

JSP & Servlet

1. JSP & Servlet 개요

2. 서블릿

3. HTML 태그와 서블릿에서 폼 데이터 처리

4. 서블릿 생명주기 & 서블릿 Context

5. JSP

6. 세션과 쿠키/예외처리

7. JDBC와 DAO패턴

8. 스크립팅 요소 제거(EL, JSTL)

Chapter

01 JSP & Servlet 개요

1. 웹 애플리케이션 개요
2. 서블릿이란?
3. JSP란?
4. MVC와 Model아키텍처
5. 개발환경설정

1. 웹 애플리케이션

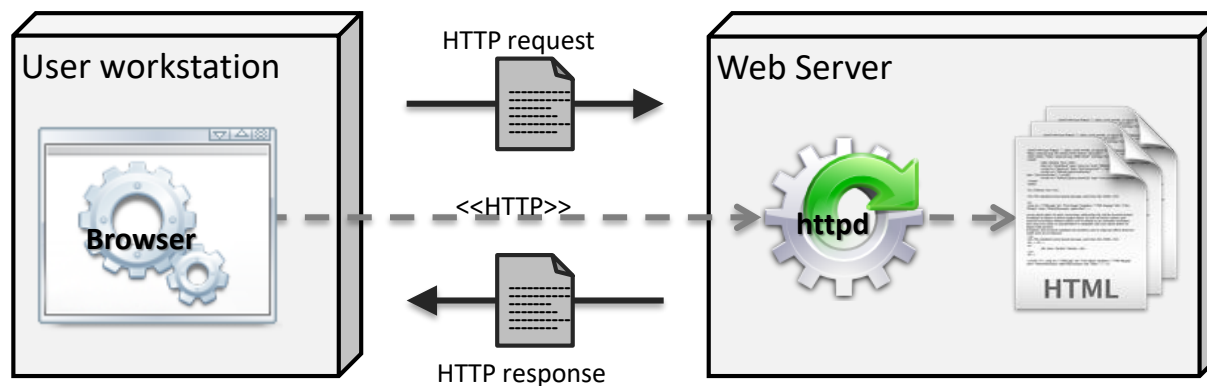
1.1 웹 애플리케이션 개요

브라우저가 서버의 페이지를 요청하면 서버는 해당 파일을 찾은 다음 HTTP 응답을 통해 클라이언트에 전송
브라우저는 응답된 페이지를 해석하여 화면에 보여준다.

request 정보는 사용자가 원하는 파일 또는 리소스의 위치와 브라우저에 관한 정보를 포함한다.

response 정보는 요청한 자원에 관한 정보를 가지고 있다.(일반적으로 텍스트, 그러나 그래픽 등의 바이너리 정보도 가능)

지금 이 그림은 정적페이지 형태를 나타낸 그림



웹 애플리케이션 형태

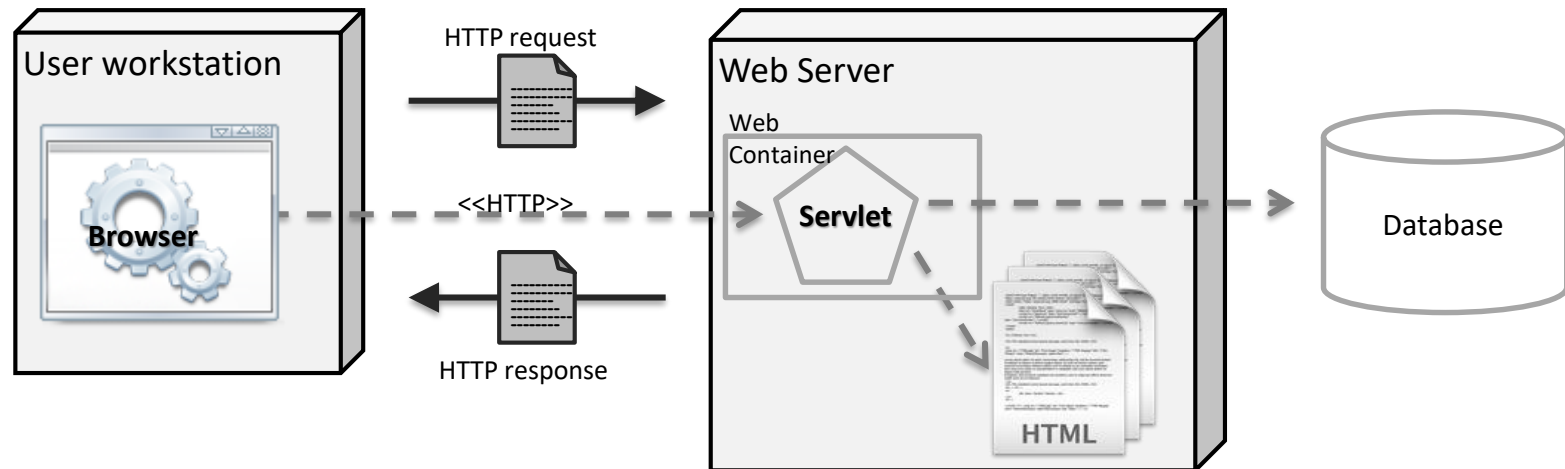
1.1 웹 애플리케이션 개요

JSP와 서블릿을 이용하면 아래와 같은 동적(dynamic)인 웹 애플리케이션을 작성 가능

동적 작동을 위해 웹 컨테이너가 추가됨 – 사용자의 요청을 처리(단, 모든 요청을 처리하지는 않음/ JSP & Servlet만)

데이터베이스의 데이터도 사용 가능 – 항상 같은 결과를 보여주는 정적이 아닌 동적 애플리케이션 작동

실행 결과는 똑같이 HTML문서 형태로 작성되어 response로 출력



2. 서블릿이란?

1.2 서블릿 개요

자바를 기반으로 하는 웹 애플리케이션 프로그래밍 기술 – 자바 클래스 형태로 웹 애플리케이션을 작성(서블릿 클래스)
웹 서버 안의 웹 컨테이너에서 실행 – 웹 컨테이너에서 서블릿 인스턴스 생성 후 사용자 요청 처리

- 서블릿 클래스만 사용해 홈페이지 개발 시 과정

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Sum of 1 to 100</TITLE></HEAD>");
    out.println("<BODY>");
    int total = 0;
    for (int cnt=1; cnt<=100; cnt++)
        total += cnt;
    out.println("1+2+3+...+100="+total);
    out.println("</BODY>");
    out.println("</HTML>");|
```

```
관리자: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

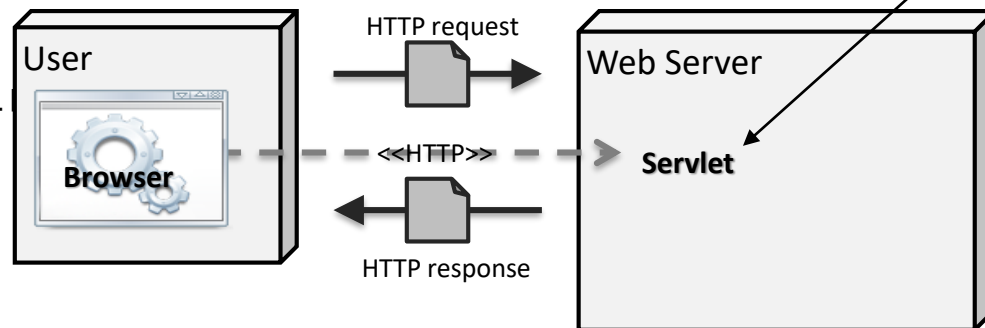
C:\Users\COM>d:

D:\>cd Projects\weclipse-workspace\WebExample\src\lab\web\servlet
D:\Projects\weclipse-workspace\WebExample\src\lab\web\servlet>javac ExampleServlet.java
```

1. 서블릿 클래스 소스코드 작성

2. 소스코드 컴파일

3. 컴파일 후 웹서버 디렉토리에 저장하 URL에 해당하는 서블릿 클래스 호출



1.2 서블릿 개요

장점

- 최초의 웹 애플리케이션 작성 기술인 CGI의 단점을 개선하기 위해 개발 – 많은 사용자의 작업 요청 시 처리 속도가 월등히 빠름
- 자바의 장점인 플랫폼 독립성을 물려받음

단점

- HTML 코드가 자바 코드 안으로 들어가는 구조로 인해 HTML 문서 자체를 이해하기 어렵게 만듦
- 표현 계층과 비즈니스 계층 분리가 불가능(화면을 담당하는 표현 계층과 화면에 나타낼 내용을 처리하는 비즈니스 계층 분리 불가)
 - ex) 웹 디자이너가 서블릿 클래스로 만든 홈페이지를 수정하려면 소스코드를 이해하지 않고는 불가능
- 수정이나 기능 추가 시 지속적인 컴파일 필요

3. JSP란?

1.3 JSP 개요

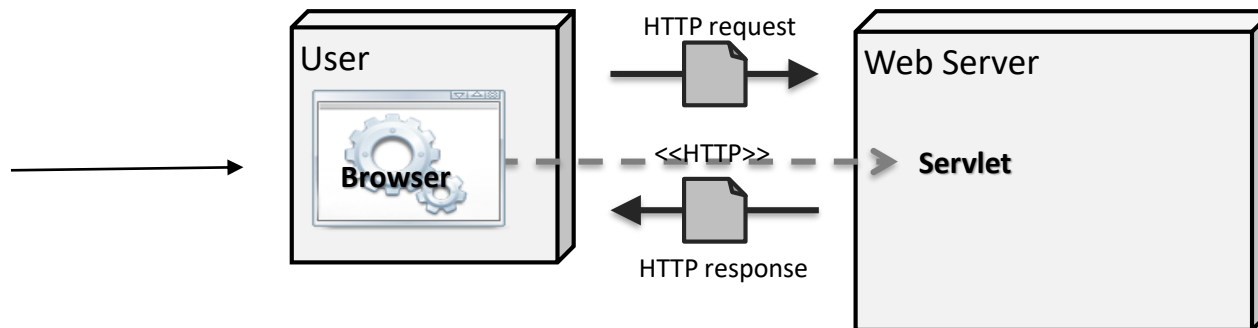
서블릿과 마찬가지로 자바를 기반으로 한 웹 애플리케이션 프로그래밍 기술 – HTML 페이지에 자바 코드를 사용 가능하게 함

JSP는 서버에서 실행되는 스크립트 형식의 파일 – 컴파일 과정 없이 서버에서 바로 실행 가능

HTML 태그 사이에 자바 언어가 삽입되는 구조 - <% 와 %> 사이에 자바 코드, 변수 값 출력은 <%= 변수명 %>

- JSP 기술을 이용해 홈페이지 개발 시 과정

