스 연결 방법의 형태

```
Open Connection
    작업
Close Connection
    .....
Open Connection
    작업
Close Connection
```

  - 필요할 때마다 데이터베이스 연결을 만들고 삭제하는 것은 시스템 부하를 늘리는 요인이 됨
    - 객체 생성과 해제에는 많은 시스템 자원이 필요

연결 풀링(Connection Pooling)

- 풀 관리자
  - 연결을 관리하고 대기를 가능하도록 만드는 역할 수행
  - 클라이언트가 연결의 사용을 마치면, 이를 회수한 다음 닫지 않은 채로 대기시킴



연결 풀링 기법 활용 : 기본(1)

- 연결 풀 관리자 클래스 만들기

```
//  연결 풀 관리자 클래스 : 풀 생성
      class Manager {
            Stack pool = new Stack();
            Public Manager() {
              Class.forName("…").newInstance();
              Connection conn;

              for (int i=0; i<10; i++) {
                  conn = DriverManager.getConnection("..","..","..");
                  pool.push();
              }
       }
```

연결 풀링 기법 활용 : 기본(2)

```
// 연결 풀 관리자 클래스 : 연결 취득
        public synchronized Connection getConnection() {
            if (pool.size() != 0)  return pool.pop();
            return null;
        }

// 연결 풀 관리자 클래스 : 연결 반환
        public void returnConnection(Connection conn) {
            pool.push(conn);
        }
```

연결 풀링 기법 활용 : 기본(3)

- 연결 풀 관리자를 이용한 후의 데이터베이스 연결 방법의 변화

Get Connection
작업
Return Connection
.....
Get Connection
작업
Return Connection

연결 풀링 기법 활용 : 기본(4)

- JDBC를 사용하여 연결 풀링을 하는 방법
  - 드라이버에서 연결 풀 관리자를 제공하지 못하는 경우
  - Manager 클래스와 같이 직접 작성
  - 공개용 소프트웨어 사용(권장)

연결 풀링 기법 활용:옵션 패키지 사용(1)

- 연결 풀링 기능을 가진 JDBC 드라이버 사용
  - 참고) Oracle의 경우는 제공
- ConnectionPoolDataSource 인터페이스
  - Javax.sql 패키지에서 제공됨 : 옵션 패키지
  - JDBC에서는 인터페이스만 제공하고 있으며,
    각 드라이버 벤더에서 별도의 구현 클래스를 제공
  - getPooledConnection 메소드를 통해 연결 풀링에 사용될
    물리적(physical) 데이터베이스 연결

연결 풀링 기법 활용:옵션 패키지 사용(2)

- ConnectionPoolDataSource 인터페이스 사용 예

```
ConnectionPoolDataSourceImpl pds =
            new ConnectionPoolDataSourceImpl();
PooledConnection pc = pds.getPooledConnection();
Connection connection = pc.getConnection();
….
pc.close();
```

✓ 연결 풀링 기능이 없을 때의 close()

- o 데이터베이스에 연결을 끊음
- o 관련된 자원 해지

✓연결 풀링 기능이 있을 때의 close()

- o 연결 종료가 아닌, 풀로 리턴

연결 풀링 기법 활용:옵션 패키지 사용(3)

- ConnectionPoolDataSource 인터페이스의 장점
  - 투명한 프로그램 작성
    - 데이터베이스 연결 풀링 기법의 목적만 이해하면 됨
  - 공개용 소프트웨어를 사용하는 것보다 안정적, 효율적
    - 드라이버에서 제공한다면 이를 사용하는 것을 권장

- ConnectionPoolDataSource 인터페이스의 문제점
  - JDBC 본래의 목적인 소스 코드를 전혀 수정하지 않아도 된다는 취지 위배
    - 드라이버마다 이를 구현한 클래스의 이름이 다름

연결 풀링 기법 활용: 오라클 옵션 패키지 사용 예(1)

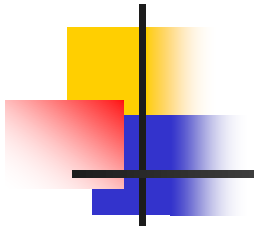
```
class PooledConnection
{
  public static void main (String args []) throws SQLException
  {
      OracleConnectionPoolDataSource ocpds =
              new OracleConnectionPoolDataSource();

    ocpds.setURL("jdbc:oracle:oci8:@[Server Name]”);
    ocpds.setUser(“db");
    ocpds.setPassword(“db");

    PooledConnection pc = ocpds.getPooledConnection();
    Connection conn = pc.getConnection();
```

oracle에서의 구현 클래스

연결 파라미터의 분리로 인한 보다 투명한 사용

연결 풀링 기법 활용: 오라클 옵션 패키지 사용 예(2)

```
    stmt = conn.createStatement ();
    ResultSet rset = stmt.executeQuery ("SELECT ename FROM emp");
    while (rset.next ())
        System.out.println (rset.getString (1));

  rset.close();
  rset = null;

  stmt.close();
  stmt = null;

  conn.close();   연결 종료가 아닌 풀로 리턴
  conn = null;

  pc.close();
  pc = null;
 }
}
```

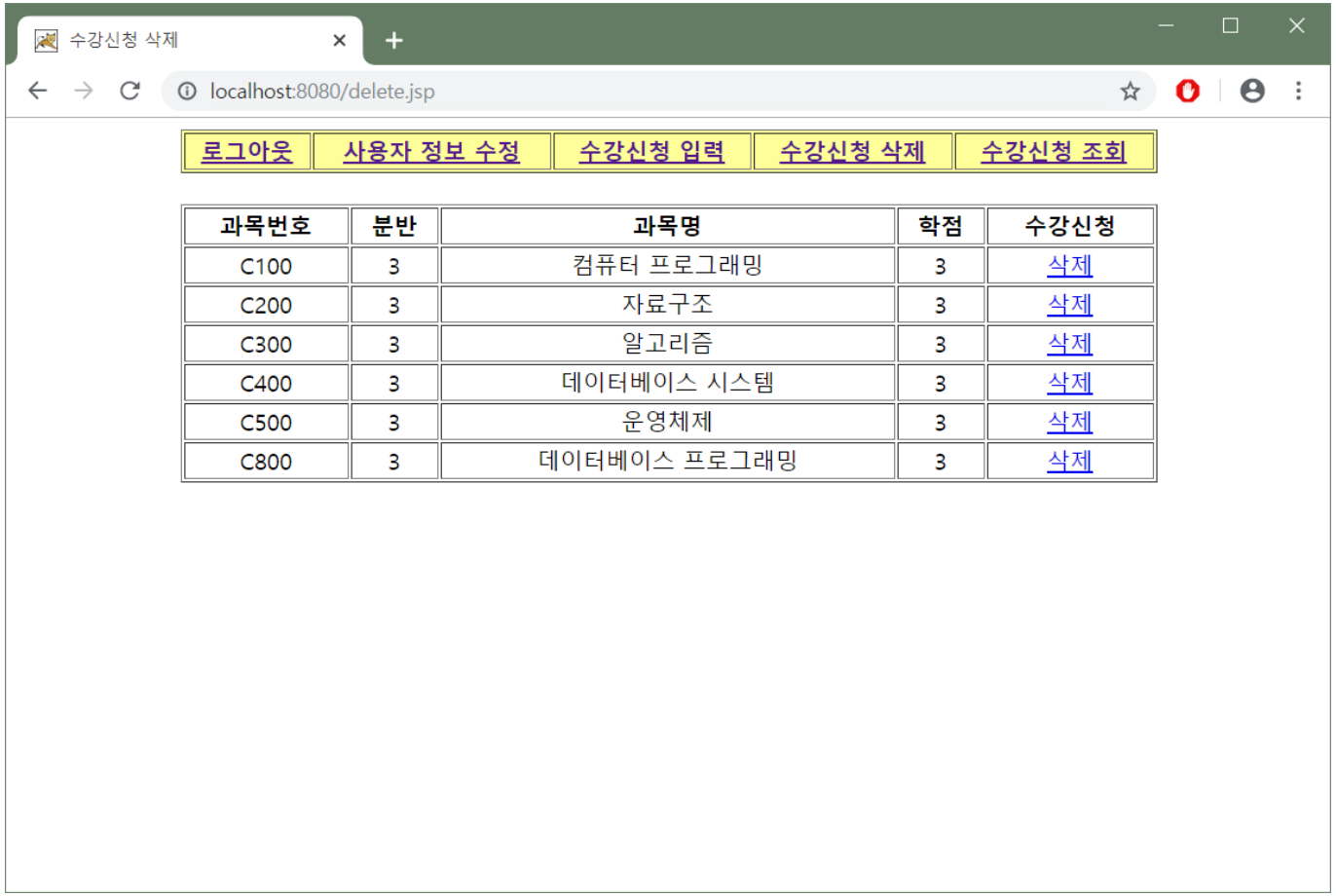


자바 빈즈, 연결 풀링 활용 실습

: 수강신청 시스템

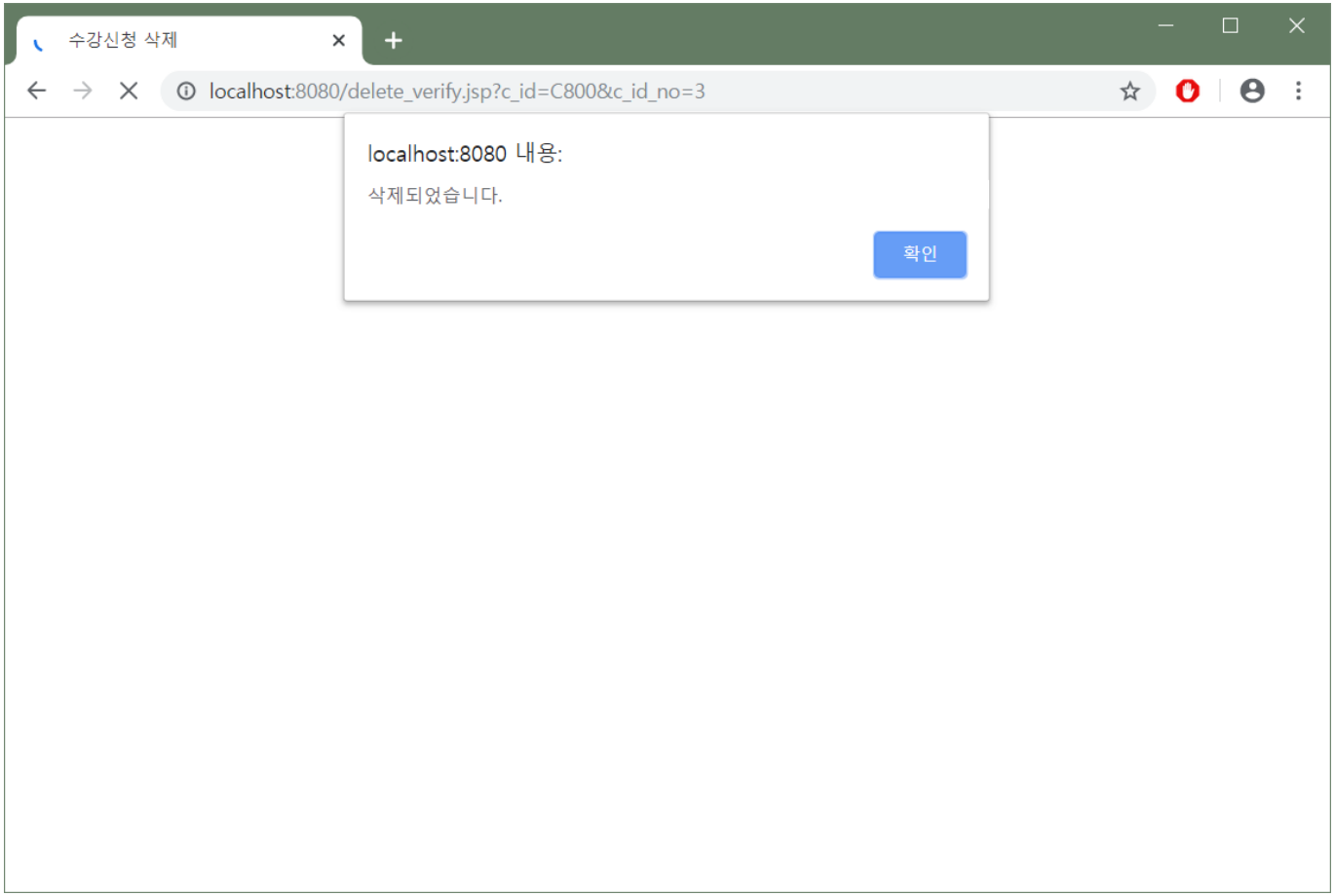
(수강신청 삭제, 수강신청 조회)

수강신청 삭제 화면 : delete.jsp



과목번호	분반	과목명	학점	수강신청
C100	3	컴퓨터 프로그래밍	3	삭제
C200	3	자료구조	3	삭제
C300	3	알고리즘	3	삭제
C400	3	데이터베이스 시스템	3	삭제
C500	3	운영체제	3	삭제
C800	3	데이터베이스 프로그래밍	3	삭제

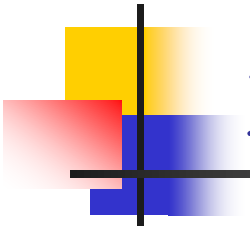
수강신청 삭제 화면 : delete_verify.jsp (과목번호 'C800' 삭제 후 확인 화면)



수강신청 삭제 화면 : delete_verify.jsp (과목번호 'C800' 삭제 후 메인 화면)

수강신청조회화면:select.jsp,select_verify.jsp (현재 년도와 현재 학기 신청과목 조회)

수강신청조회화면:select.jsp,select_verify.jsp (년도와 학기 변경 검색 조건 후 조회)



자바 빈즈

- EnrollMgr.java
- Enroll.java

공통적으로 사용할 자바빈즈 작성

- EnrollMgr.java
  - 데이터베이스 연결 풀링을 이용한 데이터베이스 연결과 해제
  - 수강신청 조회, 수강신청 삭제에 필요한 비즈니스 로직
    - Vector getEnrollList(String)
      - 현재 년도와 학기에 해당하는 해당 학생 수강신청 내역 조회
      - int getCurrentYear() : 현재 년도 얻어옴
      - int getCurrentSemester() : 현재 학기 얻어옴
    - Vector getEnrollList(String, int, int)
      - 년도, 학기 임의 지정에 따른 해당 학생 수강신청 내역 조회
    - void deleteEnroll(String, String, int)
      - 해당 학기, 과목번호, 분반에 해당하는 수강신청 내역 삭제

공통적으로 사용할 자바빈즈 작성

- Enroll.java
  - EnrollMgr 클래스에서 리턴되는 여러 개의 행 중 특정 한 행을 관리
  - void setXXX(XXX), XXXX getXXX()

EnrollMgr.java - EnrollMgr()

```
package enrollBean;

import java.sql.*;
import java.util.*;
import javax.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.pool.*;
import enrollBean.*;

public class EnrollMgr {
    private OracleConnectionPoolDataSource ocpds = null;
    private PooledConnection pool = null;

    public EnrollMgr() {
        try{
            ocpds = new OracleConnectionPoolDataSource();

            ocpds.setURL("jdbc:oracle:thin:@db.sd.ac.kr:1521:ora9");
            ocpds.setUser("db");    ocpds.setPassword("db");
            pool  = ocpds.getPooledConnection();
        }catch(Exception e){
            System.out.println("Error : 커넥션 얻어오기 실패");
        }
    }
```

EnrollMgr.java
- getEnrollList(String)

```
public Vector getEnrollList(String s_id) {
    Connection conn = null;   PreparedStatement pstmt = null;
    CallableStatement cstmt1 = null;  CallableStatement cstmt2 = null;
    ResultSet rs = null;
    Vector vecList = new Vector();

    try {
        conn = pool.getConnection();

        cstmt1 = conn.prepareCall("{? = call Date2EnrollYear(SYSDATE)}");
        cstmt1.registerOutParameter(1, java.sql.Types.INTEGER);
        cstmt1.execute();
        int nYear = cstmt1.getInt(1);

        cstmt2 = conn.prepareCall("{? = call Date2EnrollSemester(SYSDATE)}");
        cstmt2.registerOutParameter(1, java.sql.Types.INTEGER);
        cstmt2.execute();
        int nSemester = cstmt2.getInt(1);
```

```
        String  mySQL = “SELECT e.c_id cid,e.c_id_no cid_no,c.c_name cname,
        c.c_unit cunit FROM enroll e, course c WHERE e.s_id=? AND e.e_year=? AND
        e.e_semester=? and e.c_id=c.c_id and e.c_id_no=c.c_id_no";
         pstmt = conn.prepareStatement(mySQL);
         pstmt.setString(1,s_id);  pstmt.setInt(2, nYear);
        pstmt.setInt(3, nSemester);
         rs = pstmt.executeQuery();

         while (rs.next()) {
                 Enroll en = new Enroll();
                 en.setCId(rs.getString("cid"));
                en.setCIdNo(rs.getInt("cid_no"));
                en.setCName(rs.getString("cname"));
                en.setCUnit(rs.getInt("cunit"));
                vecList.add(en);
          }
           cstmt1.close();   cstmt2.close();   pstmt.close();
          conn.close();
    } catch (Exception ex) {
          System.out.println("Exception" + ex);
    }
    return vecList;
}
```

EnrollMgr.java
- getEnrollList(String, int, int)

```
public Vector getEnrollList(String s_id, int nYear, int nSemester) {
    Connection conn = null;
    PreparedStatement pstmt = null;
    ResultSet rs = null;
    Vector vecList = new Vector();

    try {
        conn = pool.getConnection();

        String mySQL = "SELECT e.c_id cid,e.c_id_no cid_no,c.c_name
        cname,c.c_unit cunit FROM enroll e, course c WHERE e.s_id=?
        AND e.e_year=? AND e.e_semester=? AND e.c_id=c.c_id AND
        e.c_id_no=c.c_id_no";
        pstmt = conn.prepareStatement(mySQL);
        pstmt.setString(1,s_id);
        pstmt.setInt(2, nYear);
        pstmt.setInt(3, nSemester);
        rs = pstmt.executeQuery();
```

```
        while (rs.next()) {
            Enroll en = new Enroll();
            en.setCId(rs.getString("cid"));
            en.setCIdNo(rs.getInt("cid_no"));
            en.setCName(rs.getString("cname"));
            en.setCUnit(rs.getInt("cunit"));
            vecList.add(en);
        }
        pstmt.close();
        conn.close();
    } catch (Exception ex) {
        System.out.println("Exception" + ex);
    }

    return vecList;
}
```

EnrollMgr.java - getCurrentYear()

```
public int getCurrentYear()
{
        int nYear=0;      Connection conn = null;
        CallableStatement cstmt = null;

        try {
            conn = pool.getConnection();

            cstmt = conn.prepareCall("{? = call Date2EnrollYear(SYSDATE)}");
            cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
            cstmt.execute();
            nYear = cstmt.getInt(1);

            cstmt.close();
            conn.close();
        } catch (Exception ex) {
            System.out.println("Exception" + ex);
        }
        return nYear;
}
```

EnrollMgr.java - getCurrentSemester()

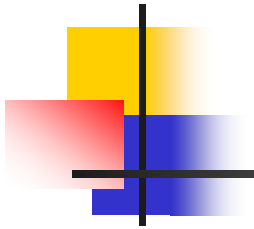
```
public int getCurrentSemester()
{
	int nSemester=0;
	Connection conn = null;	CallableStatement cstmt = null;

	try {
		conn = pool.getConnection();

		cstmt = conn.prepareCall("{? = call Date2EnrollSemester(SYSDATE)}");
		cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
		cstmt.execute();
		nSemester = cstmt.getInt(1);

		cstmt.close();
		conn.close();
	} catch (Exception ex) {
		System.out.println("Exception" + ex);
	}
	return nSemester;
}
```

EnrollMgr.java
- deleteEnroll(String, String, String)

```
public void deleteEnroll(String s_id, String c_id, int c_id_no) {
   Connection conn = null;
   PreparedStatement pstmt = null;

   try {
      conn = pool.getConnection();

      String mySQL = “DELETE FROM enroll WHERE s_id=? AND c_id=?
                              AND c_id_no=?";
      pstmt = conn.prepareStatement(mySQL);
      pstmt.setString(1,s_id);
      pstmt.setString(2, c_id);
      pstmt.setInt(3, c_id_no);
      pstmt.executeUpdate();

      pstmt.close();
      conn.close();
    } catch (Exception ex) {
      System.out.println("Exception" + ex);
    }
  }
}
```

Enroll.java

```
package enrollBean;

public class Enroll
{
    private String c_id;
    private int c_id_no;
    private String c_name;
    private int c_unit;

    public Enroll()
      { c_id = null;  c_id_no = 0;  c_name = null; c_unit = 0; }

    public void setCId(String c_id) { this.c_id = c_id; }

    public void setCIdNo(int c_id_no) { this.c_id_no = c_id_no; }
```



```
    public void setCName(String c_name) { this.c_name = c_name; }

    public void setCUnit(int c_unit) { this.c_unit = c_unit; }

    public String getCId() { return c_id; }

    public int getCIdNo() { return c_id_no; }

    public String getCName() { return c_name; }

    public int getCUnit() { return c_unit; }
}
```

JSP : 수강신청 삭제

- delete.jsp
  - 삭제할 과목 조회
    - EnrollMgr.getEnrollList(String) 사용
    - Enroll.getXXX()사용
  - 수강신청 삭제
    - delete_verify.jsp 호출
- delete_verify.jsp
  - 선택된 수강신청 과목 삭제
    - EnrollMgr.deleteEnroll(String,String,int) 사용

JSP : 수강신청 조회

- select.jsp
  - 년도, 학기가 선택이 안된 경우 현재날짜 기준 년도, 학기 얻어오기
    - EnrollMgr.getCurrentYear() 사용
    - EnrollMgr.getCurrentSemester() 사용
  - 입력되거나 얻어진 년도, 학기에 해당하는 수강신청 내역 조회
    - EnrollMgr.getEnrollList(String, int, int) 사용
    - Enroll.getXXX() 사용
  - 년도, 학기 조건에 해당하는 수강신청 내역 조회
- select_verify.jsp
  - 년도, 학기 검색조건 값을 select.jsp 에 보내줌

delete.jsp

```
<%@ page contentType="text/html; charset=euc-kr"%>
<%@ page import="java.util.*, enrollBean.*"%>
<html>
<head>
<title>수강신청 삭제</title></head>

<body>

<%@ include file="top.jsp" %>

<%
    if (session_id==null) response.sendRedirect("login.jsp");
%>

<table width="75%" align="center" border>
<tr><th>과목번호</th><th>분반</th><th>과목명</th>
    <th>학점</th><th>수강신청</th></tr>
<BR>
```

```
<jsp:useBean id="enrollMgr" class="enrollBean.EnrollMgr" />
<%
    Vector vlist = enrollMgr.getEnrollList(session_id);
    int counter = vlist.size();
    for(int i=0; i<vlist.size(); i++){
            Enroll en = (Enroll)vlist.elementAt(i);
%>
<tr>
<td align="center"><%=en.getCId()%></td>
<td align="center"><%=en.getCIdNo()%></td>
<td align="center"><%=en.getCName()%></td>
<td align="center"><%=en.getCUnit()%></td>
<td align="center"><a href="delete_verify.jsp?c_id=
  <%= en.getCId() %>&c_id_no=<%= en.getCIdNo() %>">삭제</a></td>
<%
    }
%>
</tr>
</table> </body> </html>
```

delete_verify.jsp

```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="enrollBean.*" %>

<html>
<head><title> 수강신청 삭제 </title></head>
<body>

<%
    String s_id = (String)session.getAttribute("user");
    String c_id = request.getParameter("c_id");
    int c_id_no = Integer.parseInt(request.getParameter("c_id_no"));
%>

<jsp:useBean id="enrollMgr" class="enrollBean.EnrollMgr" scope="page" />
<%
    enrollMgr.deleteEnroll(s_id,c_id,c_id_no);
%>
<script>
    alert("삭제되었습니다.");
    location.href="delete.jsp";
</script>
</form></body></html>
```

select.jsp

```
<%@ page contentType="text/html; charset=euc-kr"%>
<%@ page import="java.util.*, enrollBean.*"%>
<html>
<head>
<title>수강신청 조회</title><head>

<body>

<%@ include file="top.jsp" %>

<%
    if (session_id==null) response.sendRedirect("login.jsp");
%>

<table width="75%" align="center" border>
<tr><th>과목번호</th><th>분반</th><th>과목명</th><th>학점</th></tr>
<BR>
```

```
<jsp:useBean id="enrollMgr" class="enrollBean.EnrollMgr" />
<%
Vector vlist = null;
int year, semester;

if (request.getParameter("year") == null &&
    request.getParameter("semester")==null)
{
    year = enrollMgr.getCurrentYear();
    semester = enrollMgr.getCurrentSemester();
}
else
{
    year = Integer.parseInt(request.getParameter("year"));
    semester = Integer.parseInt(request.getParameter("semester"));
}

vlist = enrollMgr.getEnrollList(session_id,year,semester);
```

```
int counter = vlist.size();
int  totUnit = 0;

for(int i=0; i<vlist.size(); i++) {
     Enroll en = (Enroll)vlist.elementAt(i);
     totUnit += en.getCUnit();
%>

<tr>
<td align="center"><%=en.getCId()%></td>
<td align="center"><%=en.getCIdNo()%></td>
<td align="center"><%=en.getCName()%></td>
<td align="center"><%=en.getCUnit()%></td>

<%
   }
%>

</tr> </table><br>
```

```
<table width="75%" align="center" border>
<tr><th>총신청과목수</th> <td align="center"><%= counter %></td> </tr>
<tr><th>총신청학점</th><td align="center"><%= totUnit %></td></tr>
</table>
<br><br>
<table width="30%" align="center">
<FORM method="post" action="select_verify.jsp" >
  <tr>
   <td>
      <input type="text" name="year" value=<%= year %> size=4>년도
      <input type="text" name="semester" value=<%= semester %> size=1>
        학기
      <INPUT TYPE="SUBMIT" NAME="Submit" VALUE="조회">
    </td>
  </tr>
</table></FORM></body></html>
```

select_verify.jsp

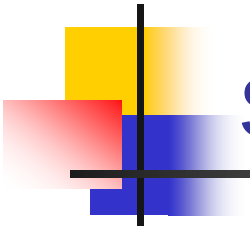
```
<%@ page contentType="text/html; charset=EUC-KR" %>
<%@ page import="java.sql.*" %>

<html>
<head><title> 수강신청 조회 </title></head>
<body>

<%
    String year = request.getParameter("year");
    String semester = request.getParameter("semester");

    String url = "select.jsp?year="+ year + "&semester=" + semester;

    response.sendRedirect(url);
%>
</body></html>
```