```
Exam.jsp
1 <%@ page contentType="text/html; charset=UTF-8"%>
2 <!DOCTYPE html>
3 <html>
4 <head>
5 <meta charset="UTF-8">
6 <title>1부터 100까지의 합</title>
7 </head>
8 <body>
9 <% int total=0;
10 for(int cnt=1; cnt<=100; cnt++)
11 total+=cnt;%>
12 1부터 100까지의 합은? <%=total %>
13 </body>
14 </html>
```



1. JSP 형식으로 홈페이지 코드 작성

2. 컴파일 없이 바로 웹 서버의 디렉토리에 저장 후 호출

장점

- 컴파일 과정을 거치지 않기 때문에 빠른 작성 및 수정이 가능함
- 일반 자바프로그램과 똑같이 취급하기 때문에 서블릿이 가지고 있는 장점을 모두 계승함

단점

- 서블릿과 마찬가지로 비즈니스 로직과 표현 로직을 분리하기 어려움
- 웹 애플리케이션의 로직이 복잡해지면 HTML 중심의 코드가 오히려 소스코드를 이해하기 어렵게 만들 수 있음
- 소스코드를 쉽게 변경 가능하기 때문에 소스코드의 안전성이 보장되지 않음
ex) 웹 디자이너가 소스코드를 망가뜨릴 수 있음

JSP는 표현 로직, 서블릿은 비즈니스 로직을 담당
하게
코드를 작성한다면 모든 단점을 해결 가능

4. MVC와 Model 아키텍처

1.4 MVC 구조

웹 애플리케이션에는 서블릿/JSP와 함께 일반 자바 클래스도 같이 사용함

- 가장 이상적인 구조

자바 클래스 – 데이터베이스와 통신/주요 비즈니스 로직 구현

서블릿 – 사용자의 요청 처리/제어 컨트롤러 구현

JSP – 화면에 보이는 페이지 작성

이와 같은 구조를 MVC, Model View Controller 라고 명칭함

MVC와 Model 아키텍처

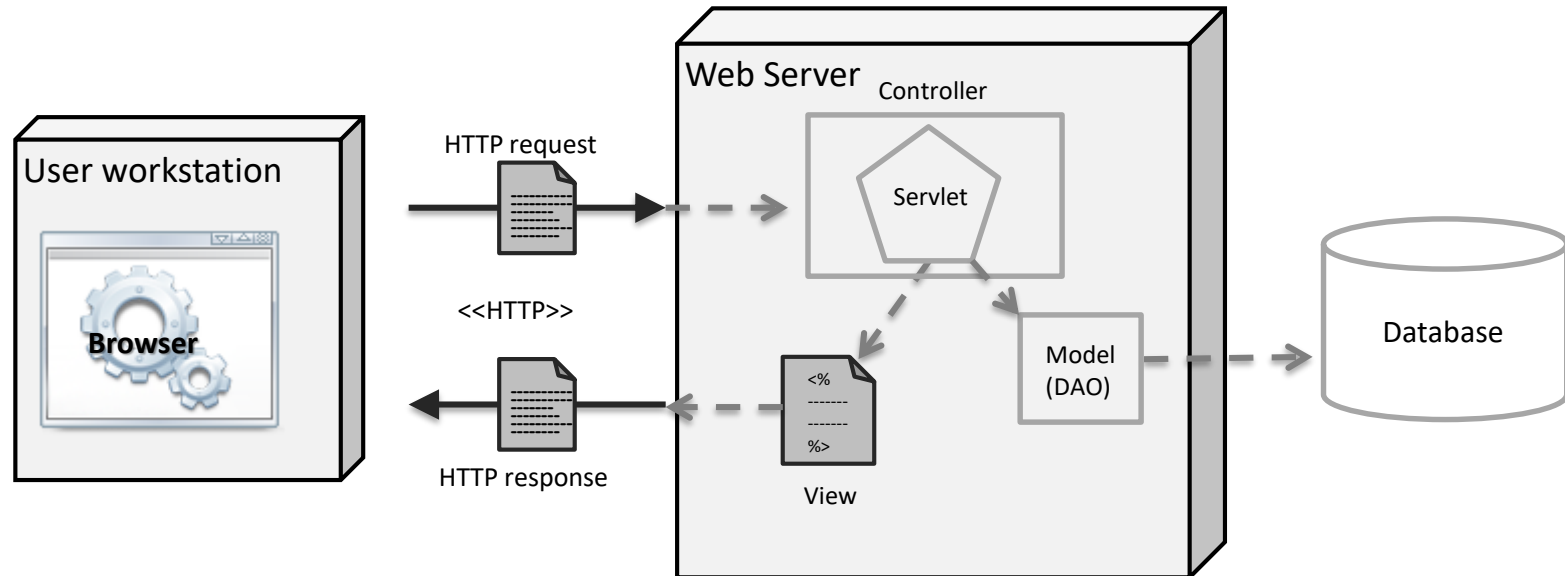
1.4 MVC 구조

자바 클래스 – 비즈니스 로직, 주로 데이터베이스와 통신하는 DAO(Data Access Object)가 존재

서블릿 – 컨트롤러, 사용자로 부터 요청된 데이터를 검증하고 모델을 호출한 후 뷰에 데이터를 보내 응답하도록 함

JSP – 뷰, 데이터를 받아 화면에 보이게 코드를 작성하거나 사용자가 입력할 수 있도록 HTML 폼(form)을 작성

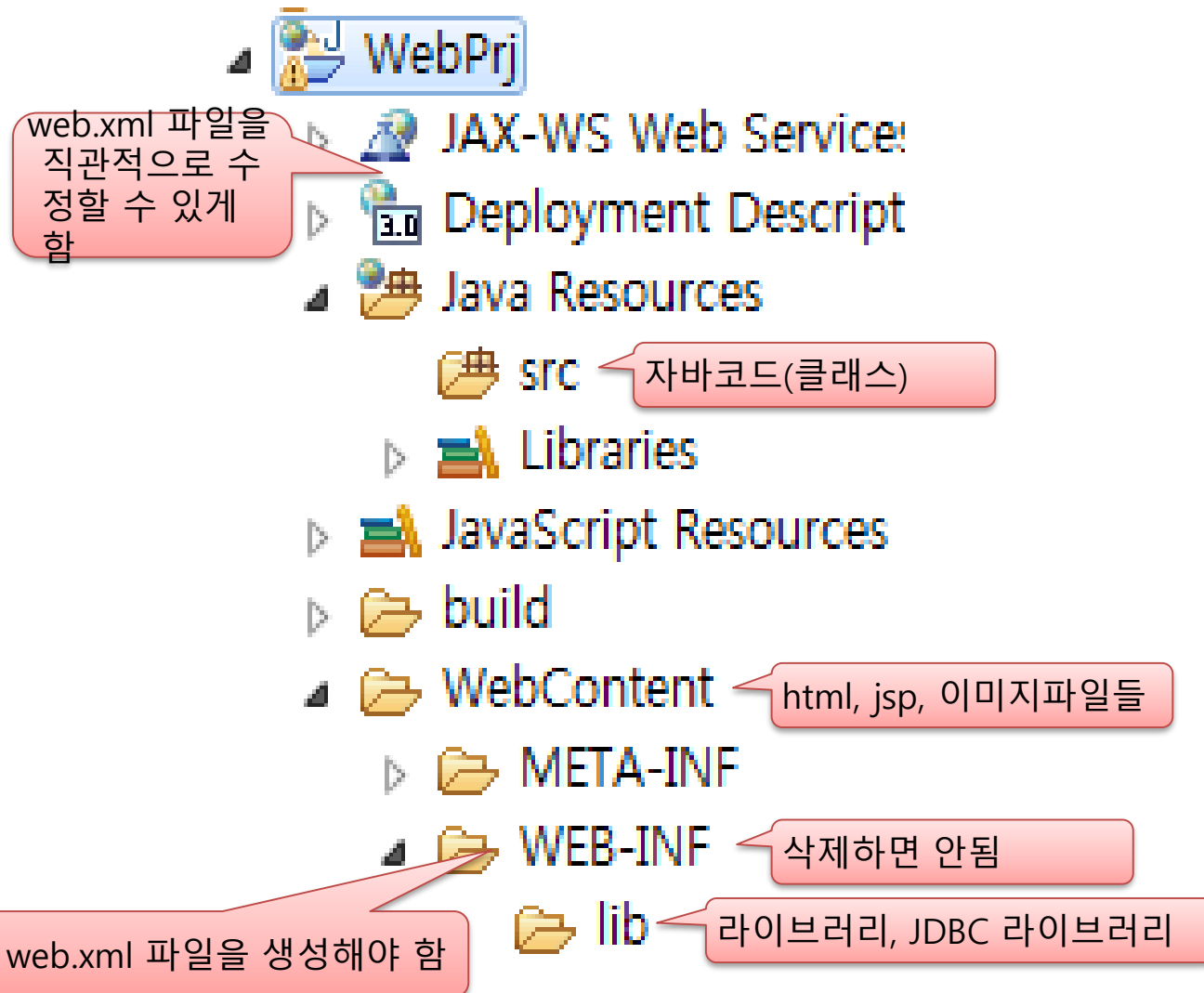
항상 MVC를 분리하지는 않음 – Model 1은 컨트롤러와 뷰가 혼재



1.4 MVC 장점

MVC 구조의 장점

- 프로그래머가 비즈니스 로직을 구현하기 편함
- 확장성, 유지보수성이 뛰어남
- JSP와 서블릿의 단점을 한꺼번에 해결
- 많은 프레임워크들을 통해 쉽고 빠르게 배우거나 개발 가능
(Struts 프레임워크, Spring MVC 프레임워크, JavaServer Faces 등)



Chapter

02 서블릿

1. 서블릿 클래스 작성

1. 서블릿 클래스 작성

2.1 이클립스를 사용하지 않는 서블릿 클래스 작성

작성할 때 지켜야 할 규칙

- 서블릿 클래스는 javax.servlet.http.HttpServlet 클래스를 상속하도록 만들어야 함
 - 따라서 import javax.servlet.http.*; , import javax.servlet.*; , import java.io.*; 등의 코드가 들어가야함
- doGet 또는 doPost 메서드 안에 웹 브라우저로부터 요청이 왔을 때 해야 할 일을 기술해야 함
 - 서블릿의 디폴트 실행 메서드는 doGet, 보통 doGet은 데이터를 조회하는 메서드
doPost는 데이터 저장/처리를 하는 메서드를 입력함
- doPost는 데이터 처리가 필요한 메서드를 입력함
- HTML 문서는 doGet, doPost의 두 번째 파라미터인 HttpServletResponse를 통해 출력해야 함

2.1 이클립스를 사용하지 않는 서블릿 클래스 작성

HundredServlet.java 작성 (D:\Projects 안에 저장)

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class HundredServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        int total = 0;
        for (int cnt = 1; cnt < 101; cnt++)
            total += cnt;

        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hundred Servlet</TITLE></HEAD>");
        out.println("<BODY>");
        out.printf("1+2+3+...+100 = %d", total);
        out.println("</BODY>");
        out.println("</HTML>");
    }
}
```

Response 객체를 만들어 `getWriter()` 메서드를 불러옵니다. `getWriter()` 메서드의 `println`은 html 코드를 웹페이지에 표기하는 역할을 합니다.

2.1 이클립스를 사용하지 않는 서블릿 클래스 작성

web.xml 작성

web.xml – web application deployment descriptor 로써 설정 파일. 서블릿이 실행되면서 가장 먼저 읽어 들임.

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="4.0">
  <servlet>
    <servlet-name>hundred-servlet</servlet-name>
    <servlet-class>HundredServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hundred-servlet</servlet-name>
    <url-pattern>/hundred</url-pattern>
  </servlet-mapping>
</web-app>
```

문법의 식별자 & 문법 버전

Mapping 엘리먼트가 name의 이름 값으로 servlet 엘리먼트를 참조함

서블릿클래스의 이름

서블릿의 URL 주소

web.xml의 위치는 WEB-INF 폴더 안 classes와 같이 존재하면 됩니다.

Chapter

03 HTML 태그와 서블릿에서 폼 데이터 처리

1. HTML 태그
2. 태그 예제
3. 서블릿에서 폼 데이터 접근

1. HTML 태그

3.1 HTML 태그

Form 태그


<FORM ACTION = '/BBS'>

<FORM ACTION = '/BBS' METHOD='GET'>

<FORM ACTION = '/BBS' METHOD='POST'>

ACTION : 사용자가 내용 입력 후 전송할 때 데이터가 전달 될 장소를 지정하는 부분(URL)

METHOD : 데이터가 반환되는 유형을 결정하는 것. 아무것도 입력하지 않으면 GET을 실행하지만 주로 POST가 많이 이용됨

	GET 방식	POST 방식
용도	서버에 데이터를 요청하는 용도	서버에 데이터를 전송하는 용도
데이터	데이터가 주소에 묻어서 감	전송되는 데이터가 전송객체의 body를 통해 전달 됨
보안성	 localhost:8080/BBS?name=엄영범&id=gctserf&password=0000 전송했던 데이터는 브라우저의 주소바의 입력 속했던 주소와 함께 남아있음	브라우저에는 전달되는 데이터가 남지 않음
주요 용도	게시판 글 조회 등 서버의 정보를 가져오기 위해 사용	비밀번호, 주민번호 등 private 한 데이터를 서버에 전 송할 때 사용
대량 데이터	전송할 수 있는 최대 크기는 브라우저 별로 다르 지만 크기가 정해져 있음	게시판 등 대량의 데이터 전송 시 사용 가능
폼 양식	반드시 폼 양식이 있어야 함	폼 양식이 없어도 됨

3.1 HTML 태그

Input 태그와 type 속성의 값

Input 엘리먼트는 한 줄로 된 텍스트 입력상자인 에디트 필드를 만들어 주며, type 값에 따라 여러 형식의 데이터를 입력할 수 있도록 해줌

text & password

```
<input type="text" name="이름" size="크기" maxlength="글자 수">
```

```
<input type="password" name="이름" size="크기" maxlength="글자 수">
```

text는 글자를 그대로, password는 ***로 표시해서 보여줌

은 html 태그에서 띄어쓰기를 의미

<p> 는 문단을 만들어 줌

3.1 HTML 태그

Input 태그와 type 속성의 값

Checkbox : 여러 가지 선택 항목을 나열하고 그 중에서 원하는 항목을 체크할 수 있는 형식. 동시에 여러 개 선택 가능

`<input type="checkbox" name="이름" value="제어값" checked>` 선택항목

Radio : 여러 개의 항목 중 하나의 항목만 선택하도록 함

`<input type="radio" name="이름" value="제어값">` 선택항목

submit : 입력 창에 등록된 내용들을 서버로 전송 시킬 때 사용하는 버튼. Value는 버튼에 나타나는 글자를 뜻함

`<input type="submit" value="입 력">`

Reset : 입력 창에 등록된 내용들을 초기화

`<input type="reset" value="취 소">`

Upload : type 속성 값을 file로 하면 파일을 선택할 수 있는 폼이 나타남

`<input type="file" name="pic" accept="image/gif">`

3.1 HTML 태그

Input 태그와 type 속성의 값(HTML5 추가 버전)

Email : 이메일 주소를 입력하게 해 줌. 텍스트와 다른 점은 이메일 주소의 형식이 유효한지를 자동으로 체크
<input type="email" name="이름">

url : 이메일과 마찬가지로 자동으로 url 형식에 맞는지 체크
<input type="url" name="이름">

Number : 숫자만 표시해야 하는 입력필드에 사용. 입력되는 숫자의 범위 설정 가능. Step 속성은 숫자 간격을 의미
<input type="number" name="points" min="1" max="10" step="1">

Range : 슬라이드 바 모양의 범위를 선택할 수 있도록 함. Number 속성과 비슷함
<input type="range" name="points" min="1" max="10" step="1">

Search : 검색 필드를 만들 때 사용. 보통 text필드와 동일하게 동작. Placeholder 속성은 필드 값 안에 들어갈 텍스트
<input type="search" placeholder="Search...">

3.1 HTML 태그

Input 태그와 type 속성의 값(HTML5 추가 버전)

Color : 색상 값을 입력할 수 있도록 해 줌. 크롬 20버전 이상 지원, 익스플로러는 지원하지 않음

```
<input type="color" name="이름">
```

Tel : 전화번호를 입력할 수 있도록 해 줌. Pattern 속성은 정규표현식을 사용 가능하게 해 줌.