데이터베이스 고급프로그래밍 사례

JDBC Template,

JPA(Java Persistence API),

Spring Data JPA

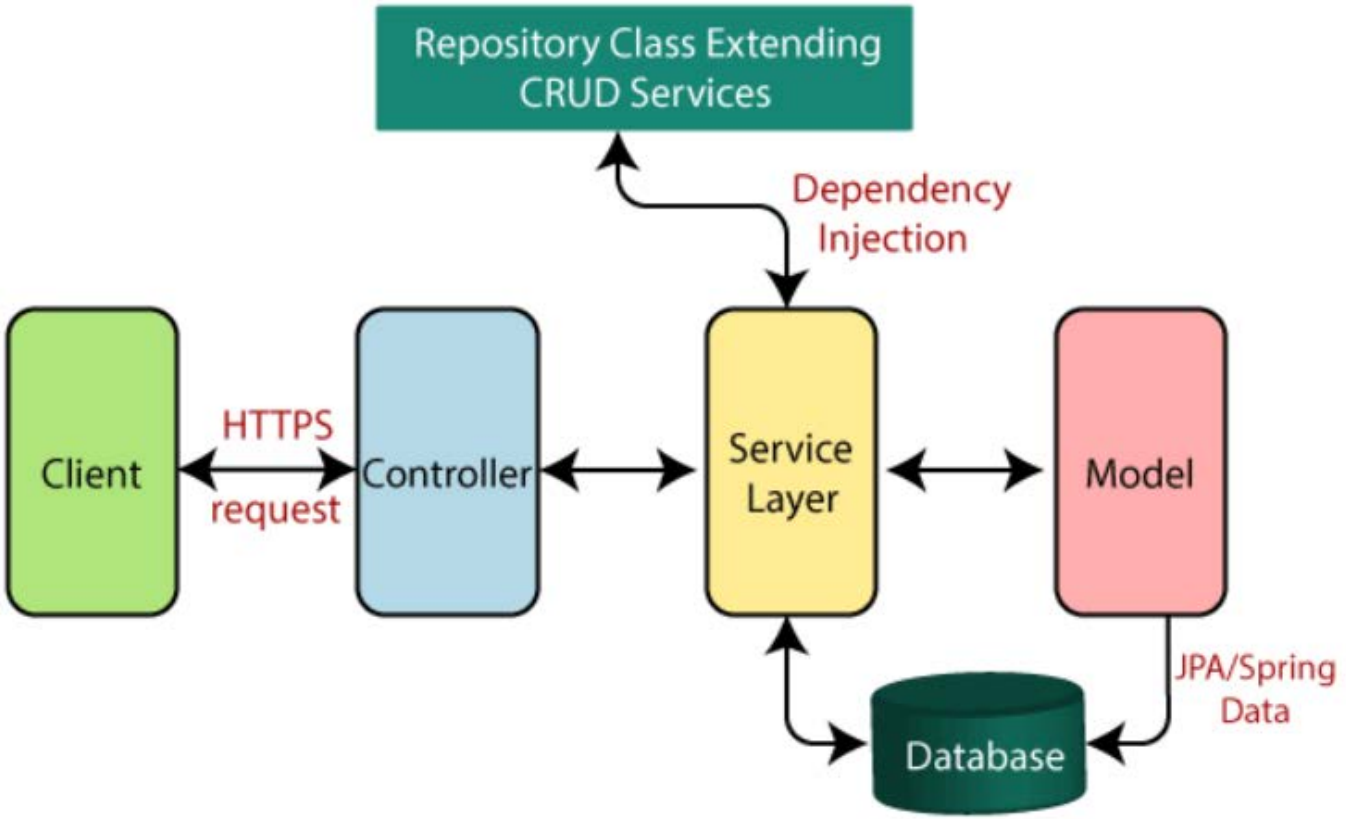


Spring Boot

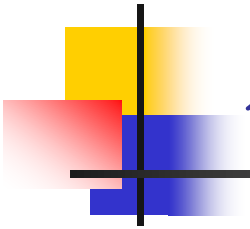
- Spring Boot
  - Spring Framework의 모듈, Spring 기반의 어플리케이션 개발
  - 최소의 구성으로 독립형 어플리케이션을 구축할 수 있음
  - 쉽고 빠르게 어플리케이션을 개발할 수 있음
- Spring Boot Architecture
  - Presentation Layer
  - Business Layer
  - Persistence Layer
  - Database Layer

Spring Boot

- Spring Boot Flow Architecture

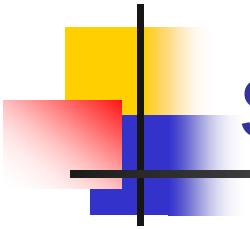


출처 : https://www.javatpoint.com/spring-boot-architecture



설치

- Systems, Languages and Tools Used in Class
  - DBMS : MySQL
  - Database Language : SQL, JPQL, QueryDSL
  - DB Connection : JDBC, JDBC Template, JPA, Spring Data JPA
  - Program Language : Java(JDK)
  - Framework : Spring Boot
  - IDE : Intellij
  - Testing : Postman, Chrome
  - Project Start : Spring Initializer



Spring Boot 프로젝트 생성

- Spring Initializer를 통해 프로젝트 생성
  - Project : Maven 또는 Gradle 선택
  - Language : Java
  - Dependencies
    - MySQL Driver
    - Spring Web
    - Lombok
    - Spring Data JPA
    - Spring Data JDBC

Spring Boot 설정

- build.gradle

```
plugins {
	id 'org.springframework.boot' version '2.7.1'
	id 'io.spring.dependency-management' version '1.0.11.RELEASE'
	id 'java'
}

group = 'dbp'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

configurations {
	compileOnly {
		extendsFrom annotationProcessor
	}
}

repositories {
	mavenCentral()
}
```

Spring Boot 설정

- build.gradle

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'mysql:mysql-connector-java'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'}

tasks.named('test') {
    useJUnitPlatform()
}
```

Spring Boot 설정

- application.properties

```
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
spring.datasource.url = jdbc:mysql://localhost:3306/dbp_univ  //데이터베이스명
spring.datasource.username = root              //MySQL 계정
spring.datasource.password = 1234              //MySQL 비밀번호
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
```

JDBC

UnivJdbc.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.*;

public class UnivJdbc {
    public static void main(String[] args) {

        String dbdriver = "com.mysql.cj.jdbc.Driver";
        Connection conn = null;
        Statement stmt = null;
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        String sql = "";
        String url ="jdbc:mysql://localhost:3306/dbp_univ";   //데이터베이스명
        String user = "root";              //MySQL 계정
        String passwd = "1234";        //MySQL 비밀번호

        try {
            Class.forName(dbdriver);
            conn = DriverManager.getConnection(url, user, passwd);
```

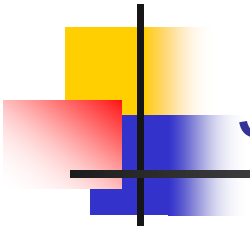
JDBC

UnivJdbc.java

```
            int dept_id = 10;

            System.out.println("Statement");    //Statement 객체 사용
            sql = "select id, name, did from student where did = ";
            stmt = conn.createStatement();
            rs = stmt.executeQuery(sql + dept_id);
            while (rs.next()) {
                System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getInt(3));
            }

            System.out.println("Prepared Statement");   //Prepared Statement 객체 사용
            sql = "select id, name, did from student where did =?";
            pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, dept_id);
            rs = pstmt.executeQuery();
            while (rs.next()) {
                System.out.println(rs.getInt(1) + " " + rs.getString(2) + " " + rs.getInt(3));
            }
            conn.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

실행 결과

```
Statement
1 홍길동 10
Prepared Statement
1 홍길동 10
```



JDBC Template

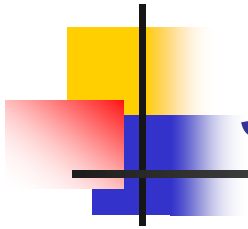
- JDBC Template
  - 기본적이며 가장 많이 쓰이는 spring JDBC 접근법
  - JDBC 코어 패키지의 중심 클래스
- JDBC Template 클래스
  - SQL 쿼리 실행
  - 구문과 저장된 프로시저 호출을 수정
  - ResultSet 인스턴스에 대한 반복을 실행함
  - 반환된 매개변수 값을 추출함
  - JDBC 예외를 발생시켜 더 일반적이고 유익하게 변환함

JDBC Template

- 예제 테이블 구조

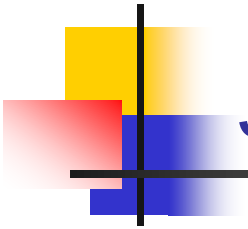
- CLUB 테이블

속성	의미
id	아이디
name	이름
category	분야



JDBC Template

- 예제 실습
  - Club 테이블 전체 검색
  - Club 테이블에서 id로 검색(조건 검색)
  - Club 테이블에 데이터 삽입(행 삽입)
  - Club 테이블에서 해당 id의 데이터 삭제(행 삭제)



JDBC Template

- entity
  - Club.java
- repository
  - ClubRepository.java
- controller
  - ClubController.java

entity

Club.java

```
package dbp.UnivJdbcTemplate.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Club {
    @Id     // 기본키 매핑
    @GeneratedValue(strategy = GenerationType.IDENTITY)  // 기본키 생성
    private int id;
    private String name;
    private String category;

    public Club(){
    }

    public Club(int id, String name, String category){
        this.id = id;
        this.name = name;
        this.category = category;
    }
```

entity

Club.java

```
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }
}
```

repository

ClubRepository.java

```
package dbp.UnivJdbcTemplate.repository;

import odbp.UnivJdbcTemplate.entity.Club;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public class ClubRepository {

    @Autowired
    JdbcTemplate jdbcTemplate;

    public List<Club> getAllClubs() {        // club 테이블 전체 검색
        List<Club> clubs = jdbcTemplate.query("select id, name, category from club",
      (result, rowNum) -> new Club(result.getInt("id"), result.getString("name"),
      result.getString("category")));
        return clubs;
    }
```

repository

ClubRepository.java

```
    public Optional<Club> getClub(int id) {              // id로 검색
        List<Club> result = jdbcTemplate.query("select * from club where id = ?",
      clubRowMapper(), id);
        return result.stream().findAny();
    }

    public int addClub(int id,String name,String category){       // 데이터 삽입
        String query = "insert into club values (?,?,?)";
        return jdbcTemplate.update(query,id,name,category);
    }

    public int deleteClub(int id){                // 데이터 삭제
        String query = "delete from club where ID =?";
        return jdbcTemplate.update(query,id);
    }

    private RowMapper<Club> clubRowMapper() {
        return (rs, rowNum) -> {
            Club club = new Club();
            club.setId(rs.getInt("id"));
            club.setName(rs.getString("name"));
            club.setCategory(rs.getString("category"));
            return club;
        };
    }
}
```

controller

ClubController.java

```
package dbp.UnivJdbcTemplate.controller;

import dbp.UnivJdbcTemplate.entity.Club;
import dbp.UnivJdbcTemplate.repository.ClubRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/")
public class ClubController {

    @Autowired
    ClubRepository clubRepository;

    @GetMapping("/all")          // club 테이블 전체 검색
    public List<Club> getAllClubs() {
        return clubRepository.getAllClubs();
    }
```

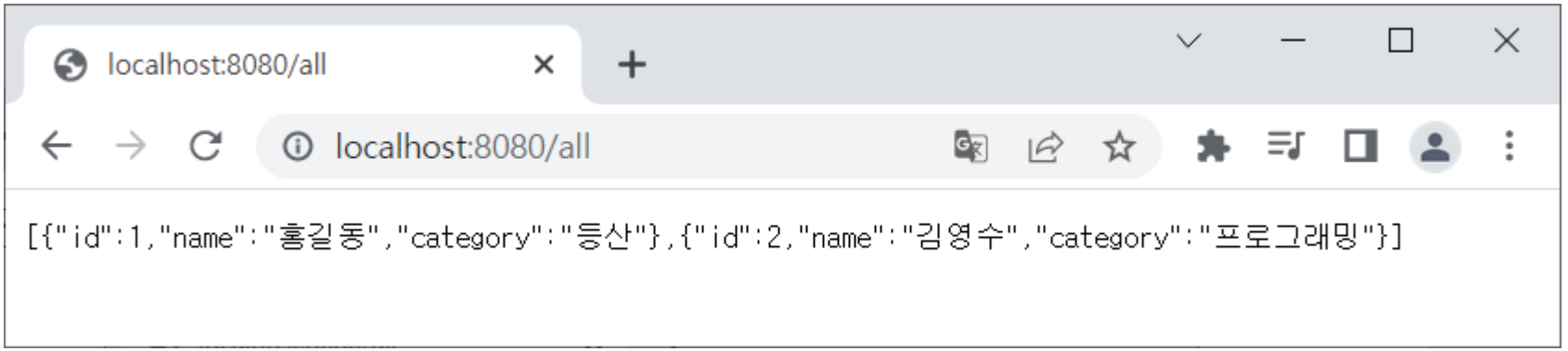
controller

ClubController.java

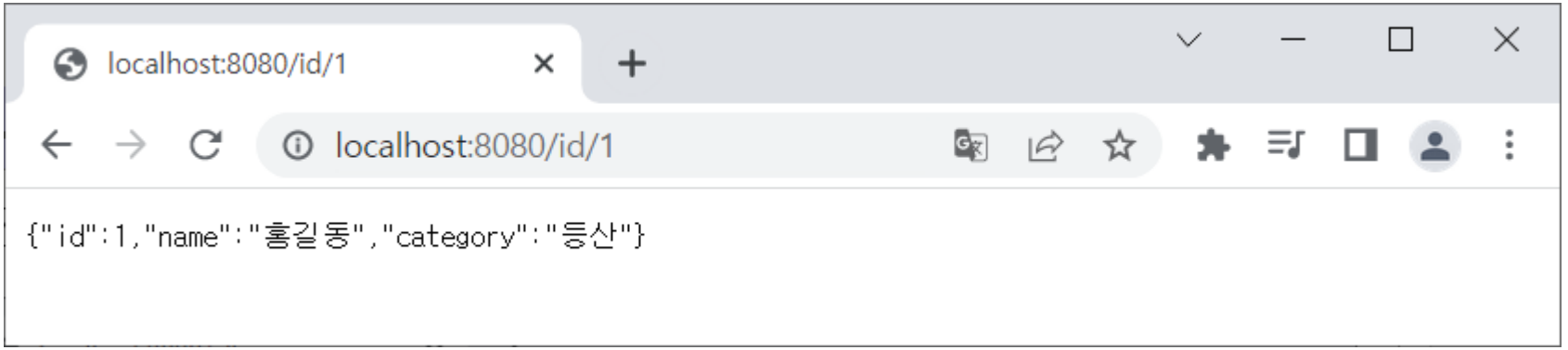
```
    @GetMapping("/id/{id}")       // id로 검색
    public Optional<Club> getClub(@PathVariable int id) {
        return clubRepository.getClub(id);
    }

    @GetMapping("/addClub")                  // 데이터 삽입
    public String addClub(@RequestParam("id") int id, @RequestParam("name")
      String name, @RequestParam("category") String category) {
        if (clubRepository.addClub(id, name, category) >= 1) {
            return "Club Added";
        }else {
            return "Not Added";
        }
    }

    @DeleteMapping("/delete/{id}")          // 데이터 삭제
    public String deleteClub(@PathVariable int id) {
        if (clubRepository.deleteClub(id) >= 1) {
            return "Club Deleted";
        }else {
            return "Not Deleted";
        }
    }
}
```

실행 결과

- 전체 조회 - @GetMapping("/all")

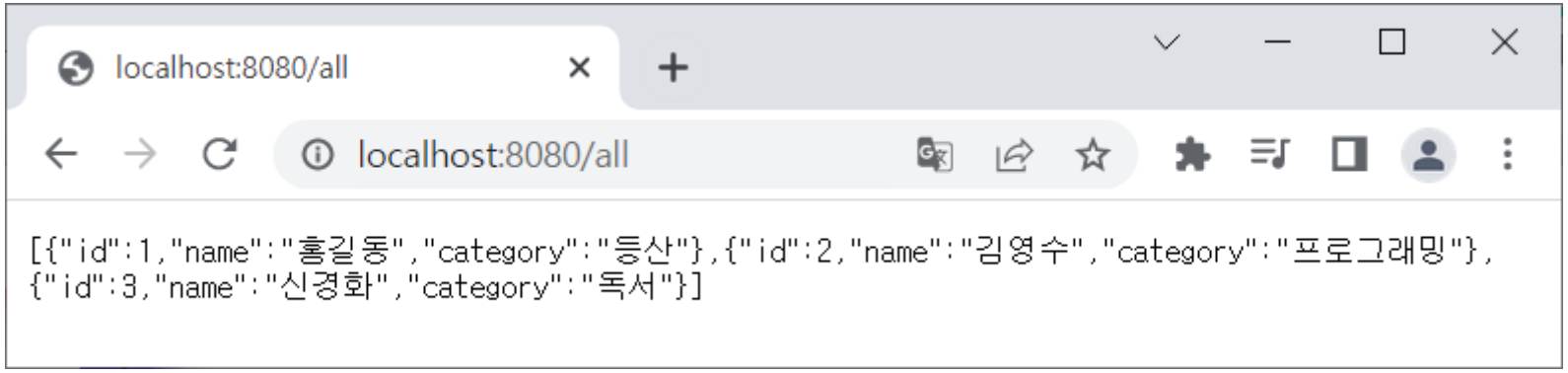


- 일부 조회 - @GetMapping("/id/{id}")



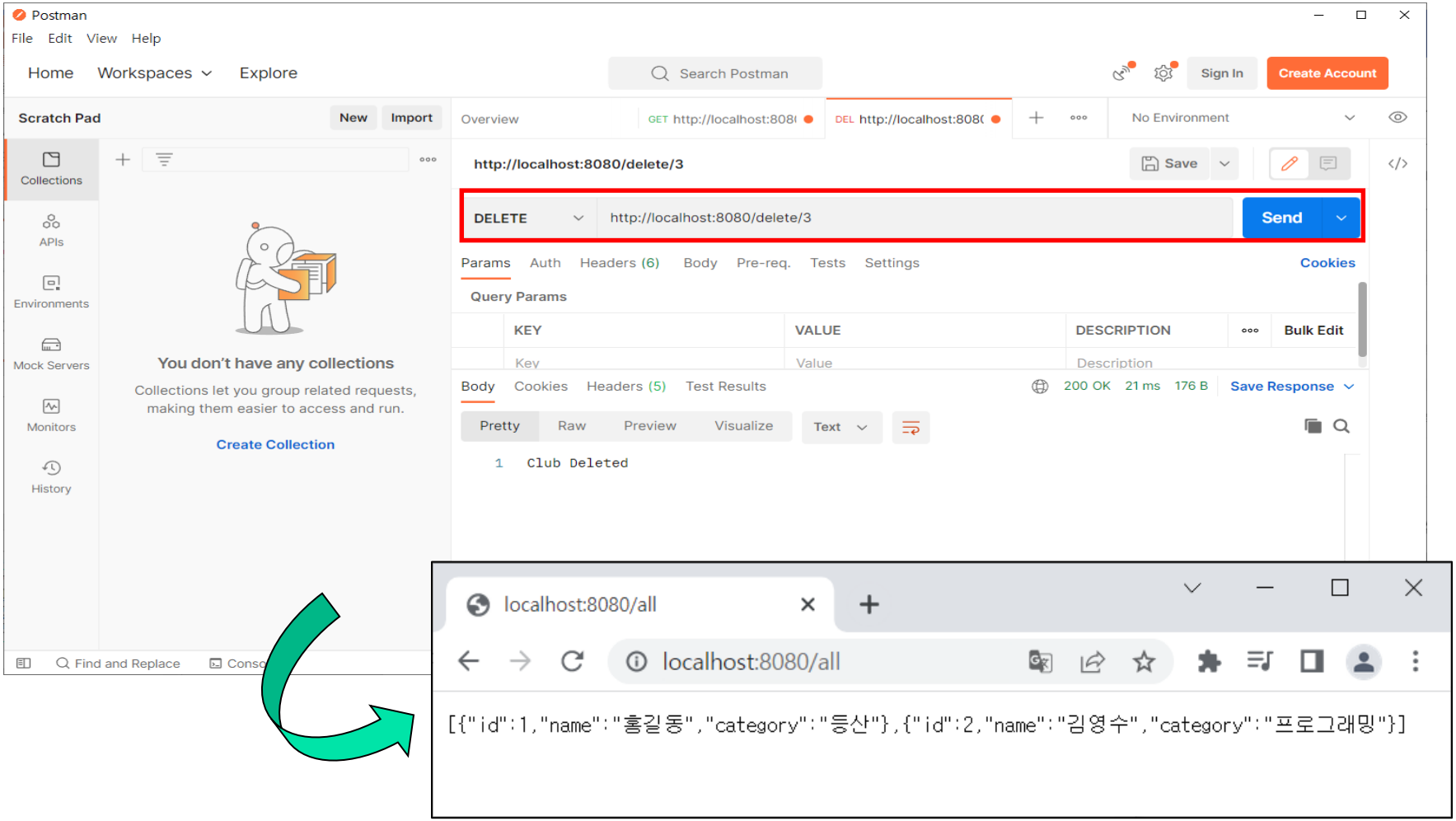
실행 결과

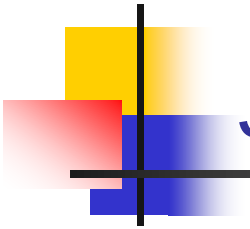
- 삽입 - @GetMapping("/addClub")



실행 결과

- 삭제 - @DeleteMapping("/delete/{id}")





JPQL

- JPQL(Java Persistence Query Language)
  - 엔티티 객체를 조회하는 객체지향 쿼리
  - SQL을 추상화하여 특정 데이터베이스에 의존하지 않음
  - SQL보다 코드가 간결함
  - JPQL을 분석하여 적절한 SQL로 변환하여 데이터베이스를 조회
  - 엔티티 이름과 엔티티 객체의 필드명으로 작성

JPQL

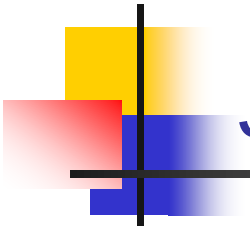
- 예제 테이블 구조

STUDENT 테이블

속성	의미
id	학번
name	이름
address	주소
did	전공번호

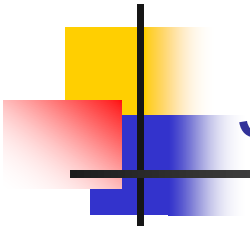
DEPT 테이블

속성	의미
id	전공번호
name	전공이름
building	위치



JPQL

- 예제 실습
  - Dept 테이블에서 name으로 검색(조건 검색)
  - Dept 테이블에서 building으로 검색(조건 검색)
  - Student 테이블에서 id로 검색(조건 검색)
  - Student 테이블에서 name으로 검색(조건 검색)
  - Student 테이블에서 해당 id의 address 수정(행 수정)
  - Student 테이블에서 해당 id의 데이터 삭제(행 삭제)
  - Student 테이블에 데이터 삽입(행 삽입)
  - Student 테이블과 Dept 테이블 전체 검색(조인 검색)



JPQL

- entity
  - Dept.java, Student.java
- repository
  - DeptRepository.java, StudentRepository.java
- controller
  - DeptController.java, StudentController.java
- service
  - DeptService.java, StudentService.java

entity

Dept.java

```
package dbp.UnivJpql.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Dept {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String building;

}
```

entity

Student.java

```
package dbp.UnivJpql.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    @Id
    private Long id;
    private String name;
    private String address;

    @ManyToOne(targetEntity = Dept.class)          // 다대일 매핑(N:1)
    @JoinColumn(name = "did", foreignKey = @ForeignKey(name = "fk_student_did"))
    private Dept dept;

}
```

repository

DeptRepository.java

```
package dbp.UnivJpql.repository;

import dbp.UnivJpql.entity.Dept;
import dbp.UnivJpql.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import javax.swing.text.html.Option;
import java.util.List;
import java.util.Optional;

public interface DeptRepository extends JpaRepository<Dept, Long> {

    @Query(value = "from Dept where name=?1")   // name으로 검색
    public Optional<Dept> findDeptByName(String name);

    @Query(value = "from Dept where building=?1“)   // building으로 검색
    public List<Dept> findDeptByBuilding(String building);

}
```

repository

StudentRepository.java

```
package dbp.UnivJpql.repository;

import dbp.UnivJpql.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

public interface StudentRepository extends JpaRepository<Student, Long> {

    @Query(value = "from Student where id = ?1")  // id로 검색
    public Optional<Student> findStudentById(Long id);

    @Query(value = "from Student where name = ?1")  // name으로 검색
    public List<Student> findStudentByName(String name);

    @Modifying         // address 수정
    @Query(value = "update Student set address = ?2 where id = ?1", nativeQuery = true)
    @Transactional
    public void updateStudentAddress(Long id, String address);
```

repository

StudentRepository.java

```
    @Modifying        // 데이터 삭제
    @Query(value = "delete from Student where id = ?1")
    @Transactional
    public void deleteStudentById(Long id);

    @Modifying        // 데이터 삽입
    @Query(value = "insert into Student(id, name, address, did) values(?1, ?2, ?3, ?4)",
      nativeQuery = true)
    @Transactional
    public void insertStudent(Long id, String name, String address, Long did);

    @Query(value = "select s from Student s join fetch s.dept")          // 전체 조회
    public List<Student> findAll();

}
```

controller

DeptController.java

```
package dbp.UnivJpql.controller;

import dbp.UnivJpql.entity.Dept;
import dbp.UnivJpql.service.DeptService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/")
public class DeptController {

    @Autowired
    DeptService deptService;
```

controller

DeptController.java

```
    @GetMapping("/dept/name/{name}")    // name으로 검색
    public Optional<Dept> findByDeptName(@PathVariable String name) {
        return deptService.findDeptByName(name);
    }

    @GetMapping("/dept/building/{building}")    // building으로 검색
    public List<Dept> findByDeptBuilding(@PathVariable String building) {
        return deptService.findDeptByBuilding(building);
    }

}
```

controller

StudentController.java

```
package dbp.UnivJpql.controller;

import dbp.UnivJpql.entity.Student;
import dbp.UnivJpql.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/")
public class StudentController {
    @Autowired
    StudentService studentService;

    @GetMapping("/student/id/{id}")         // id로 검색
    public Optional<Student> findStudentById(@PathVariable Long id) {
        return studentService.findStudentById(id);
    }

    @GetMapping("/student/name/{name}")           // name으로 검색
    public List<Student> findStudentByName(@PathVariable String name) {
        return studentService.findStudentByName(name);
    }
```

controller

StudentController.java

```
    @PutMapping("/update/{id}/{address}")              // address 수정
    public String updateStudentAddress(@PathVariable Long id, @PathVariable
      String address) {
        studentService.updateStudentAddress(id, address);
        return "updated";
    }

    @DeleteMapping("/delete/{id}")          // 데이터 삭제
    public String deleteStudentById(@PathVariable Long id) {
        studentService.deleteStudentById(id);
        return "deleted";
    }

    @PostMapping("/student/insert")        // 데이터 삽입
    public String insertStudent(@RequestBody Student student) {
        studentService.insertStudent(student.getId(), student.getName(),
      student.getAddress(), student.getDept().getId());
        return "inserted";
    }

    @GetMapping("/student/all")              // 전체 조회
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }
}
```

service

DeptService.java

```
package dbp.UnivJpql.service;

import dbp.UnivJpql.entity.Dept;
import dbp.UnivJpql.repository.DeptRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class DeptService {