```
<input type="tel" pattern="^\d{3}\d{3}\d{3,4}\d{4}$">
```

위 패턴은 000-000-0000 또는 000-0000-0000 형식의 입력 값을 받음.

Textarea : 여러 줄 내용을 입력 받는 태그

```
<input type="textarea" name="이름" cols="글자 수" rows="줄 수">...</textarea>
```

Select : 선택 리스트(drop-down list)를 만들 때 사용. Option 태그를 사용해 선택 옵션 정의

```
<select name="변수명">
```

```
    <option value="값1">항목1
```

```
    <option value="값2">항목2
```

```
...
```

```
</select>
```

2. 태그 예제

3.2 태그 예제

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>폼 테스트</title>
</head>
<body>
<h1>입력값 테스트</h1>
<form action = "/BBS" method="post">
아이디 : <input type="text" name="id" size="10"
maxlength="10" value="user01"><br>
비밀번호 : <input type="password" name="pw" size="10"><br>
좋아하는 과일(여러개 선택 가능):
<input type="checkbox" name="fruit" value="10">사과
<input type="checkbox" name="fruit" value="20" checked>귤
<input type="checkbox" name="fruit" value="30">감<br>
성별:
<input type="radio" name="sex" value="M" checked>남자
<input type="radio" name="sex" value="F" checked>여자<br>
```

순서대로



직업:

```
<select name="job" size="2">
<option value="학생">학생
<option value="회사원" selected>회사원
<option value="변호사">변호사
<option value="의사">의사
<option value="감사">감사
</select><br>
<textarea name="data" cols="50" rows="5">기본값은 여기에 넣습니
다.
공백도 나타냅니다.</textarea><br>
<input type="file" name="file"><br>
<input type="submit" value="저 장">
<input type="reset" value="취 소">
</form>
</body>
</html>
```

3.2 태그 예제

BBSPostServlet의 doPost() 메서드에 작성

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    request.setCharacterEncoding("utf-8");
    String id = request.getParameter("id");
    String pw = request.getParameter("pw");
    String [] fruit = request.getParameterValues("fruit");
    String sex = request.getParameter("sex");
    String job = request.getParameter("job");
    String data = request.getParameter("data");
    response.setContentType("text/html; charset=utf-8");
    PrintWriter out = response.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>태그결과출력화면</TITLE></HEAD>");
    out.println("<BODY>");
    out.printf("아이디: %s <BR>", id);
    out.printf("비밀번호: %s <BR>", pw);
    out.print("과일: ");
    for(String val : fruit) {
        out.printf("%s ", val);
    }
    out.print("<BR>");
    out.printf("성별: %s <BR>", sex);
    out.printf("직업: %s <BR>", job);
    out.println("-----<BR>");
    out.printf("<PRE>%s</PRE>", data);
    out.println("-----<BR>");
    out.println("저장되었습니다.");
    out.println("</BODY>");
    out.println("</HTML>");
}
```


3.2 태그 예제

정상 실행 화면



입력값 테스트

아이디 :

비밀번호 :

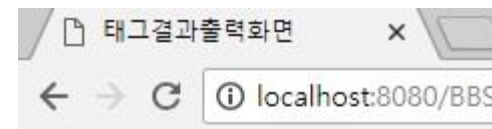
좋아하는 과일(여러개 선택 가능): ☐ 사과 ☒ 귤 ☐ 감

성별: ☐ 남자 ☒ 여자

직업: (dropdown menu showing '학생' and '회사원')

기본값은 여기에 넣습니다.
공백도 나타냅니다.

선택된 파일 없음



아이디: user01
비밀번호: 1234
과일: a b
성별: F
직업: 회사원

기본값은 여기에 넣습니다.
공백도 나타냅니다.

저장되었습니다.

3. 서블릿에서 폼 데이터 접근

3.3 서블릿에서 폼 데이터 접근

웹 애플리케이션은 사용자가 데이터를 받기만 하지 않음 - 때에 따라 전송도 해야함

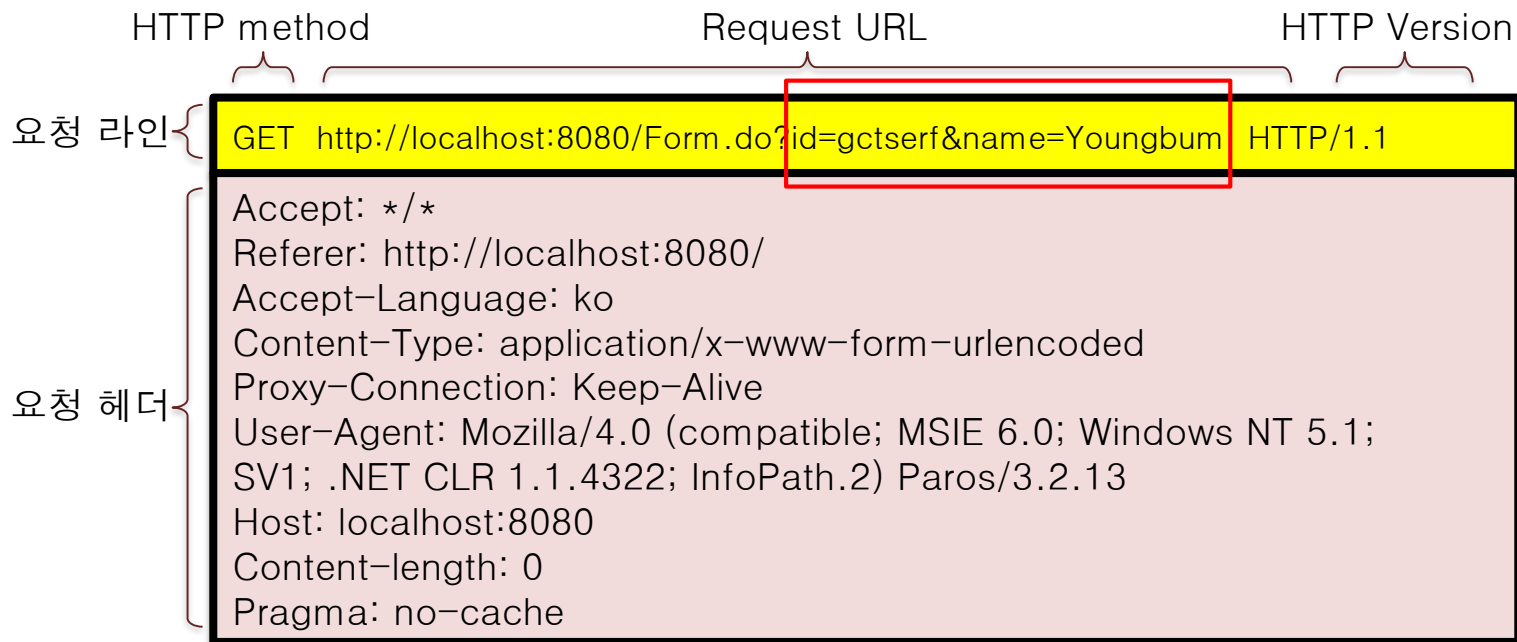
이 때 입력하는 데이터를 **폼 파라미터**라고 부르며 요청 방식에 따라 파라미터 전송 방식이 다름

GET 방식은 메시지의 URL에 포함되어 데이터가 전송됨

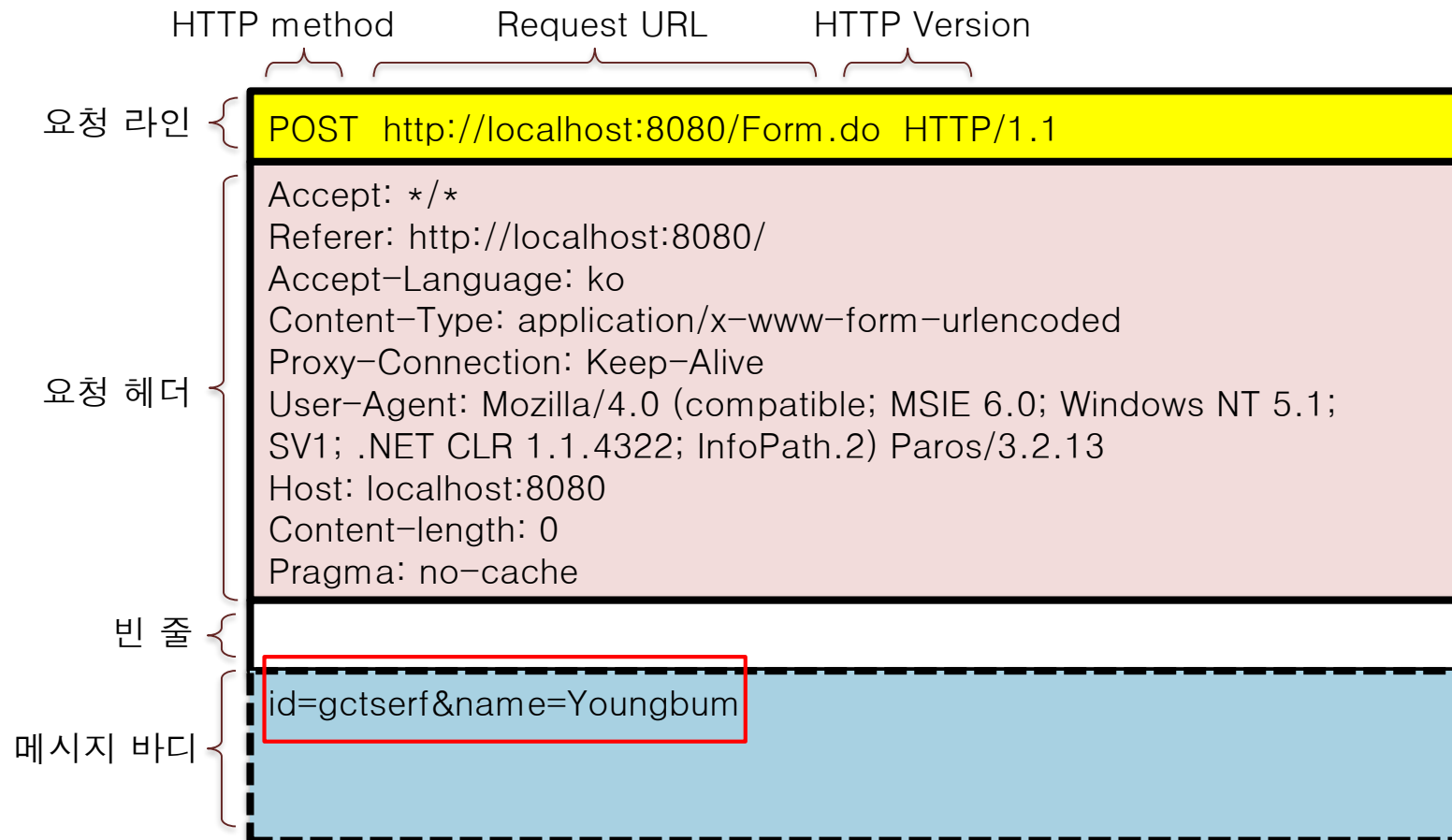
POST 방식은 URL이 아닌 요청 메시지 바디에 포함되어 전송됨

3.3 서블릿에서 폼 데이터 접근

GET 방식 요청



POST 방식 요청



3.3 서블릿에서 폼 데이터 접근

GET 방식을 사용하는 경우

- 동일한 요청에 대해 동일한 응답 결과를 가져와야 하는 경우
- 데이터의 양이 적을 경우
- 즐겨찾기에 등록되게 하고 싶은 경우

POST 방식을 사용하는 경우

- 요청의 처리가 데이터베이스에 데이터를 저장하는 것 처럼 서버를 변경할 경우
- 데이터의 양이 클 경우
- 비밀번호 필드 등 데이터 내용이 URL을 통해 보이면 안 되는 경우

3.3 서블릿에서 폼 데이터 접근

doGet과 doPost는 HttpServletRequest 객체와 HttpServletResponse 객체를 파라미터 변수로 받음
따라서 이 둘의 인터페이스에 정의되어 있는 API를 사용할 수 있음

API

String getParameter(String name) : name에 해당하는 파라미터의 값을 반환합니다. Name 값은 폼의 요소들의 name 속성의 값과 일치해야 함

Enumeration getParameterNames() : 파라미터의 이름 목록을 반환함

String[] getParameterValues(String name) : name에 해당하는 파라미터들의 값을 반환. 같은 이름의 파라미터가 여러 개의 값을 가지고 있을 경우 사용. Ex)폼의 체크박스, 리스트

Map(String, String[]) getParameterMap() : 요청 파라미터 정보를 Map 형식으로 반환함

Void setCharacterEncoding(String charset) : 요청 데이터의 인코딩 처리

3.3 서블릿에서 폼 데이터 접근

- 서블릿에서 파라미터 값을 처리할 때는 모든 파라미터 값이 문자열로 전달 됨
ex) 정수 사용을 원하면 `Integer.parseInt(String data)` 로 형변환해야 함
- 요청 파라미터의 파라미터 이름을 모를 경우 `getParameterMap()` 메서드를 통해 모든 요청 파라미터 조회 가능
- GET 방식 요청 시 파라미터에 한글을 포함하고 싶은 경우 `request.setCharacterEncoding("utf-8");` 로 요청객체의 인코딩만 설정해서는 불가능 함 – 톰캣 conf 폴더 안 `server.xml`에서 `<Connector>` 태그에 `URIEncoding="utf-8"` 속성을 추가해야 함

Chapter

04 서블릿 생명주기 & 서블릿 Context

1. 서블릿 생명주기
2. 서블릿 Context

1. 서블릿 생명주기

4.1 서블릿 생명주기

- 서블릿 생명주기 - 서블릿 클래스가 로드된 다음 `init()` → `service()` → `destroy()` 순으로 메서드가 호출 됨

`init()` : 서블릿 인스턴스가 만들어진 후 웹 컨테이너에 의해 호출됨. 서블릿 생명주기 동안 한 번만 호출 되고, `doGet()` 메서드나 `doPost()` 메서드를 실행하기 전에 해야 할 내용을 정의함. 주로 자원 획득과 관련된 코드가 포함 됨. `ServletConfig` 객체를 생성자로 받으며 이를 통해 `web.xml` 문서의 `<servlet>` 태그 안 `<init-param>` 값 획득 가능.

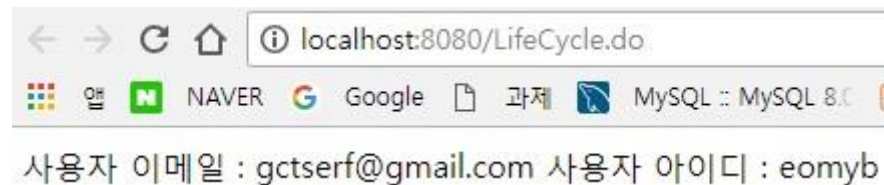
`service()` : 사용자의 요청이 있을 때 웹 컨테이너에 의해 호출됨. 요청된 URL을 URL mapping 정보랑 매핑해 해당 서블릿을 호출함. 사용자의 요청 방식에 따라 `do****()` 메서드를 호출하며 `request`와 `response` 객체를 생성자로 받음.

`destroy()` : 서블릿 인스턴스가 메모리에서 제거될 때 호출되는 메서드. 웹 애플리케이션이 shutdown 될 때 컨테이너는 모든 서블릿의 `destroy()`를 호출함. `init()` 메서드에서 획득한 자원을 반납하고, 서블릿을 종료시킴. `init()` 과 마찬가지로 생명주기 동안 한 번만 호출되고, 서블릿이 지정한 시간 동안 호출되지 않거나 자원이 모자란 경우 임의로 서블릿의 `destroy()`를 호출해 해당 서블릿을 메모리에서 제거 가능함.

4.1 서블릿 생명주기

```
@WebServlet(  
    urlPatterns = {"/LifeCycle.do"},  
    initParams = {  
        @WebInitParam(name="email", value="gctserf@gmail.com"),  
        @WebInitParam(name="userid", value="eomyb")  
    })  
public class LifeCycle extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
    private ServletConfig config;  
  
    public LifeCycle() {  
        super();  
        System.out.println("서블릿 생성자 호출");  
    }  
  
    public void init(ServletConfig config) throws ServletException {  
        this.config = config;  
        System.out.println("init() 메서드 호출");  
    }  
  
    public void destroy() {  
        System.out.println("destroy() 메서드 호출");  
    }  
}
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
    System.out.println("doGet() 호출됨");  
    response.setContentType("text/html;charset=utf-8");  
    PrintWriter out = response.getWriter();  
    String email = config.getInitParameter("email");  
    String userid = config.getInitParameter("userid");  
    out.println("사용자 이메일 : " + email);  
    out.println("사용자 아이디 : " + userid);  
}  
}
```



2. 서블릿 Context

- 서블릿 Context

웹 애플리케이션은 HTML페이지, 미디어 파일, 데이터 또는 리소스 파일, 서블릿과 JSP 파일, 그리고 자바 클래스 등 여러 가지 정적/동적 리소스들을 포함하고 있음. ServletContext 객체는 웹 애플리케이션의 런타임 표현이라고 할 수 있고, 쉽게 말하면 웹 애플리케이션을 대표하는 객체라고 표현할 수 있음. ServletContext 객체는 컨텍스트 당 하나씩 생성되며, 웹 애플리케이션 내의 모든 자원과 연결될 수 있음.

Context를 알아야 하는 가장 중요한 이유는 데이터바인딩의 범위 때문

데이터바인딩 - ServletContext 객체 속에 데이터(객체)를 저장하는 것.
getParameter()와 다르게 String 타입이 아닌 객체를 반환

Request 객체에서도 동일하게 객체 바인딩 가능

- 서블릿 Context

Object `getAttribute(String name)` : 컨텍스트에 바인딩 되어 있는 객체를 반환함

Boolean `setAttribute(String name, Object value)` : 컨텍스트에 value 객체를 name 이름으로 바인딩함

Void `removeAttribute(String name)` : 컨텍스트에 바인딩 되어 있는 객체를 삭제

Enumeration<String> `getAttributeNames()` : 컨텍스트에 바인딩 되어 있는 객체의 이름들을 반환

ServletContext 객체를 얻으려면 ServletConfig 객체를 이용 해야 함

- `init()` 메서드 호출 시 인자로 넘기므로 `init()` 메서드에서 ServletConfig 객체를 멤버변수에 저장하면 `doGet()` 이나 `doPost()` 메서드에서 `getServletContext()` 메서드를 통해 사용 가능

- ServletConfig는 서블릿당 한 개, ServletContext는 컨텍스트마다 하나씩 만들어 짐(착각하지말것)

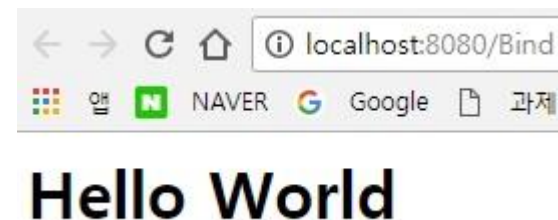
4.2 서블릿 Context

```
@WebServlet("/Bind")
public class BindServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    ServletConfig config;

    public BindServlet() {
        super();
    }

    public void init(ServletConfig config) throws ServletException {
        this.config = config;
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse
        response) throws ServletException, IOException {
        ServletContext application = config.getServletContext();
        application.setAttribute("message", "Hello World");
        response.setContentType("text/html;charset=utf-8");
        PrintWriter out = response.getWriter();
        out.println("<h1>" + application.getAttribute("message"));
    }
}
```



Message라는 이름의 “Hello World” 객체가 setAttribute()메서드를 통해 전달된 것을 확인 가능

Chapter

05 JSP

1. JSP

1. JSP

JSP란?

JSP 기술에서 웹 애플리케이션을 구현할 때 작성하는 코드를 'JSP 페이지' 라고 함. JSP 페이지는 HTML 문서 사이 사이에 JSP 문법의 코드가 삽입되는 형태로 작성됨.

JSP 페이지에 있는 HTML 코드는 웹 브라우저로 그대로 전송되지만 JSP 문법의 코드는 웹 컨테이너 쪽에서 실행되고 결과만 브라우저로 전송 됨.

실행 순서

JSP 페이지를 서블릿 클래스의 소스코드로 변환 – 소스코드 컴파일 – 컴파일 결과로 객체를 만듦 – 서블릿 실행

JSP의 기초 문법

JSP 코드 : `<%...%>` 의 형태로 자바 코드를 스크립트 안에 넣을 수 있는 형태

EL(Expression Language) : `${...}` 의 형태로 MVC Model2 구조에 적합하게 스크립팅 요소를 없애기 위해 나옴

JSTL : XML의 형태로 기술됨. 액션이라고도 함. 표준화된 태그 라이브러리를 이용함으로써 JSP 페이지의 가독성을 높여줌 ex) `<c:forEach>`

JSP 코드 문법

<%@ 지시자 %> : Directive 태그라고도 하며 지시하는 역할을 함. 주로 페이지 내의 인코딩 방식이나 태그 라이브러리를 설정하는데 쓰임

<%! 멤버(변수, 메서드) 선언%> : Declaration 태그. JSP파일 내에서 변수 또는 메서드를 선언할 때 사용.

<% 자바 코드 %> : Scriptlet 태그. 자바 코드를 작성합니다. 페이지 내에 여러 개의 블록이 존재 가능.

<%= 출력한 표현식 %> : 변수 또는 표현식을 출력하기 위한 태그. Expression 태그라고도 부름.

<%-- --%> : JSP 주석

* HTML 이나 Java 코드 주석도 활용 가능. (<!-- HTML Comments --!>, <% /* Java Comments */ %>)

JSP 코드 문법

<%@ page > : 페이지 디렉티브. 페이지 내의 언어, 상속받을 클래스, 패키지 import, session, errorPage, 인코딩 등의 속성을 지정할 수 있음

<%@ include file=> : 해당 파일의 소스코드를 그대로 입력. 유사한 문법으로 액션 태그의 <jsp:include page=>가 있음.

<%@ taglib uri = "WEB-INF/tag/taglib.tld" prefix="mytag" %>

JSP 내장 객체

Request : 요청 객체, 서블릿과 동일하게 사용 가능

Response : 응답 객체, 서블릿과 동일하게 사용 가능

Out : 웹 브라우저로 HTML 코드 출력

Session : 세션 객체, 세션을 통해 클라이언트의 상태 정보를 저장 가능

setAttribute(String Key, Object value) 객체 바인딩

getAttribute(String Key) : Object 객체 바인딩

removeAttribute(String Key) 객체 바인딩

getId() : String 세션의 아이디

invalidate() 세션 무효화(로그아웃 처리 시)

Application : 애플리케이션 객체, 서블릿의 컨텍스트 객체와 같음

JSP 내장 객체

pageContext : JSP 내장 객체를 참조할 수 있는 객체

Page : JSP 페이지 객체

Exception : 예외 객체, JSP에서 예외가 발생할 경우에만 생성
getMessage() : String 예외 발생 원인에 해당하는 메시지 출력
printStackTrace() 예외 발생 경로를 추적해 줌

Config : 서블릿마다 하나씩 생성되는 객체, 서블릿과 동일

Chapter

06 세션과 쿠키

1. 세션
2. 쿠키
3. `response.sendRedirect()/RequestDispatcher.forward()`
4. 예외처리

1. 세션

HTTP의 프로토콜은 Connectionless 프로토콜 – 클라이언트에서 서버에 요청하면 응답 후 연결 종료

이 특징 때문에 웹 서버는 방문한 사용자를 구분하지 못함. 서버에서 이전 방문자를 기억하기 위해선 **세션** 객체를 사용해 클라이언트의 정보를 저장해야 함.

세션 객체에 저장한 클라이언트의 정보로 서버는 각 클라이언트를 구분 할 수 있게 됨.

클라이언트에서 웹 서버에 접속할 때 요청 헤더에 세션ID를 포함시켜 요청하면 서버에서는 요청 헤더에 포함된 세션 ID와 서버에 저장되어 있는 세션 객체의 ID를 비교해 클라이언트를 인증

HttpSession은 request.getSession() 메서드를 이용해 얻고 사용 가능.

HttpSession 주요 메서드

Void setAttribute(String Key, Object Value) : 세션 객체에 Key로 Value를 바인딩 시킴

Object getAttribute(String Key) : 세션 객체에 key로 바인딩 되어 있는 값을 반환

Enumeration<String> getAttributeNames() : 세션 객체에 바인딩 되어 있는 객체의 key값 들을 반환

Void removeAttribute(String Key) : 세션 객체에 바인딩 되어 있는 객체를 삭제

String getId() : 세션의 id 반환

Void invalidate() : 세션 무효화

Void setMaxInactiveInterval(int second) : 세션의 유효시간을 초 단위로 지정

Int getMaxInactiveInterval() : 세션의 유효시간을 초 단위로 반환

2. 쿠키

쿠키는 웹 서버에서 response를 통해 클라이언트로 보내 지고 클라이언트 컴퓨터에 서버 도메인 이름으로 저장 됨

세션과 다르게 모든 쿠키 정보는 오직 클라이언트에 저장되고 관리 됨

Response.addCookie(Cookie) 메서드를 이용해 쿠키를 추가할 수 있고

Request.getCookie() 메서드를 통해 클라이언트가 보낸 쿠키를 조회할 수 있음

통상 서버에서 사용자가 로그인할 경우

1. 서버에 세션 객체를 생성한 다음 세션 ID를 response 메시지를 통해 클라이언트에 전달
2. 클라이언트는 세션 ID를 쿠키에 저장했다가 다음 요청 시에 request 메시지 헤더에 세션 ID를 포함시켜 요청

만약 사용자가 쿠키를 수신할 수 없는 상태라면(쿠키 차단)

세션 자체가 무의미해짐 – response로 쿠키를 보내도 다음 request 요청시에 세션ID를 보내지 못함

**3.response.sendRedirect()
RequestDispatcher.forward()**

6.3 response.sendRedirect()/RequestDispatcher.forward()

서블릿으로 데이터 처리 후 View로 데이터를 보내줄 때에는 대표적으로
response.sendRedirect() 메서드와 RequestDispatcher.forward() 메서드를 사용함.

두 메서드 다 URL 주소를 인수로 받음(String URL)

이 두 메서드가 중요한 이유는

서로의 전송 방식 차이 때문.

요청 시에 들어온 데이터가 View에서 쓰이지 않는 경우 - response.sendRedirect() 사용

요청 시에 들어온 데이터가 View에서도 쓰여야 하는 경우 - RequestDispatcher.forward() 사용

response.sendRedirect()

리다이렉트 메서드는 인수로 전달 된 URL 홈페이지를 재전송 해 줌

사용자의 화면에서는

(Redirect 메서드 호출 – 곧바로 Redirect의 URL에 해당하는 화면을 표시) 로 보이지만 실제로는 그렇지 않음

서버가 서블릿의 응답을 브라우저에 표시 – 브라우저는 Redirect(URL) 페이지를 새로 요청

Request와 response 객체는 클라이언트의 요청이 있을 시 생성되고 응답과 함께 소멸함

위와 같이 처음 클라이언트가 요청 – Redirect() 메서드 호출 – 브라우저가 다시 해당 URL 홈페이지 요청

의 과정을 거친다면 처음 클라이언트가 요청했을 시의 request와 response 객체는 없어짐 – 사용 불가

RequestDispatcher.forward()

디스패처포워드 메서드는 인수로 전달 된 URL 홈페이지에 request와 response 객체를 들고 갈 수 있게 해 줌

요청을 다시 하는 일이 없이 바로 URL 홈페이지로 객체를 전달함 – request 객체 사용 가능

request에 바인딩한 객체들도 그대로 사용 가능

클라이언트의 요청 시 입력 받은 데이터를 JSP에서 쓰고 싶다면

RequestDispatcher.forward() 메서드를 이용해야 함

4. 예외처리

6.4 예외처리

웹 애플리케이션은 언제나 여러 예외상황이 발생 가능 - ex) HTTP, 자바 등

따라서 구축 시에 이런 여러 예외 사항들을 고려한 프로그래밍을 해야 함

HTTP response 메시지의 상태 라인에는 응답 코드와 응답 메시지 라는 데이터가 있음

이를 통해 에러의 유형, 원인 등을 전송 받을 수 있어

그에 대한 메시지만 띄워주면 클라이언트와 서버 측 모두 오류를 파악하고 수정 가능

이와 같은 편리성을 위해 꼭 해야 하는 것이 **예외처리** 기능

HTTP 응답 코드

- 100번대 : 정보전송/ 임시적인 응답을 나타내는 것은 Status-Line과 선택적인 헤더들로 구성되어 있음
- 200번대 : 성공/ 클라이언트의 요구가 성공적으로 수신되어 처리되었음을 의미
200 – 성공 , 201 – POST 요청 처리, 204 – 전송할 데이터 없음
- 300번대 : 리다이렉션/ 해당 요구사항을 처리하기 위해서는 사용자 에이전트에 의해 수행되어야 할 추가적 사항이 있음
- 400번대 : 클라이언트 측 에러/ 클라이언트가 서버에게 보내는 요구 메시지를 완전히 처리하지 못한 경우 등의 클라이언트에서 오류가 발생한 경우 나옴
- 500번대 : 서버측 에러/ 서버 자체에서 발생한 오류상황이나 요구사항을 제대로 처리할 수 없을 때 사용

자주 사용되는 에러 코드

403 : 접근 권한이 없을 경우 발생

404 : 요청한 주소를 찾을 수 없는 경우(파일명/경로 오타, 대문자 소문자 구별해야함)

405 : 사용자가 요청한 방식을 서블릿에서 처리할 수 없는 경우

ex) 사용자가 get방식으로 요청했는데 doGet() 메서드가 서블릿에서 재정의 되어 있지 않은 경우

500 : 프로그램 오류. 클래스 파일이 없거나, 있어도 안의 코드 내용에서 오류가 있거나 하는 경우

HTTP 에러 처리

Web.xml 문서에서 선언적 방법으로 HTTP 에러 해결 가능.