    @Autowired
    DeptRepository deptRepository;

    public Optional<Dept> findDeptByName(String name) {
        return deptRepository.findDeptByName(name);
    }

    public List<Dept> findDeptByBuilding(String building) {
        return deptRepository.findDeptByBuilding(building);
    }

}
```

service

StudentService.java

```
package dbp.UnivJpql.service;

import dbp.UnivJpql.entity.Student;
import dbp.UnivJpql.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class StudentService {

    @Autowired
    StudentRepository studentRepository;

    public Optional<Student> findStudentById(Long id) {
        return studentRepository.findStudentById(id);
    }

    public List<Student> findStudentByName(String name) {
        return studentRepository.findStudentByName(name);
    }
```

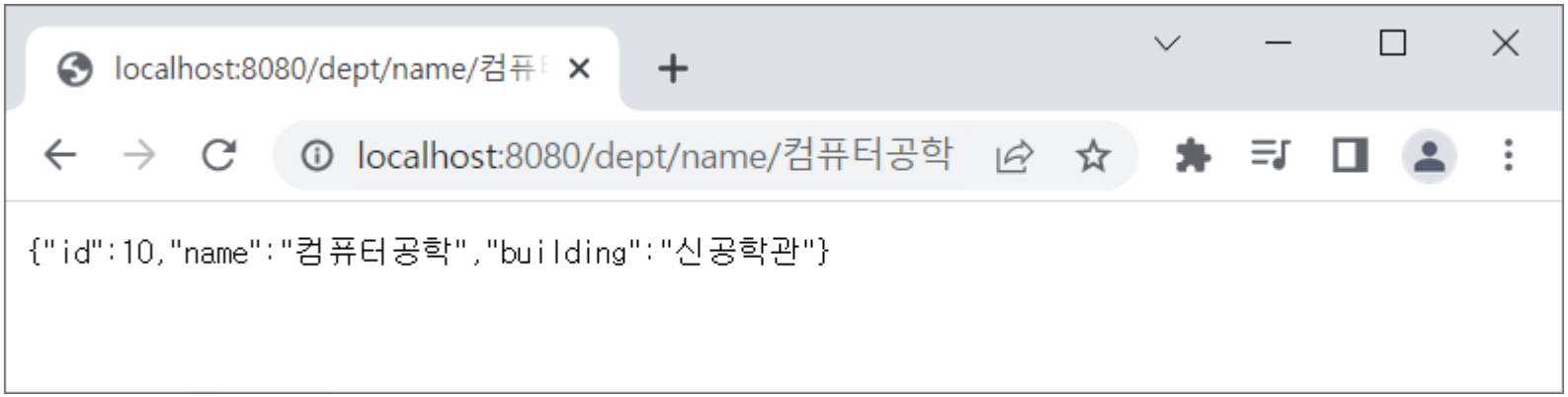
service

StudentService.java

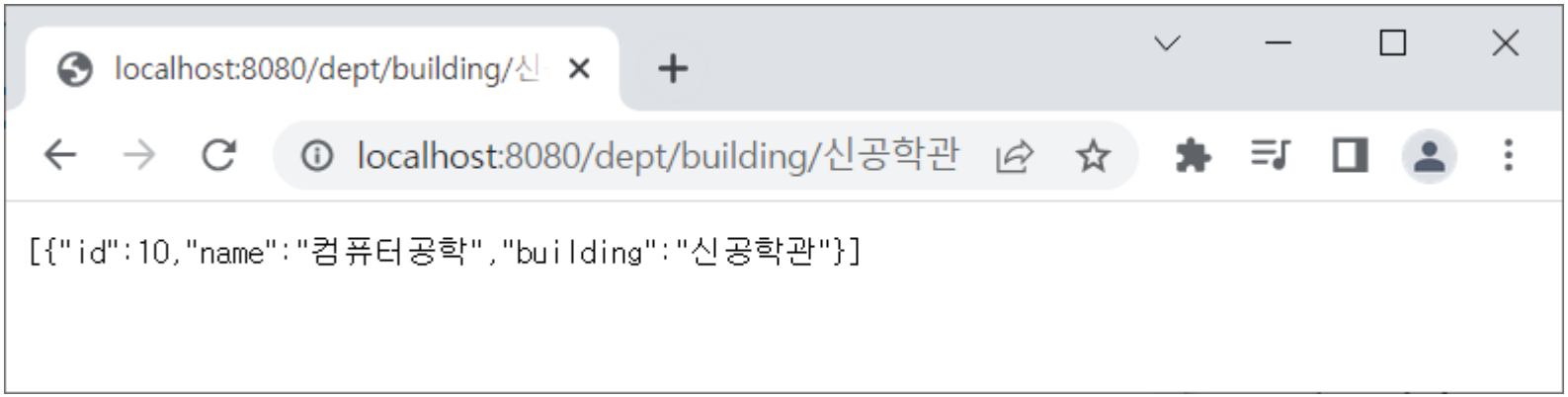
```
    public void updateStudentAddress(Long id, String address) {
        studentRepository.updateStudentAddress(id, address);
    }

    public void deleteStudentById(Long id) {
        studentRepository.deleteStudentById(id);
    }

    public void insertStudent(Long id, String name, String address, Long did) {
        studentRepository. insertStudent(id, name, address, did);
    }

    public List<Student> getAllStudents() {
        return studentRepository.findAll();
    }