```
<error-page>  
<error-code>404</error-code>  
<location>/error404.jsp</location>  
</error-page>
```

Web.xml 문서에 옆 내용의 코드를 추가하세요.

6.4 예외처리

```
<%@ page contentType="text/html; charset=UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>페이지를 찾을 수 없습니다.</title>
</head>
<body>
<h1>Error</h1>
<h2>Page Not Found</h2>
</body>
</html>
```

Web.xml 문서에 코드 추가 후

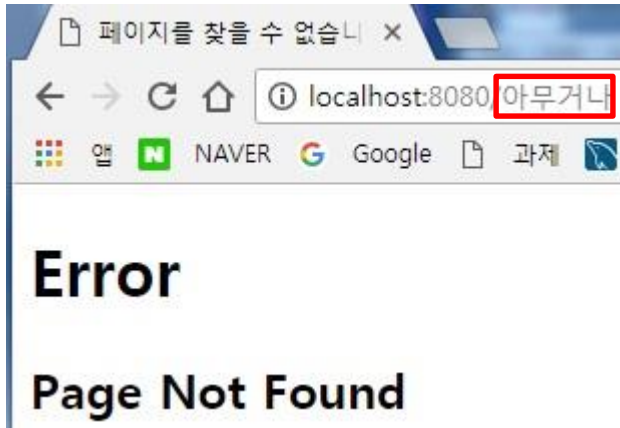
코드에 작성한 홈페이지 이름인

error404.jsp 를 만들어 주세요.

코드 내용은 아래와 같습니다.

6.4 예외처리

실행 화면



작성 완료 후

주소창에 존재하지 않는 페이지를 치면

위와 같이 404 코드 에러이므로

error404.jsp 페이지를 띄워주는 것을 확인 가능

Chapter

07 JDBC와 DAO패턴

1. SQL 기본 구문
2. JDBC
3. DAO패턴

1. SQL 기본 구문

데이터베이스에서 우리가 할 작업들은

데이터 조회(Select), 입력(insert), 수정(update), 삭제(delete)

이 4가지 기본 작업들에 대한 SQL 구문을 알지 못 하면 웹 애플리케이션 구축 자체가 불가능

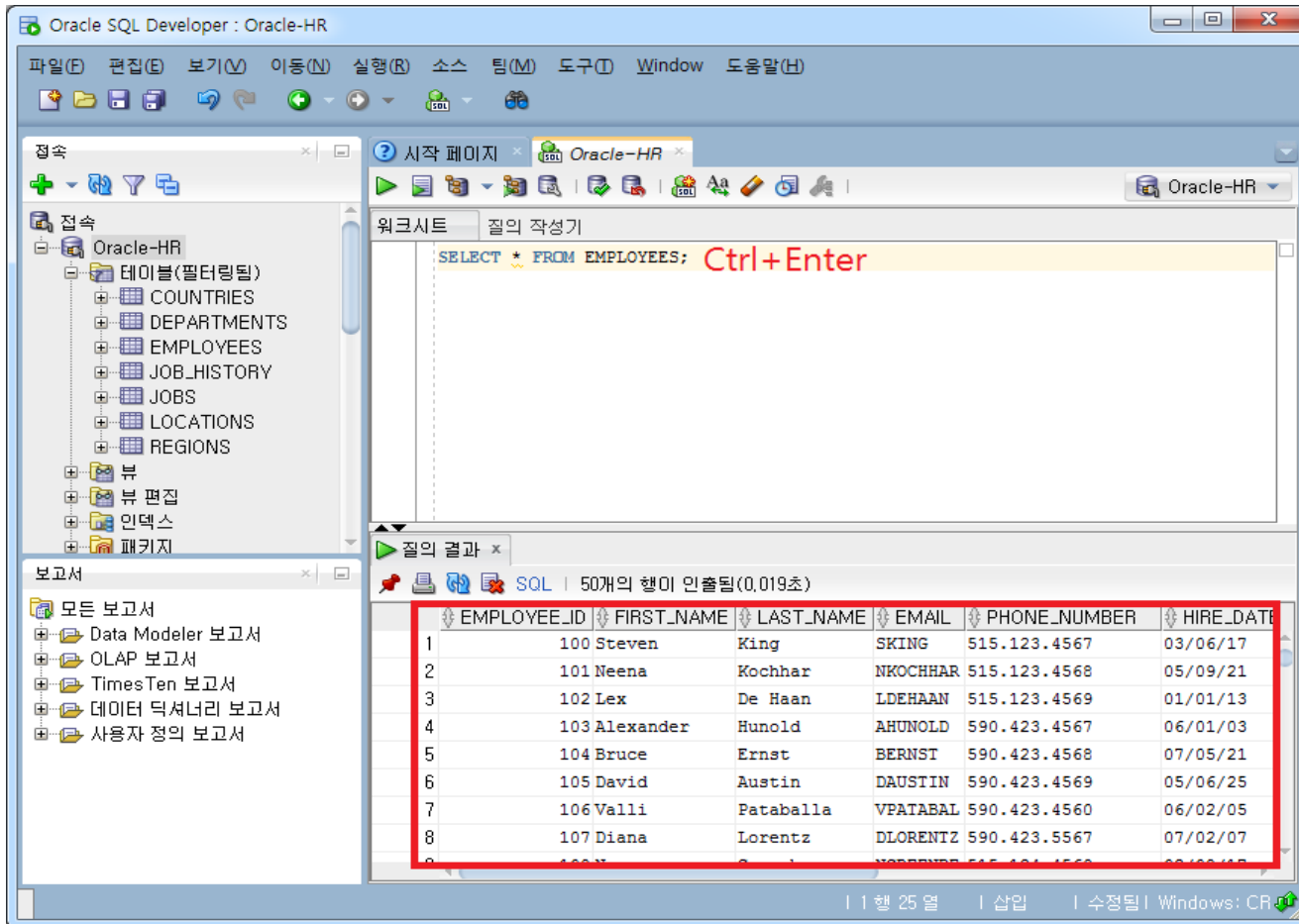
JDBC API 를 통해 자바 코드로 작성하지만 쿼리결과는 쿼리문으로 얻기 때문에 SQL 쿼리문 지식이 필요

최대한 간단하게 필요한 구문들만 작성해보고 JDBC로 넘어갈 것

SQL 구문 규칙

- 대소문자를 구별하지 않음
- 구문의 끝에 ; 을 입력해 구문의 끝임을 알려야 함
- 키워드는 여러 줄에 나누거나 단축될 수 없음
- 구문은 몇 줄이든 가능

SELECT 구문



SELECT (추출할 데이터들) FROM 테이블;

의 형식으로 작성됨

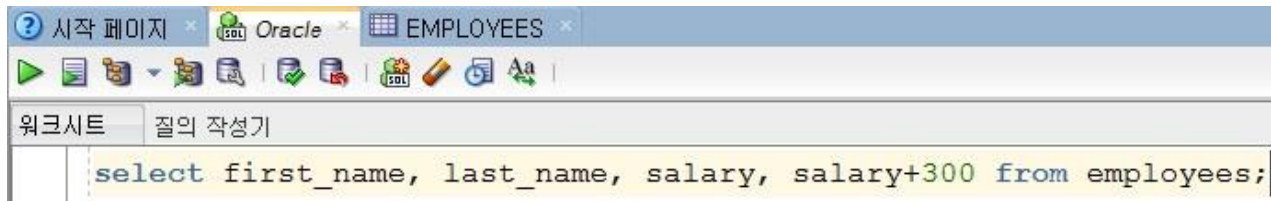
옆에 구문을 보면 * 은 테이블 전체 데이터를 뜻함

EMPLOYEES 라는 테이블의 전체 데이터를 SELECT(조회) 하겠다 라는 뜻임

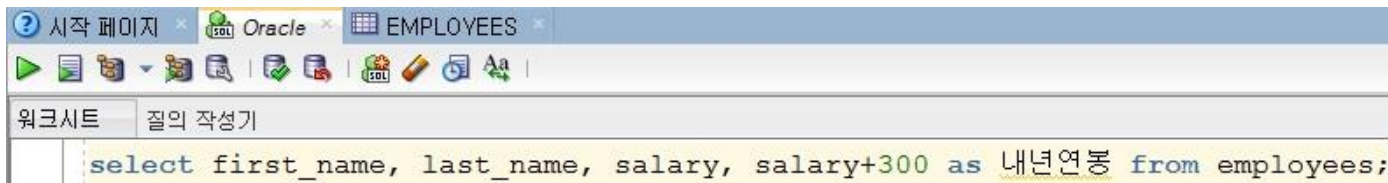
따라서 결과에 EMPLOYEES 테이블의 전체 데이터가 나오는 것을 확인 가능

SELECT 구문 (특정 열에 대해 계산/ 열 이름 변경)

EMPLOYEES 테이블에서 사람들의 성, 이름, 급여, 증가된 급여 조회



EMPLOYEES 테이블에서 사람들의 성, 이름, 급여, 증가된 급여를 내년연봉으로 열 이름을 바꿔서 조회



질의 결과 x

SQL | 50개의 행이 인출됨(0.007초)

	FIRST_NAME	LAST_NAME	SALARY	SALARY+300
1	Steven	King	24000	24300
2	Neena	Kochhar	17000	17300
3	Lex	De Haan	17000	17300
4	Alexander	Hunold	9000	9300
5	Bruce	Ernst	6000	6300
6	David	Austin	4800	5100
7	Valli	Pataballa	4800	5100
8	Diana	Lorentz	4200	4500
9	Nancy	Greenberg	12008	12308
10	Daniel	Faviet	9000	9300
11	John	Chen	8200	8500
12	Ismael	Sciarra	7700	8000
13	Jose Manuel	Urman	7800	8100
14	Luis	Popp	6900	7200
15	Den	Raphaely	11000	11300
16	Alexander	Khoo	3100	3400
17	Shelli	Baida	2900	3200
18	Sigal	Tobias	2800	3100
19	Guv	Himuro	2600	2900

질의 결과 x

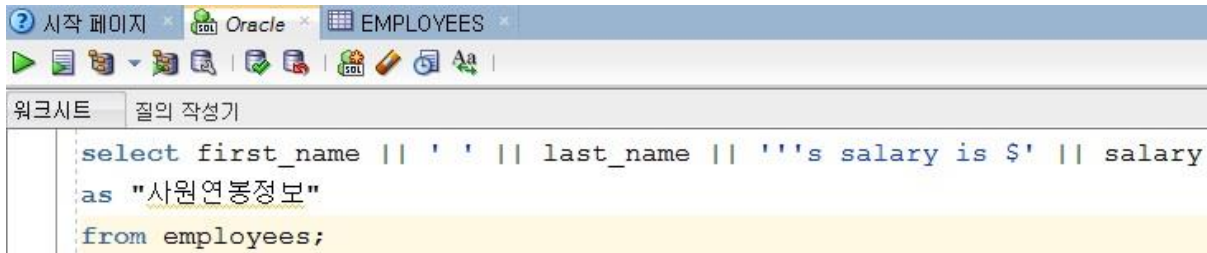
SQL | 50개의 행이 인출됨(0.006초)

	FIRST_NAME	LAST_NAME	SALARY	내년연봉
1	Steven	King	24000	24300
2	Neena	Kochhar	17000	17300
3	Lex	De Haan	17000	17300
4	Alexander	Hunold	9000	9300
5	Bruce	Ernst	6000	6300
6	David	Austin	4800	5100
7	Valli	Pataballa	4800	5100
8	Diana	Lorentz	4200	4500
9	Nancy	Greenberg	12008	12308
10	Daniel	Faviet	9000	9300
11	John	Chen	8200	8500
12	Ismael	Sciarra	7700	8000
13	Jose Manuel	Urman	7800	8100
14	Luis	Popp	6900	7200
15	Den	Raphaely	11000	11300
16	Alexander	Khoo	3100	3400
17	Shelli	Baida	2900	3200
18	Sigal	Tobias	2800	3100
19	Guv	Himuro	2600	2900

7.2 SQL 기본 구문

SELECT 구문 (특정 열들의 데이터를 합침)
(행 번호 출력)

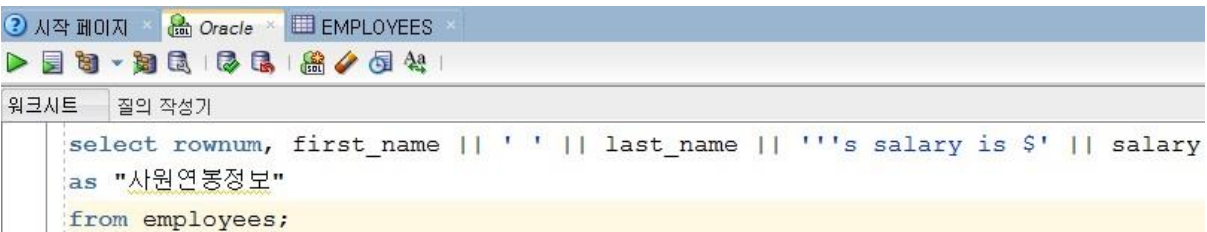
EMPLOYEES 테이블에서 사람들의 성, 이름, 급여를 한 열로 합침