}
```

실행 결과

- 부서명(name) 조회 - @GetMapping("/dept/name/{name}")

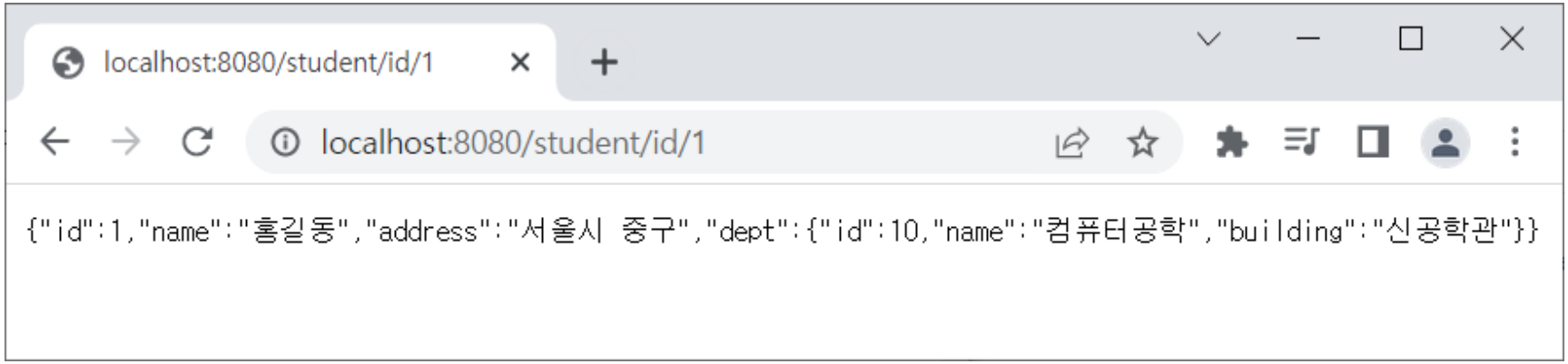


- 부서위치(building) 조회 - @GetMapping("/dept/building/{building}")

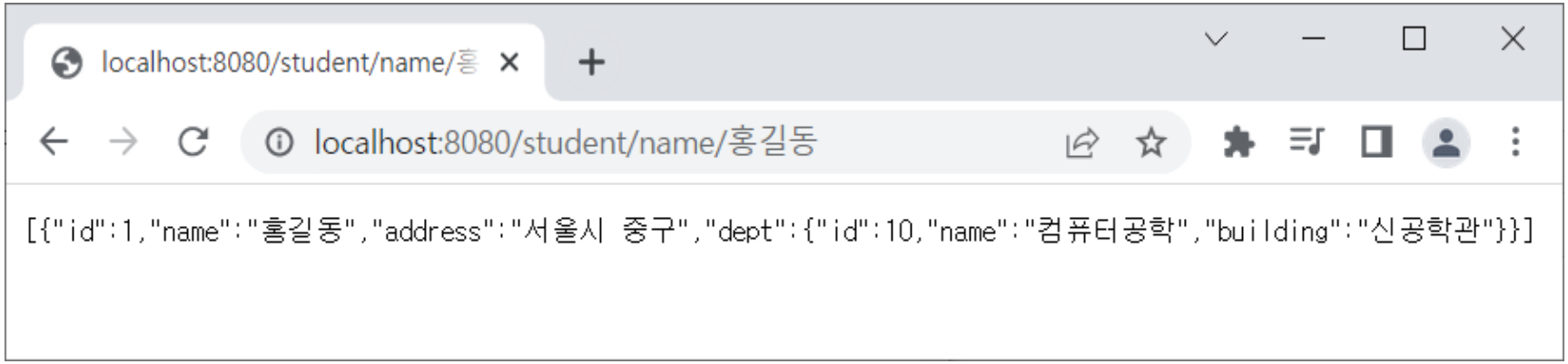


실행 결과

- 학번(id) 조회 - @GetMapping("/student/id/{id}")

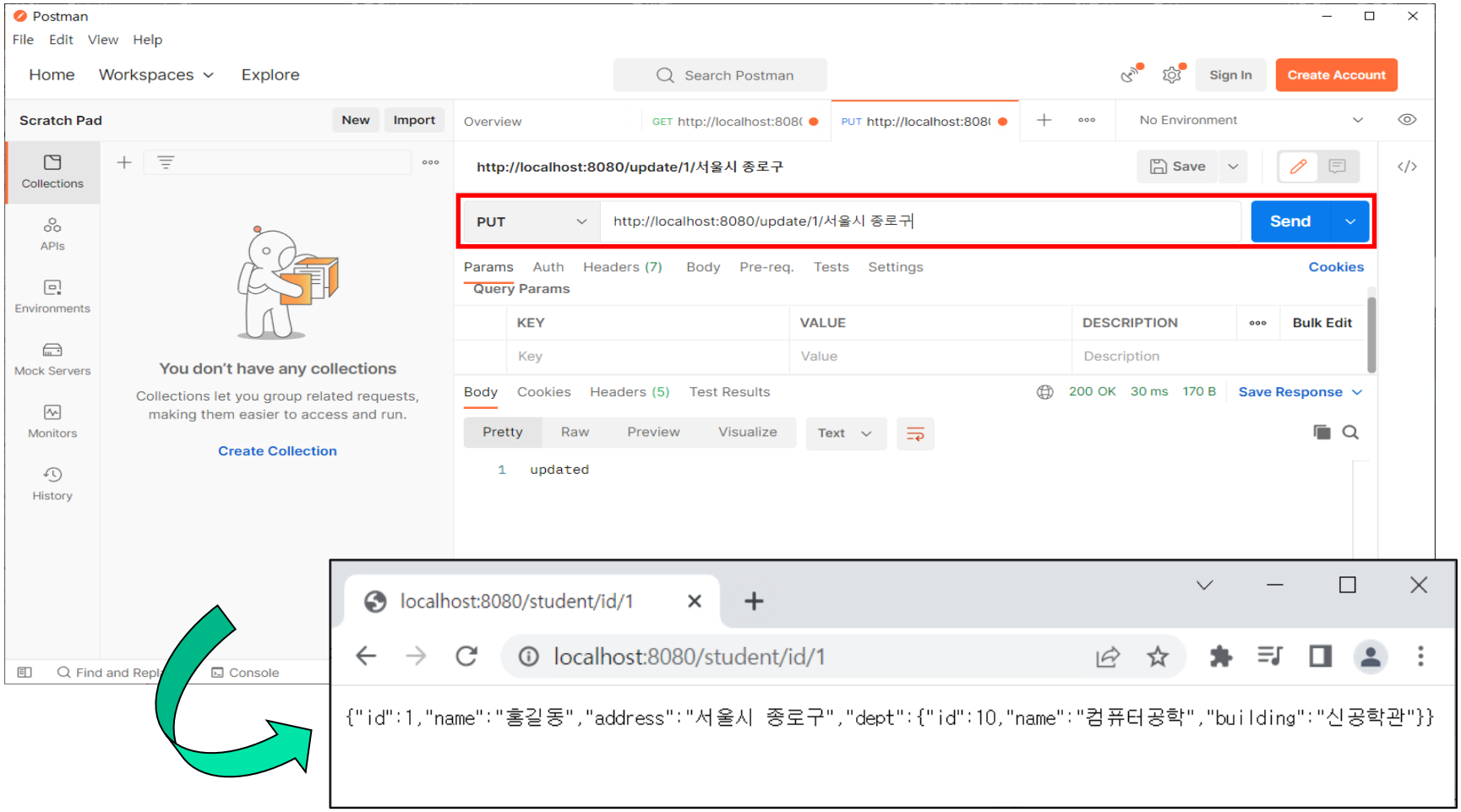


- 학생이름(name) 조회 - @GetMapping("/student/name/{name}")



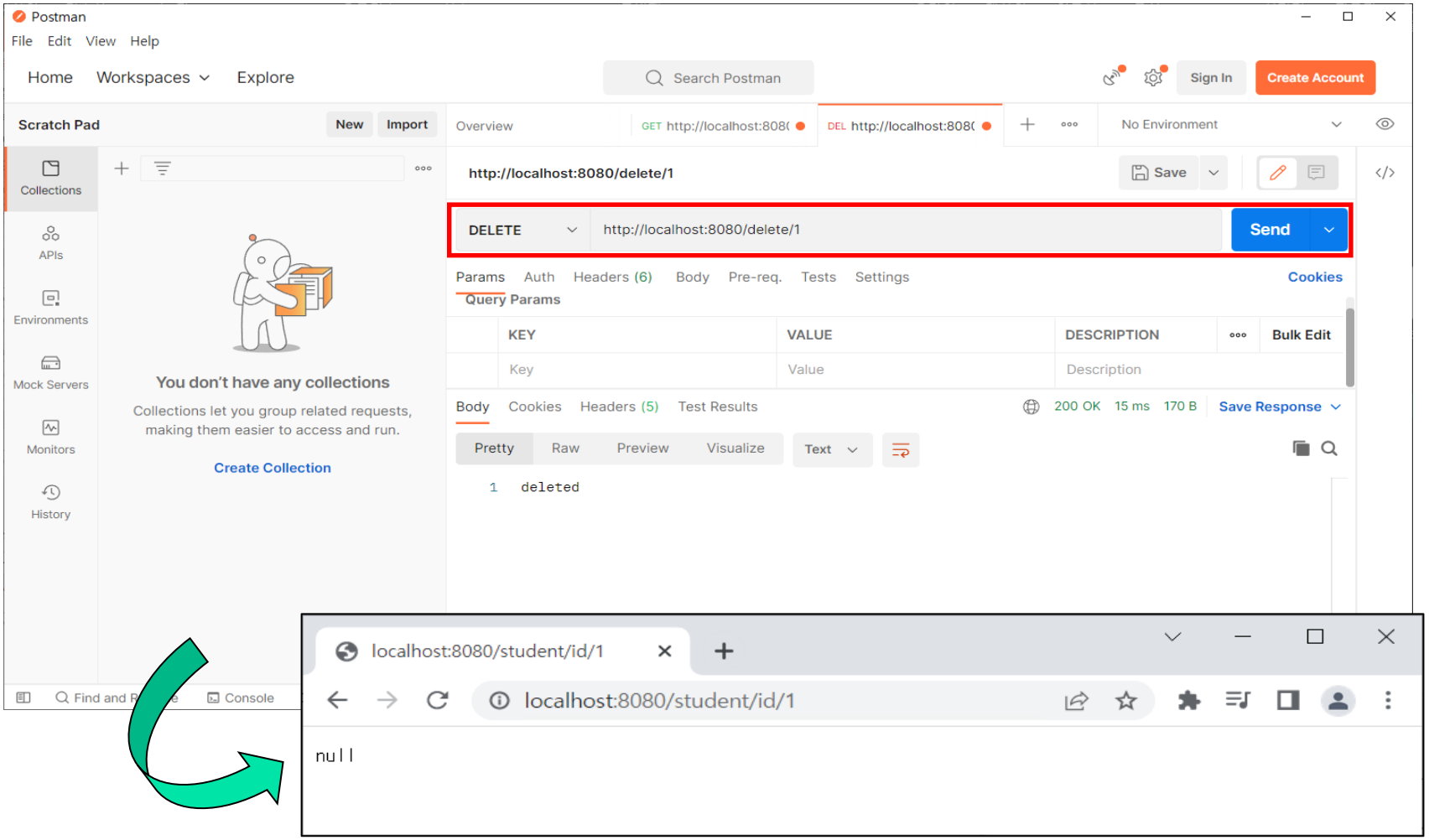
실행 결과

- 수정 - @PutMapping("/update/{id}/{address}")



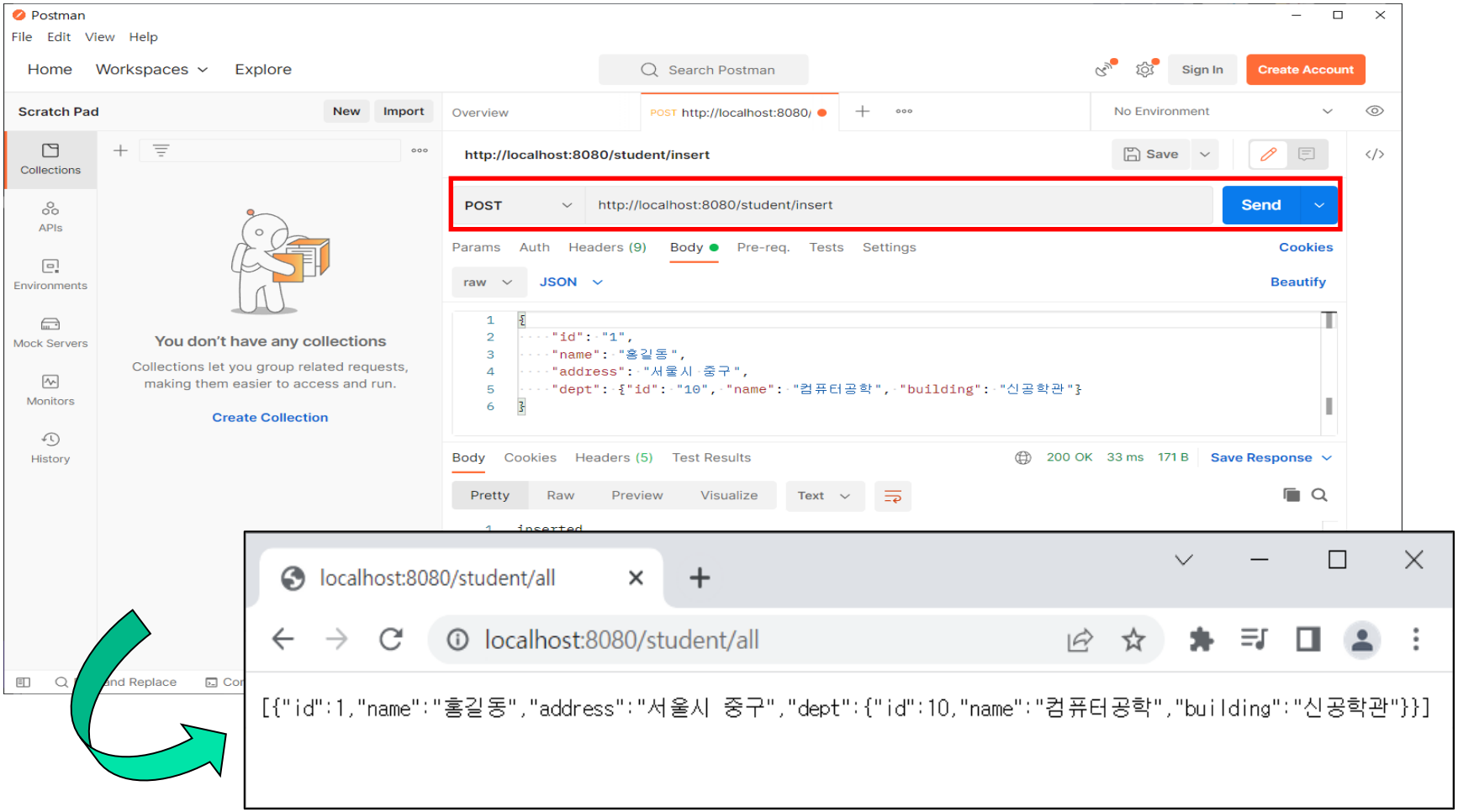
실행 결과

- 삭제 - @DeleteMapping("/delete/{id}")



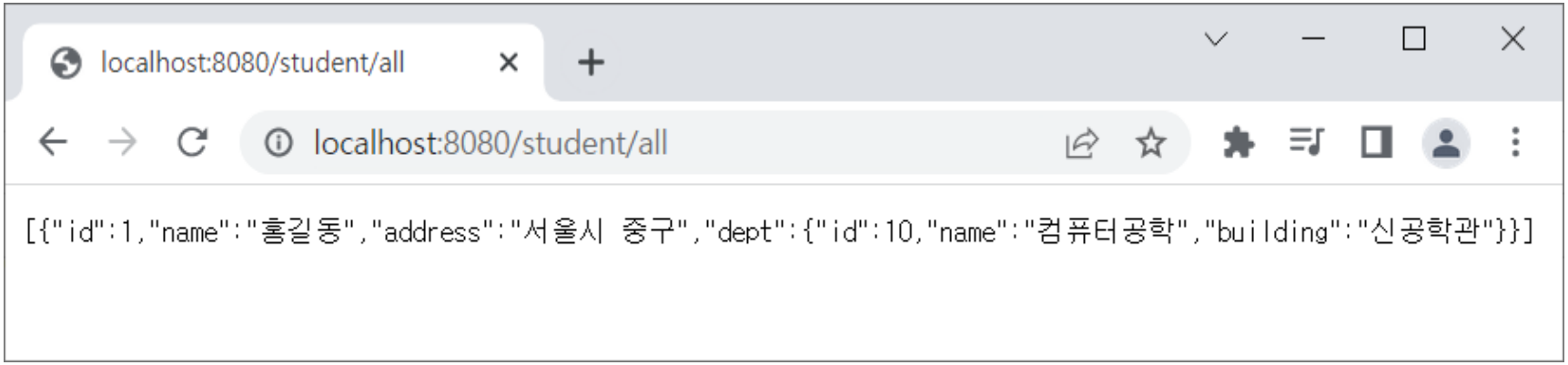
실행 결과

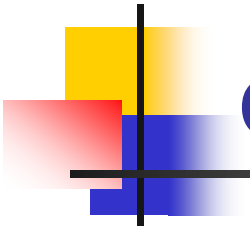
- 삽입 - @PostMapping("/student/insert")



실행 결과

- 전체 조회 - @GetMapping("/student/all")





QueryDSL

- QueryDSL
  - 정적 타입을 이용해서 SQL 등의 쿼리를 생성해주는 오픈소스 프레임워크
  - 코드 기반으로 단순하고 사용하기 쉬움
  - 컴파일 시점에 문법 오류를 쉽게 확인할 수 있음
  - 자동 완성 등 IDE의 도움을 받을 수 잇음
  - 동적 쿼리 작성이 편리함
  - 쿼리 작성 시 제약 조건 등을 메서드 추출을 통해 재사용 가능함

QueryDSL

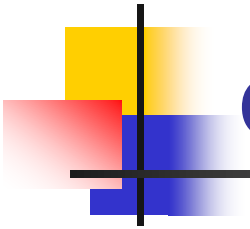
- 예제 테이블 구조

STUDENT 테이블

속성	의미
id	학번
name	이름
address	주소
did	전공번호

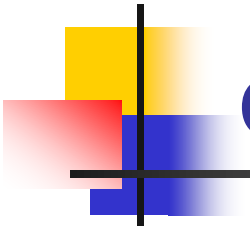
DEPT 테이블

속성	의미
id	전공번호
name	전공이름
building	위치



QueryDSL

- 예제 실습
  - Dept 테이블에서 name으로 검색(조건 검색)
  - Dept 테이블에서 building으로 검색(조건 검색)
  - Student 테이블에서 id로 검색(조건 검색)
  - Student 테이블에서 name으로 검색(조건 검색)
  - Student 테이블에서 해당 id의 address 수정(행 수정)
  - Student 테이블에서 해당 id의 데이터 삭제(행 삭제)
  - Student 테이블에 데이터 삽입(행 삽입)
  - Student 테이블과 Dept 테이블 전체 검색(조인 검색)



QueryDSL

- entity
  - Dept.java, Student.java
- repository
  - DeptRepository.java, StudentRepository.java
    - impl
      - DeptRepositoryImpl.java, StudentRepositoryImpl.java
- controller
  - DeptController.java, StudentController.java
- service
  - DeptService.java, StudentService.java
- config
  - UnivQuerydslConfiguration.java

Spring boot 추가 설정

- build.gradle

```
buildscript {
	dependencies {
		classpath("gradle.plugin.com.ewerk.gradle.plugins:querydsl-plugin:1.0.10")
	}
}

plugins {
	id 'org.springframework.boot' version '2.7.1'
	id 'io.spring.dependency-management' version '1.0.11.RELEASE'
	id 'java'
}

group = 'dbp'
version = '0.0.1-SNAPSHOT'
sourceCompatibility = '1.8'

apply plugin: "com.ewerk.gradle.plugins.querydsl"
```

Spring boot 추가 설정

- build.gradle

```
configurations {
	compileOnly {
		extendsFrom annotationProcessor
	}
}
repositories {
	mavenCentral()
}

dependencies {
	implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'
	implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
	implementation 'org.springframework.boot:spring-boot-starter-web'
	compileOnly 'org.projectlombok:lombok'
	runtimeOnly 'mysql:mysql-connector-java'
	annotationProcessor 'org.projectlombok:lombok'
	testImplementation 'org.springframework.boot:spring-boot-starter-test'

	implementation "com.querydsl:querydsl-jpa"
	implementation "com.querydsl:querydsl-apt"
}
```

Spring boot 추가 설정

- build.gradle

```
tasks.named('test') {
	useJUnitPlatform()
}

def querydslDir = "$buildDir/generated/querydsl"
querydsl {
	library = "com.querydsl:querydsl-apt"
	jpa = true
	querydslSourcesDir = querydslDir
}
sourceSets {
	main.java.srcDir querydslDir
}
compileQuerydsl{
	options.annotationProcessorPath = configurations.querydsl
}
configurations {
	querydsl.extendsFrom compileClasspath
}
```

entity

Dept.java

```
package dbp.UnivQuerydsl.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Dept {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String building;

}
```

entity

Student.java

```
package dbp.UnivQuerydsl.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {

    @Id
    private Long id;
    private String name;
    private String address;

    @ManyToOne(targetEntity = Dept.class)
    @JoinColumn(name = "did", foreignKey = @ForeignKey(name = "fk_student_did"))
    private Dept dept;

}
```

repository

DeptRepository.java

```
package dbp.UnivQuerydsl.repository;

import dbp.UnivQuerydsl.entity.Dept;
import dbp.UnivQuerydsl.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
import java.util.Optional;

public interface DeptRepository extends JpaRepository<Dept, Long> {

    public Optional<Dept> findDeptByName(String name);

    public List<Dept> findDeptByBuilding(String building);