```

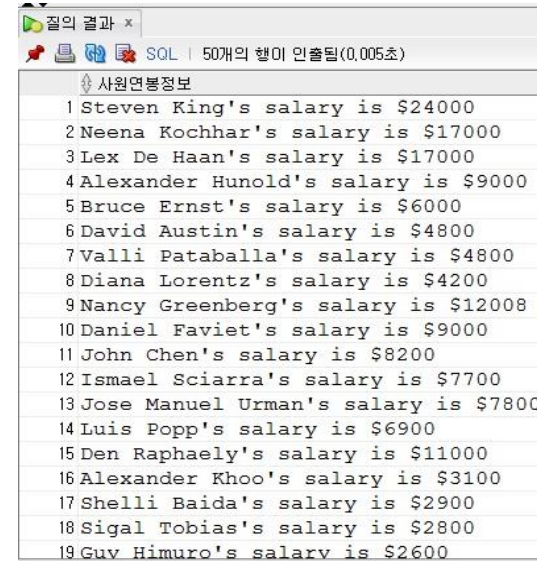
select first_name || ' ' || last_name || ''s salary is $' || salary
as "사원연봉정보"
from employees;
  
```

SELECT 한 결과에 대해 행 번호 부여



```

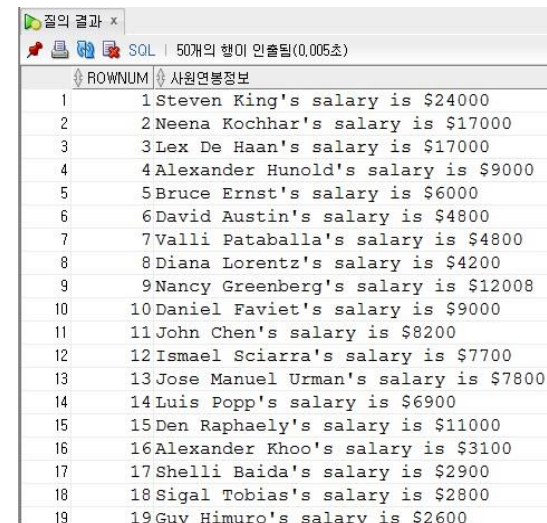
select rownum, first_name || ' ' || last_name || ''s salary is $' || salary
as "사원연봉정보"
from employees;
  
```



결과 x

SQL | 50개의 행이 인출됨(0.005초)

사원연봉정보	
1	Steven King's salary is \$24000
2	Neena Kochhar's salary is \$17000
3	Lex De Haan's salary is \$17000
4	Alexander Hunold's salary is \$9000
5	Bruce Ernst's salary is \$6000
6	David Austin's salary is \$4800
7	Valli Pataballa's salary is \$4800
8	Diana Lorentz's salary is \$4200
9	Nancy Greenberg's salary is \$12008
10	Daniel Faviert's salary is \$9000
11	John Chen's salary is \$8200
12	Ismael Sciarra's salary is \$7700
13	Jose Manuel Urman's salary is \$7800
14	Luis Popp's salary is \$6900
15	Den Raphaely's salary is \$11000
16	Alexander Khoo's salary is \$3100
17	Shelli Baida's salary is \$2900
18	Sigal Tobias's salary is \$2800
19	Guv Himuro's salary is \$2600



결과 x

SQL | 50개의 행이 인출됨(0.005초)

ROWNUM	사원연봉정보
1	1 Steven King's salary is \$24000
2	2 Neena Kochhar's salary is \$17000
3	3 Lex De Haan's salary is \$17000
4	4 Alexander Hunold's salary is \$9000
5	5 Bruce Ernst's salary is \$6000
6	6 David Austin's salary is \$4800
7	7 Valli Pataballa's salary is \$4800
8	8 Diana Lorentz's salary is \$4200
9	9 Nancy Greenberg's salary is \$12008
10	10 Daniel Faviert's salary is \$9000
11	11 John Chen's salary is \$8200
12	12 Ismael Sciarra's salary is \$7700
13	13 Jose Manuel Urman's salary is \$7800
14	14 Luis Popp's salary is \$6900
15	15 Den Raphaely's salary is \$11000
16	16 Alexander Khoo's salary is \$3100
17	17 Shelli Baida's salary is \$2900
18	18 Sigal Tobias's salary is \$2800
19	19 Guv Himuro's salary is \$2600

SELECT 구문 (특정 열의 특정 조건에 의한 데이터 조회)
WHERE 절 사용, BETWEEN 연산자 사용

EMPLOYEES 테이블에서 급여가 10000 이상인 사람만 조회

```
SELECT FIRST_NAME || ' ' || LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY >= 10000;
```



질의 결과 x

SQL | 인출된 모든 행: 19(0.004초)

	FIRST_NAME ' ' LAST_NAME	SALARY
1	Steven King	24000
2	Neena Kochhar	17000
3	Lex De Haan	17000
4	Nancy Greenberg	12008
5	Den Raphaely	11000
6	John Russell	14000
7	Karen Partners	13500
8	Alberto Errazuriz	12000
9	Gerald Cambrault	11000
10	Eleni Zlotkey	10500
11	Peter Tucker	10000
12	Janette King	10000
13	Clara Vishney	10500
14	Lisa Ozer	11500
15	Harrison Bloom	10000
16	Ellen Abel	11000
17	Michael Hartstein	13000
18	Hermann Baer	10000
19	Shelley Higgins	12008

EMPLOYEES 테이블에서 급여가 5000 이상 10000 이하인 사람

```
SELECT FIRST_NAME || ' ' || LAST_NAME, SALARY
FROM EMPLOYEES
WHERE SALARY BETWEEN 5000 AND 10000;
```



질의 결과 x

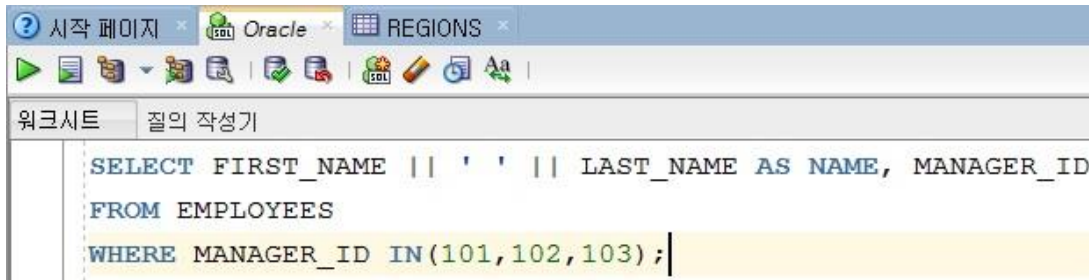
SQL | 인출된 모든 행: 43(0.004초)

	FIRST_NAME ' ' LAST_NAME	SALARY
1	Alexander Hunold	9000
2	Bruce Ernst	6000
3	Daniel Faviert	9000
4	John Chen	8200
5	Ismael Sciarra	7700
6	Jose Manuel Urman	7800
7	Luis Popp	6900
8	Matthew Weiss	8000
9	Adam Fripp	8200
10	Payam Kaufling	7900
11	Shanta Vollman	6500
12	Kevin Mourgous	5800
13	Peter Tucker	10000
14	David Bernstein	9500
15	Peter Hall	9000
16	Christopher Olsen	8000
17	Nanette Cambrault	7500
18	Oliver Tuvault	7000
19	Janette King	10000

7.2 SQL 기본 구문

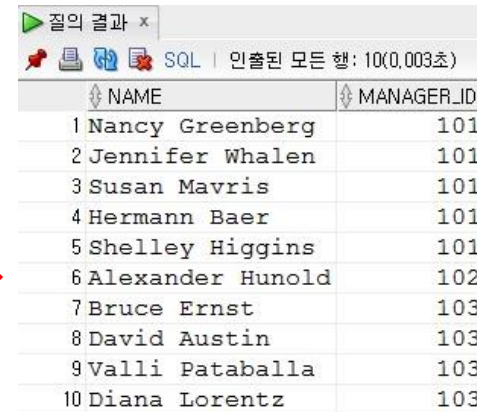
SELECT 구문 (특정 열의 특정 조건에 의한 데이터 조회)
IN 연산자 사용, LIKE 연산자 사용

MANAGER_ID 가 101, 102, 103인 사람만 조회