}
```

repository

StudentRepository.java

```
package dbp.UnivQuerydsl.repository;

import dbp.UnivQuerydsl.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

public interface StudentRepository extends JpaRepository<Student, Long> {
    public Optional<Student> findStudentById(Long id);

    public List<Student> findStudentByName(String name);

    @Modifying
    @Transactional
    public void updateStudentAddress(Long id, String address);

    @Modifying
    @Transactional
    public void deleteStudentById(Long id);

    public List<Student> getAllStudents();
}
```

repository-impl

DeptRepositoryImpl.java

```
package dbp.UnivQuerydsl.repository.impl;

import com.querydsl.jpa.impl.JPAQueryFactory;
import dbp.UnivQuerydsl.entity.Dept;
import org.springframework.data.jpa.repository.support.QuerydslRepositorySupport;
import org.springframework.stereotype.Repository;

import java.util.List;

import static dbp.UnivQuerydsl.entity.QDept.dept;

@Repository
public class DeptRepositoryImpl extends QuerydslRepositorySupport {

    private final JPAQueryFactory jpaQueryFactory;

    public DeptRepositoryImpl(JPAQueryFactory jpaQueryFactory) {
        super(Dept.class);
        this.jpaQueryFactory = jpaQueryFactory;
    }
```

repository-impl

DeptRepositoryImpl.java

```
    public Dept findDeptByName(String name) {      // name으로 조회
        return jpaQueryFactory
                .selectFrom(dept)
                .where(dept.name.eq(name))
                .fetchOne();
    }

    public List<Dept> findDeptByBuilding(String building) {      // building으로 조회
        return jpaQueryFactory
                .selectFrom(dept)
                .where(dept.building.eq(building))
                .fetch();
    }

}
```

repository-impl

StudentRepositoryImpl.java

```
package dbp.UnivQuerydsl.repository.impl;

import com.querydsl.jpa.impl.JPAQueryFactory;
import dbp.UnivQuerydsl.entity.Student;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.support.QuerydslRepositorySupport;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import static dbp.UnivQuerydsl.entity.QDept.dept;
import static dbp.UnivQuerydsl.entity.QStudent.student;

@Repository
public class StudentRepositoryImpl extends QuerydslRepositorySupport {
    private final JPAQueryFactory jpaQueryFactory;

    public StudentRepositoryImpl(JPAQueryFactory jpaQueryFactory) {
        super(Student.class);
        this.jpaQueryFactory = jpaQueryFactory;
    }
```

repository-impl

StudentRepositoryImpl.java

```
public Student findStudentById(Long id) {          // id로 조회
    return jpaQueryFactory
            .selectFrom(student)
            .where(student.id.eq(id))
            .fetchOne();
}

public List<Student> findStudentByName(String name) {    // name으로 조회
    return jpaQueryFactory
            .selectFrom(student)
            .where(student.name.eq(name))
            .fetch();
}

public void updateStudentAddress(Long id, String address) {   // address 수정
    long execute = jpaQueryFactory
            .update(student)
            .set(student.address, address)
            .where(student.id.eq(id))
            .execute();
}
```

repository-impl

StudentRepositoryImpl.java

```
    public void deleteStudentById(Long id){              // 데이터 삭제
        long execute = jpaQueryFactory
                .delete(student)
                .where(student.id.eq(id))
                .execute();
    }

    public List<Student> getAllStudents() {              // 전체 조회
        return jpaQueryFactory
                .select(student)
                .distinct()
                .from(student)
                .innerJoin(student.dept,dept)
                .fetchJoin().fetch();
    }

}
```

controller

DeptController.java

```
package dbp.UnivQuerydsl.controller;

import dbp.UnivQuerydsl.entity.Dept;
import dbp.UnivQuerydsl.service.DeptService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/dbp")
public class DeptController {

    @Autowired
    DeptService deptService;
```

controller

DeptController.java

```
    @GetMapping("/dept/name/{name}")  // name으로 조회
    public Optional<Dept> findByDeptName(@PathVariable String name) {
        return deptService.findDeptByName(name);
    }

    @GetMapping("/dept/building/{building}")   // building으로 조회
    public List<Dept> findByDeptBuilding(@PathVariable String building) {
        return deptService.findDeptByBuilding(building);
    }

}
```

controller

StudentController.java

```
package dbp.UnivQuerydsl.controller;

import dbp.UnivQuerydsl.entity.Student;
import dbp.UnivQuerydsl.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/")
public class StudentController {
    @Autowired
    StudentService studentService;

    @GetMapping("/student/id/{id}")          // id로 조회
    public Optional<Student> findByStudentId(@PathVariable Long id) {
        return studentService.findStudentById(id);
    }

    @GetMapping("/student/name/{name}")            // name으로 조회
    public List<Student> findByStudentName(@PathVariable String name) {
        return studentService.findStudentByName(name);
    }
```

controller

StudentController.java

```
    @PutMapping("/update/{id}/{address}")              // address 수정
    public String updateStudentAddress(@PathVariable Long id, @PathVariable
      String address) {
        studentService.updateStudentAddress(id, address);
        return "updated";
    }

    @DeleteMapping("/delete/{id}")          // 데이터 삭제
    public String deleteStudentById(@PathVariable Long id) {
        studentService.deleteStudentById(id);
        return "deleted";
    }

    @PostMapping("/student/add")          // 데이터 삽입
    public List<Student> addStudents(@RequestBody List<Student> students) {
        return studentService.addStudents(students);
    }

    @GetMapping("/student/all")              // 전체 조회
    public List<Student> getAllStudents() {
        return studentService.getAllStudents();
    }
}
```

service

DeptService.java

```
package dbp.UnivQuerydsl.service;

import dbp.UnivQuerydsl.entity.Dept;
import dbp.UnivQuerydsl.repository.DeptRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class DeptService {

    @Autowired
    DeptRepository deptRepository;

    public Optional<Dept> findDeptByName(String name) {
        return deptRepository.findDeptByName(name);
    }

    public List<Dept> findDeptByBuilding(String building) {
        return deptRepository.findDeptByBuilding(building);
    }

}
```

service

StudentService.java

```
package dbp.UnivQuerydsl.service;

import dbp.UnivJpql.entity.Student;
import dbp.UnivJpql.repository.StudentRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class StudentService {

    @Autowired
    StudentRepository studentRepository;

    public Optional<Student> findStudentById(Long id) {
        return studentRepository.findStudentById(id);
    }

    public List<Student> findStudentByName(String name) {
        return studentRepository.findStudentByName(name);
    }
```

service

StudentService.java

```
    public void updateStudentAddress(Long id, String address) {
       studentRepository.updateStudentAddress(id, address);
    }

    public void deleteStudentById(Long id) {
        studentRepository.deleteStudentById(id);
    }

    public List<Student> addStudents(List<Student> students){
        return studentRepository.saveAll(students);
    }

    public List<Student> getAllStudents(){
        return studentRepository.getAllStudents();
    }

}
```

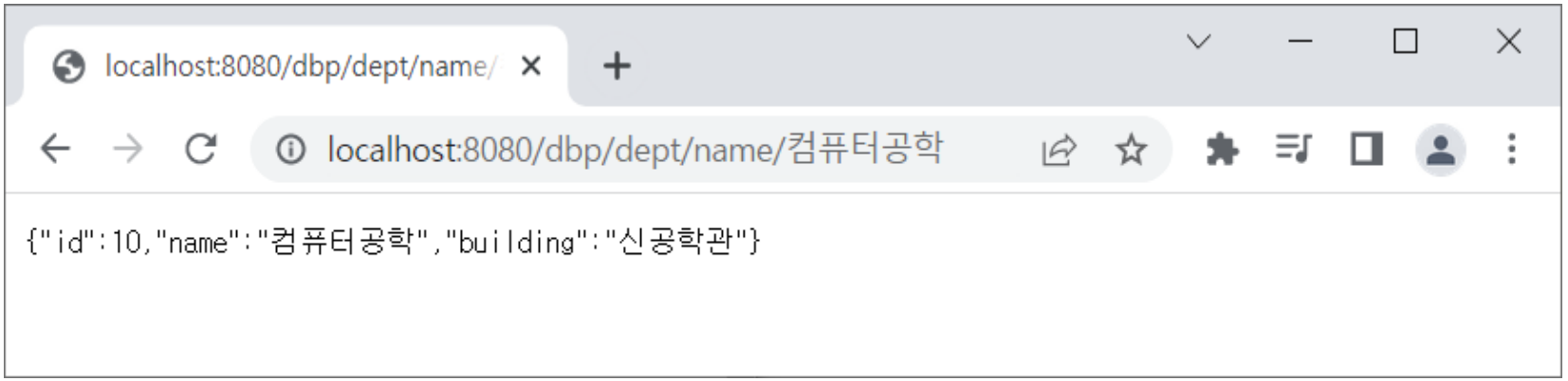
config

UnivQuerydslConfiguration.java

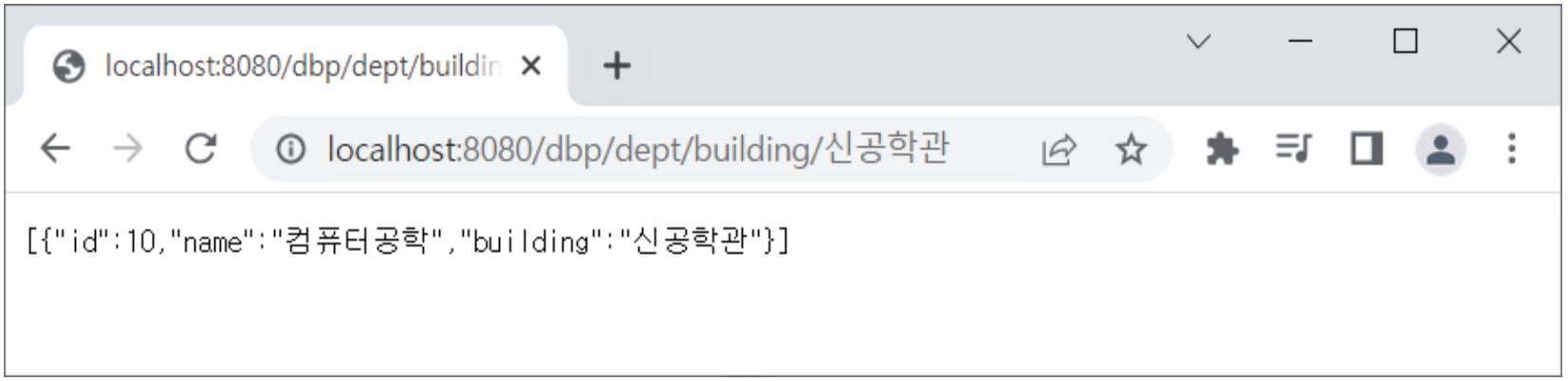
```
package dbp.UnivQuerydsl.config;

import com.querydsl.jpa.impl.JPAQueryFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Configuration
public class UnivQuerydslConfiguration {

    @PersistenceContext
    private EntityManager entityManager;

    @Bean
    public JPAQueryFactory jpaQueryFactory() {
        return new JPAQueryFactory(entityManager);
    }