```

SELECT FIRST_NAME || ' ' || LAST_NAME AS NAME, MANAGER_ID
FROM EMPLOYEES
WHERE MANAGER_ID IN(101,102,103);
  
```

	NAME	MANAGER_ID
1	Nancy Greenberg	101
2	Jennifer Whalen	101
3	Susan Mavris	101
4	Hermann Baer	101
5	Shelley Higgins	101
6	Alexander Hunold	102
7	Bruce Ernst	103
8	David Austin	103
9	Valli Pataballa	103
10	Diana Lorentz	103

입사 년도가 04년도인 사람만 조회



```

SELECT FIRST_NAME || ' ' || LAST_NAME AS NAME, HIRE_DATE
FROM EMPLOYEES
WHERE HIRE_DATE LIKE '04%';
  
```




	NAME	HIRE_DATE
1	Matthew Weiss	04/07/18
2	Jason Mallin	04/06/14
3	John Russell	04/10/01
4	Janette King	04/01/30
5	Patrick Sully	04/03/04
6	Allan McEwen	04/08/01
7	Ellen Abel	04/05/11
8	Nandita Sarchand	04/01/27
9	Sarah Bell	04/02/04
10	Michael Hartstein	04/02/17

데이터 정렬

ORDER BY 절 사용 – ASC : 오름차순(디폴트값) / DESC : 내림차순

```
SELECT * FROM EMPLOYEES ORDER BY HIRE_DATE;
```

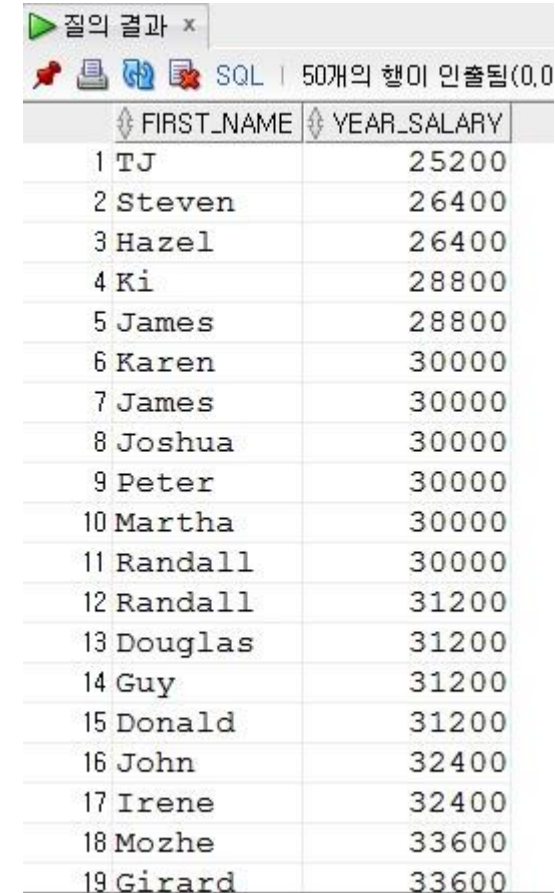
- 입사일 순으로 정렬(입사한지 가장 오래된 순서대로 나옴)

열 이름에 의한 정렬 뿐만 아니라 열의 순서를 나타내는 숫자로도 가능

```
SELECT FIRST_NAME, SALARY*12 AS YEAR_SALARY FROM EMPLOYEES ORDER BY YEAR_SALARY;
```

```
SELECT FIRST_NAME, SALARY*12 AS YEAR_SALARY FROM EMPLOYEES ORDER BY 2;
```

위 두 개의 쿼리문은 둘 다 오른쪽과 같은 결과를 출력

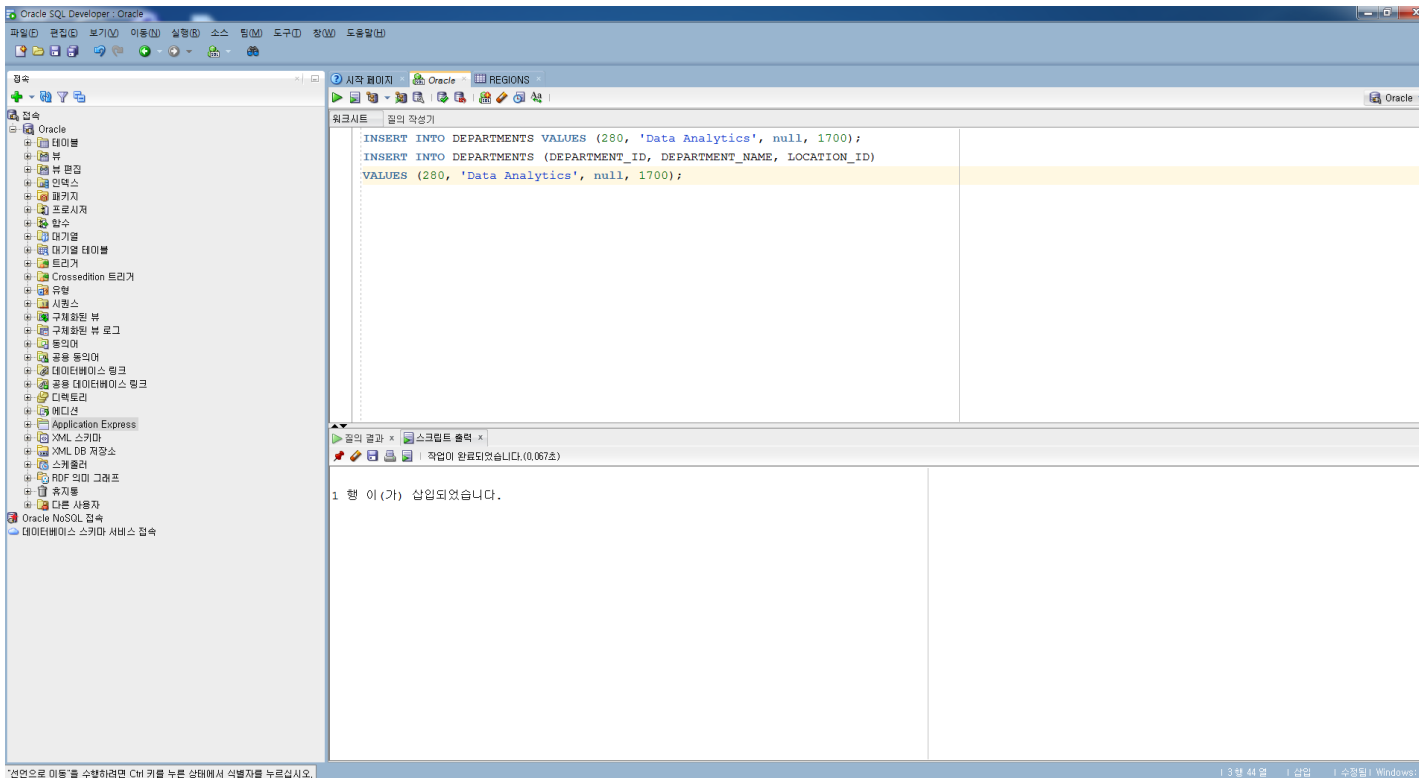


질의 결과 x

SQL | 50개의 행이 인출됨(0.0

	FIRST_NAME	YEAR_SALARY
1	TJ	25200
2	Steven	26400
3	Hazel	26400
4	Ki	28800
5	James	28800
6	Karen	30000
7	James	30000
8	Joshua	30000
9	Peter	30000
10	Martha	30000
11	Randall	30000
12	Randall	31200
13	Douglas	31200
14	Guy	31200
15	Donald	31200
16	John	32400
17	Irene	32400
18	Mozhe	33600
19	Girard	33600

INSERT 구문



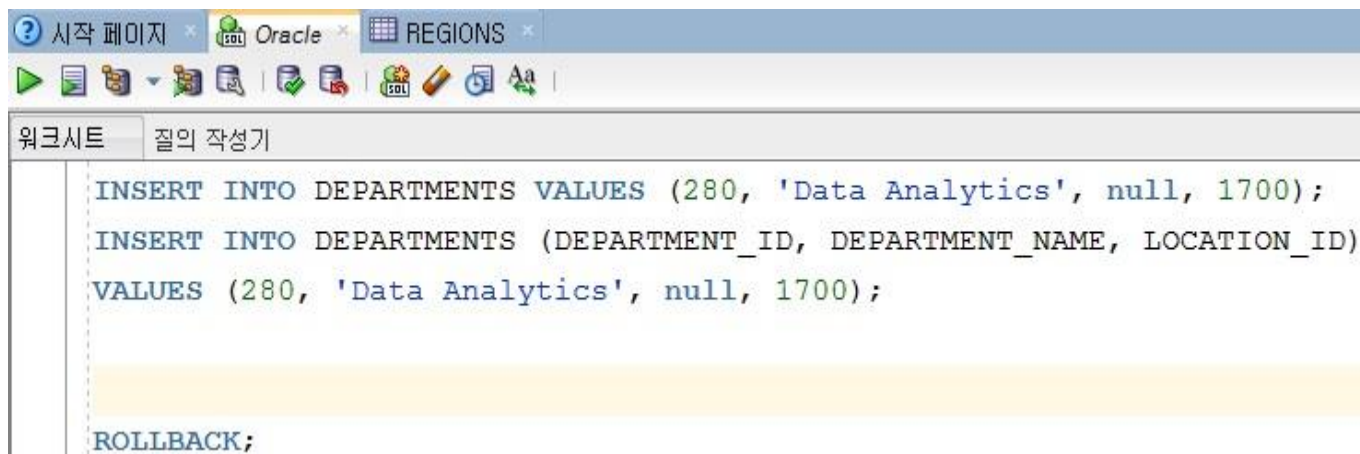
INSERT INTO 테이블 VALUES (데이터 값들)
또는
INSERT INTO 테이블 (테이블 열 지정) VALUES
(데이터 값들)

의 형식으로 작성.

결과로는 삽입된 데이터의 값이 아니라
몇 행이 삽입되었는지 출력됨.

INSERT 구문

COMMIT; / ROLLBACK;



```
INSERT INTO DEPARTMENTS VALUES (280, 'Data Analytics', null, 1700);
INSERT INTO DEPARTMENTS (DEPARTMENT_ID, DEPARTMENT_NAME, LOCATION_ID)
VALUES (280, 'Data Analytics', null, 1700);