}
```

실행 결과

- 부서명(name) 조회 - @GetMapping("/dbp/dept/name/{name}")

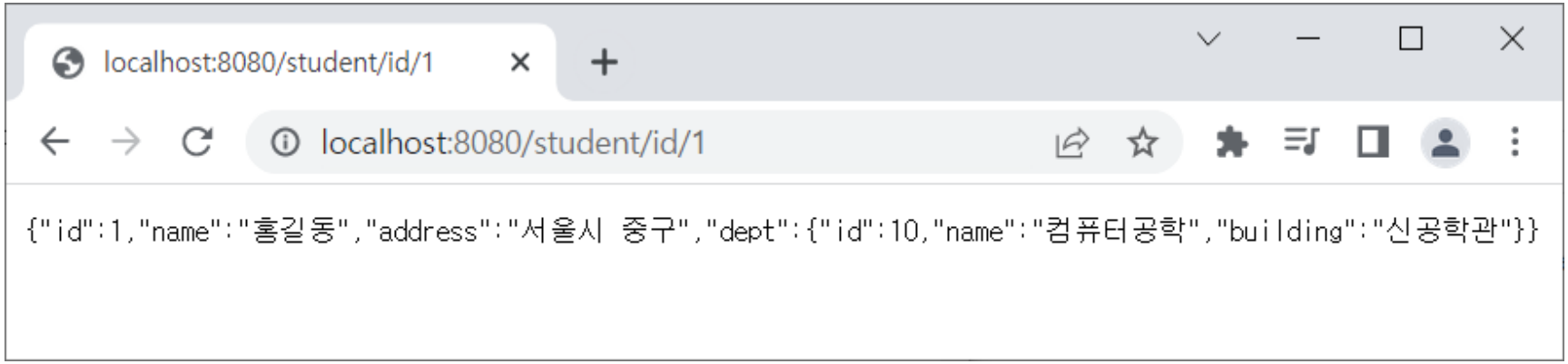


- 부서위치(building) 조회 - @GetMapping("/dbp/dept/building/{building}")

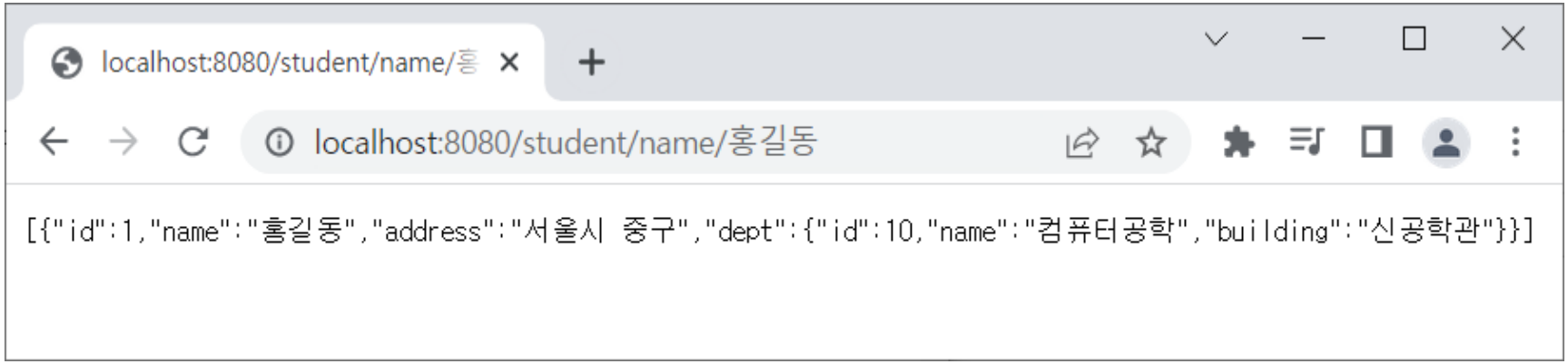


실행 결과

- 학번(id) 조회 - @GetMapping(“/student/id/{id}”)

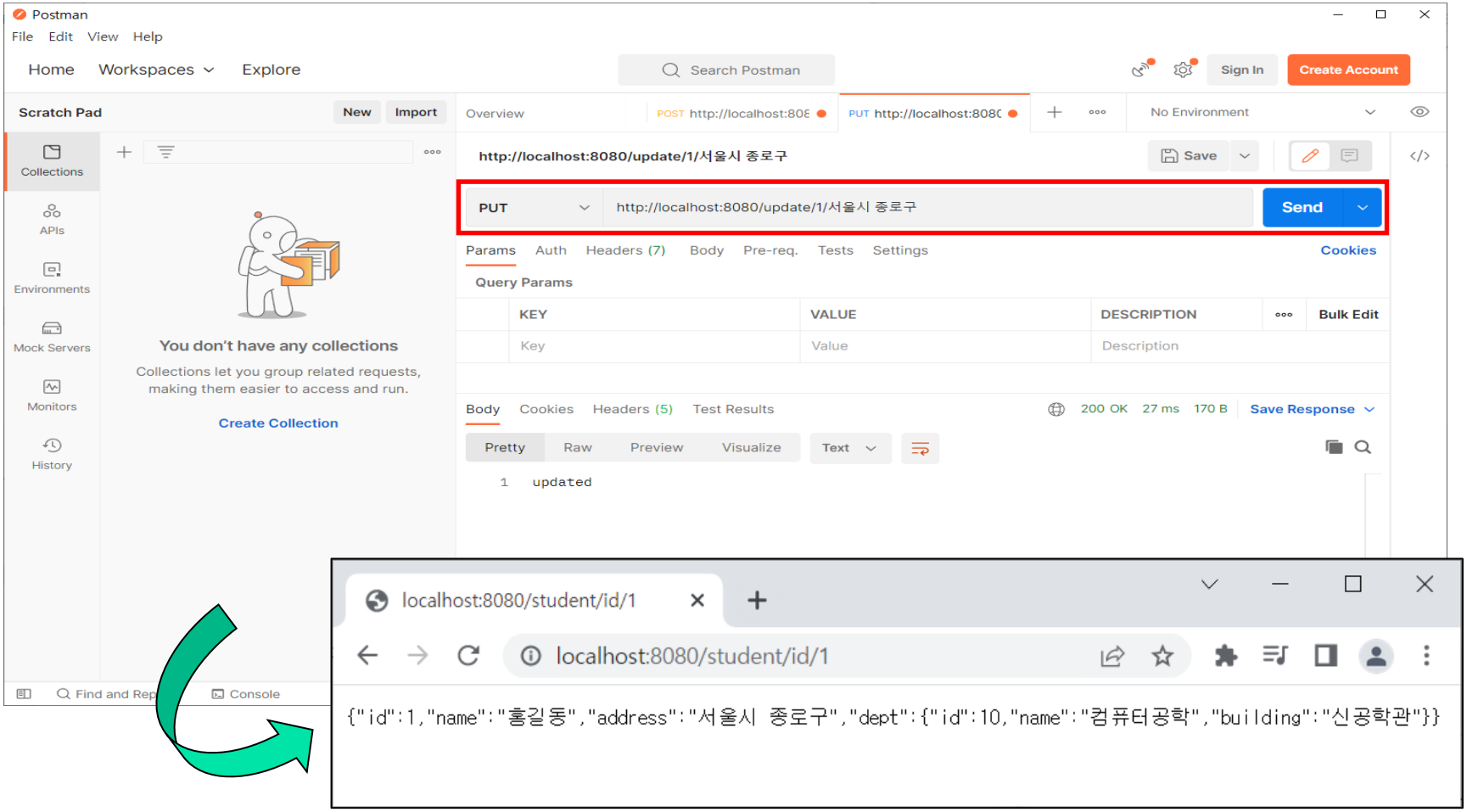


- 학생이름(name) 조회 - @GetMapping("/student/name/{name}")



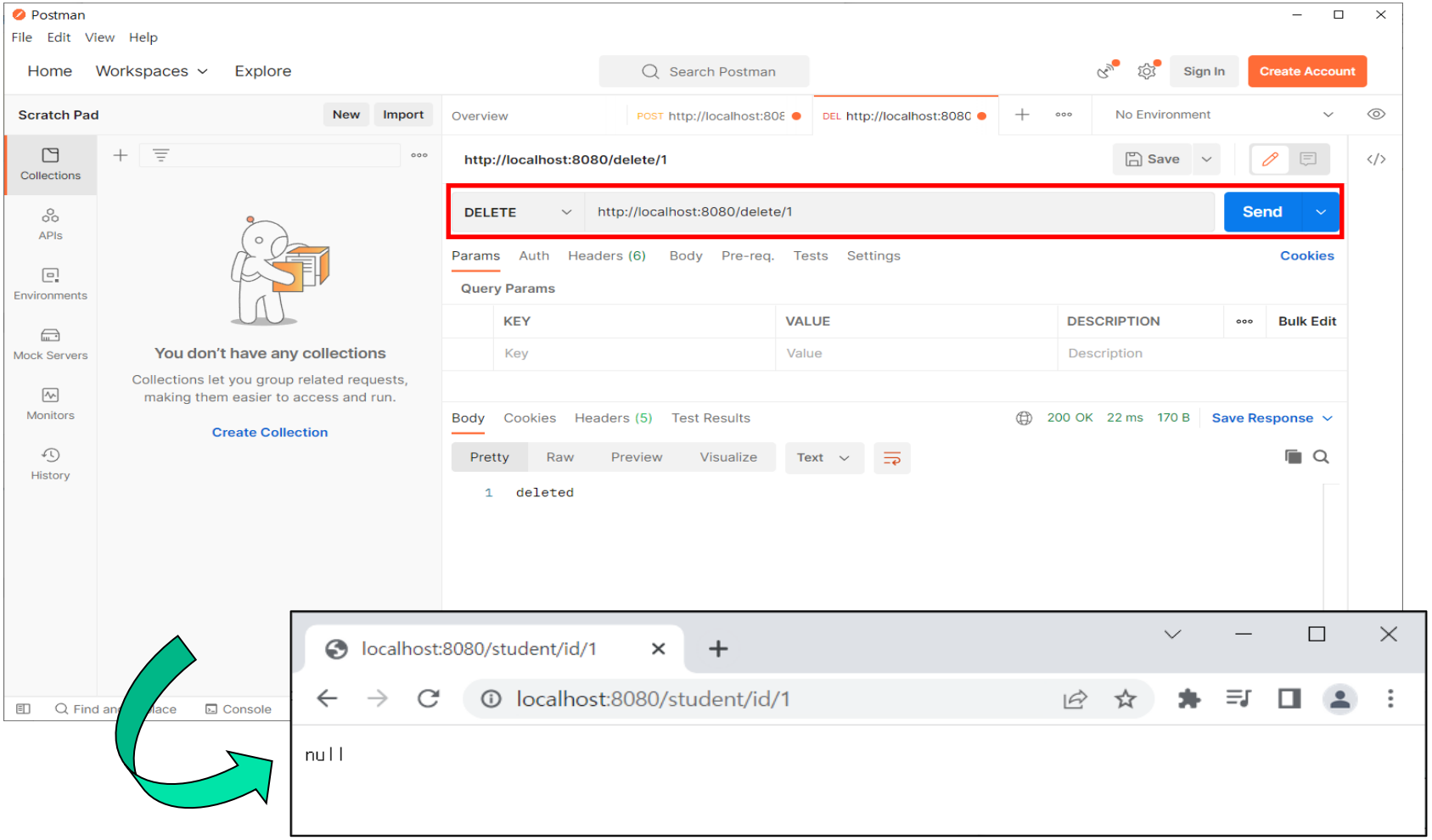
실행 결과

- 수정 - @PutMapping("/update/{id}/{address}")



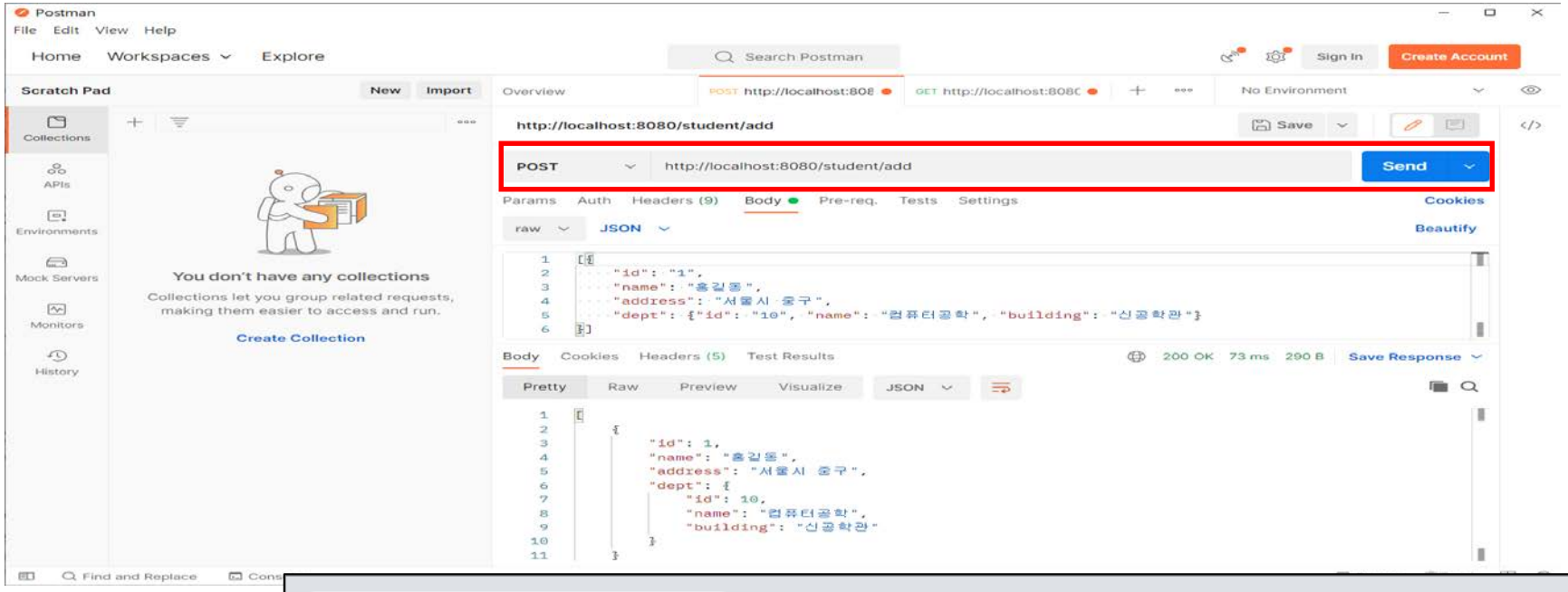
실행 결과

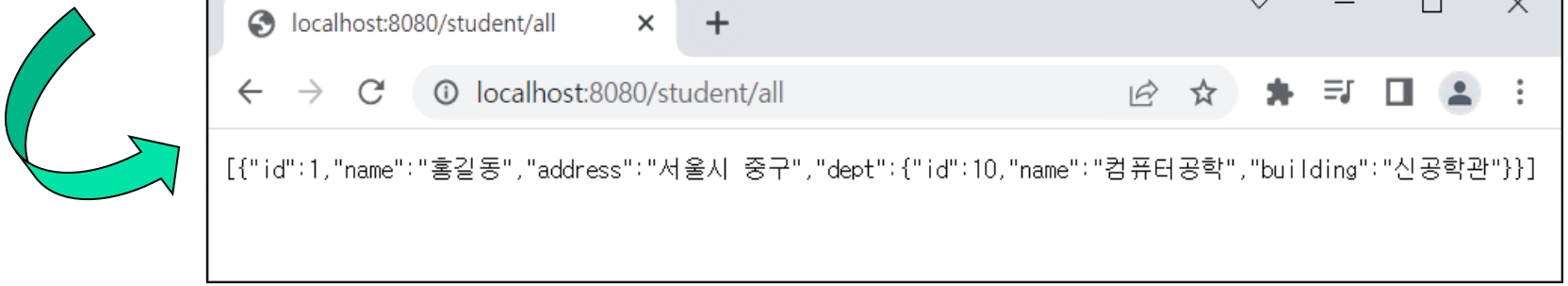
- 삭제 - @DeleteMapping("/delete/{id}")



실행 결과

- 삽입 - @PostMapping("/student/add")





실행 결과

- 전체 조회 - @GetMapping("/student/all")