ROLLBACK;
```

INSERT, UPDATE, DELETE 등과 같이

데이터베이스를 수정하는 구문은

COMMIT; 이라는 명령어를 치기 전까지는

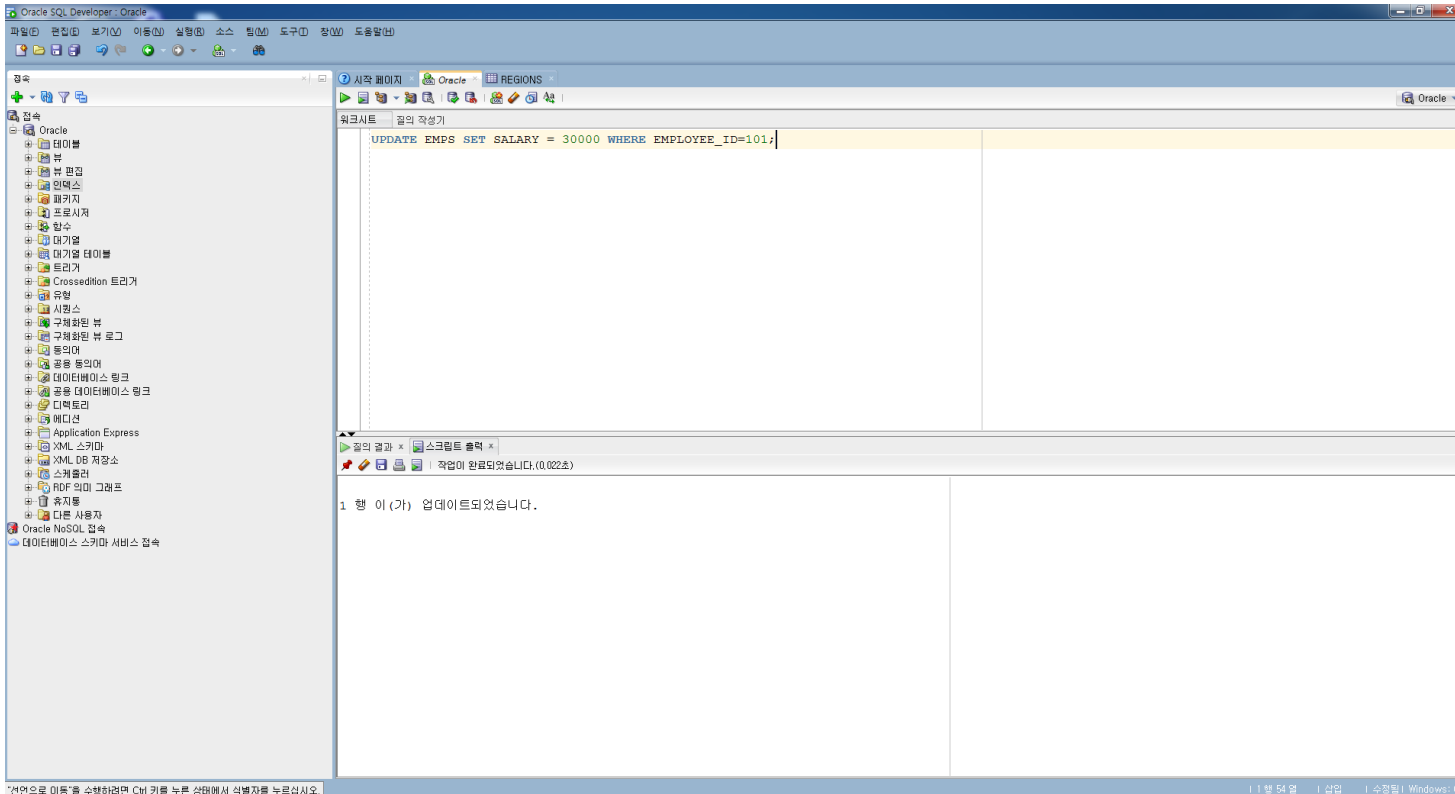
저장이 안 됨

따라서 COMMIT을 아직 치지 않았다면

ROLLBACK 명령어로 수정 전 상태로 돌리기 가능

ROLLBACK 은 한 트랜잭션 씩
COMMIT은 현재까지 진행한 모든 트랜잭션에 적용

UPDATE 구문



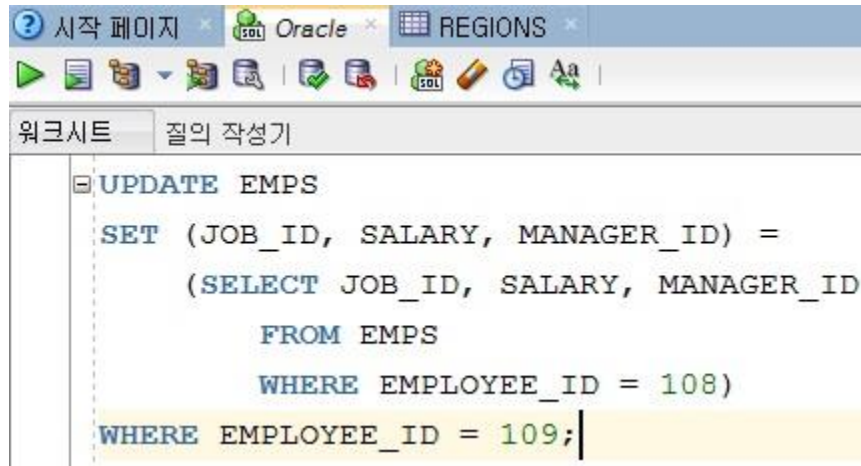
UPDATE 테이블 SET 열 이름 = 바꿀 데이터
WHERE 절 사용

의 형식으로 작성.

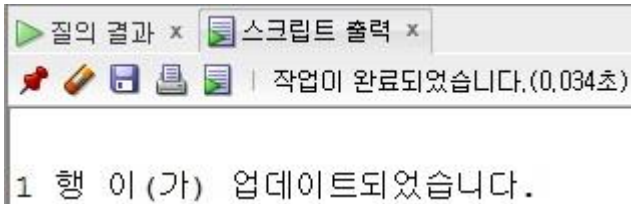
INSERT와 마찬가지로 결과는 몇 행이 업데이트
되었는지가 나옴.

마찬가지로 COMMIT 명령어를 작성 해야
사항 변경이 적용 됨

UPDATE 구문



```
UPDATE EMP
SET (JOB_ID, SALARY, MANAGER_ID) =
    (SELECT JOB_ID, SALARY, MANAGER_ID
     FROM EMP
     WHERE EMPLOYEE_ID = 108)
WHERE EMPLOYEE_ID = 109;
```



```
질의 결과 x 스크립트 출력 x
작업이 완료되었습니다.(0.034초)

1 행 이 (가) 업데이트되었습니다.
```

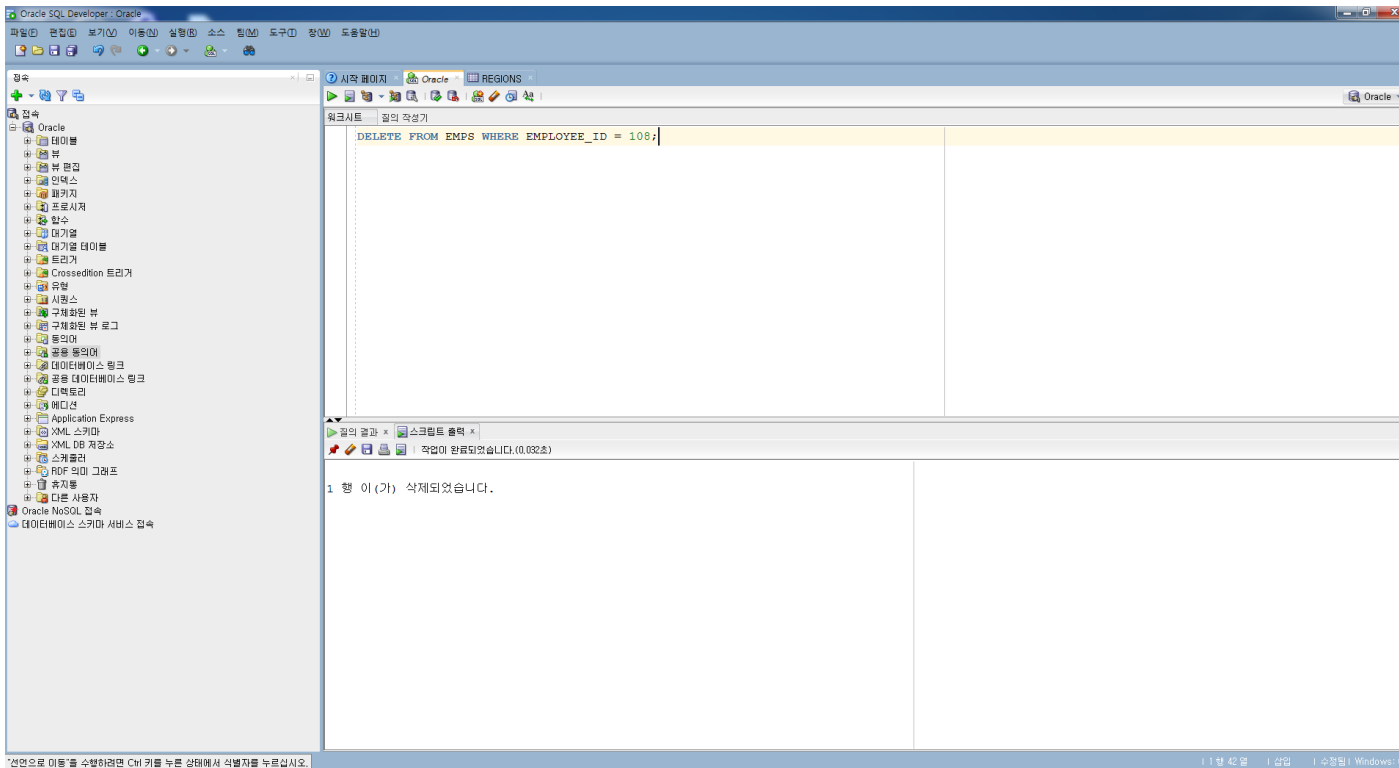
UPDATE 구문도 서브쿼리 작성 가능

옆 코드는

사원번호가 109번인 사람의
JOB_ID, SALARY, MANAGER_ID 를

사원번호가 108번인 사람의 데이터로
바꾸겠다는 뜻

DELETE 구문



DELETE FROM 테이블 WHERE 절
의 형식으로 작성.

결과는 행의 개수 반환

WHERE절에 서브쿼리 작성 가능

DELETE 구문



명령의 1 행에서 시작하는 중 오류 발생 -

```
DELETE FROM EMPSS
WHERE EMPLOYEE_ID =103
```

오류 보고 -

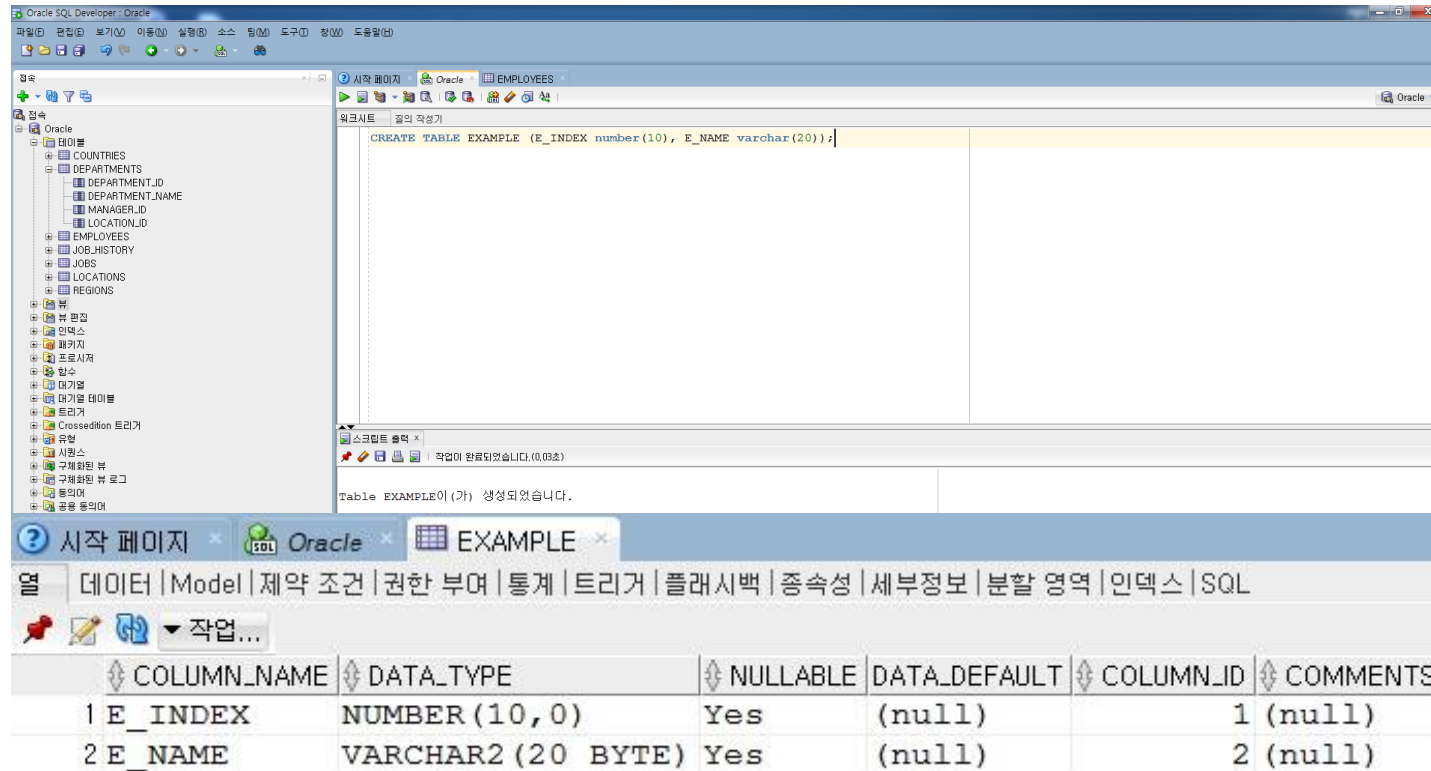
```
ORA-02292: integrity constraint (HR.EMPSS_MANAGER_FK) violated - child record found
```

DELETE할 때 주의할 점

모든 행이 삭제되지는 않음

데이터베이스의 구조상 무결성 조건들이 걸려
있는 행들은 무결성 제약조건을 위반할 수 없
기 때문에
제약조건에 따라 삭제 해야 함

CREATE 구문



CREATE TABLE EXAMPLE (E_INDEX number(10), E_NAME varchar(20));

Table EXAMPLE이 (가) 생성되었습니다.

	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1	E_INDEX	NUMBER (10,0)	Yes	(null)	1	(null)
2	E_NAME	VARCHAR2 (20 BYTE)	Yes	(null)	2	(null)

CREATE 테이블 (열, 타입(크기)
열, 타입(크기)
열, 타입(크기));

형식으로 작성.

TABLE ***이(가) 생성되었습니다. 로 결과 표현

DESCRIBE 테이블 명령어로 테이블 구조 확인 가능

2. JDBC

JDBC란? – Java Database Connectivity 의 약어. 자바에서 데이터베이스에 접속할 수 있도록 하는 자바 API 이다.

자바로 데이터베이스에 접속해 데이터베이스의 기능을 다룰 수 있게 해 줌

여러 데이터베이스가 있지만 본 수업에선 Oracle의 Database 11g Express Edition을 사용할 예정

Oracle 데이터베이스는 SQL 언어를 사용하기 때문에 기본 SQL에 대한 이해가 있어야 사용 가능

JDBC를 사용하려면 (자바 + 해당 데이터베이스의 언어) 에 대한 지식이 있어야 가능함

- 실행 순서

드라이버 로딩 – 커넥션 연결 – 쿼리문 실행 후 Statement 객체에 쿼리문 결과 저장 – 자바에서 소비

3. DAO패턴

DAO패턴이란?

Data Access Object 의 약어

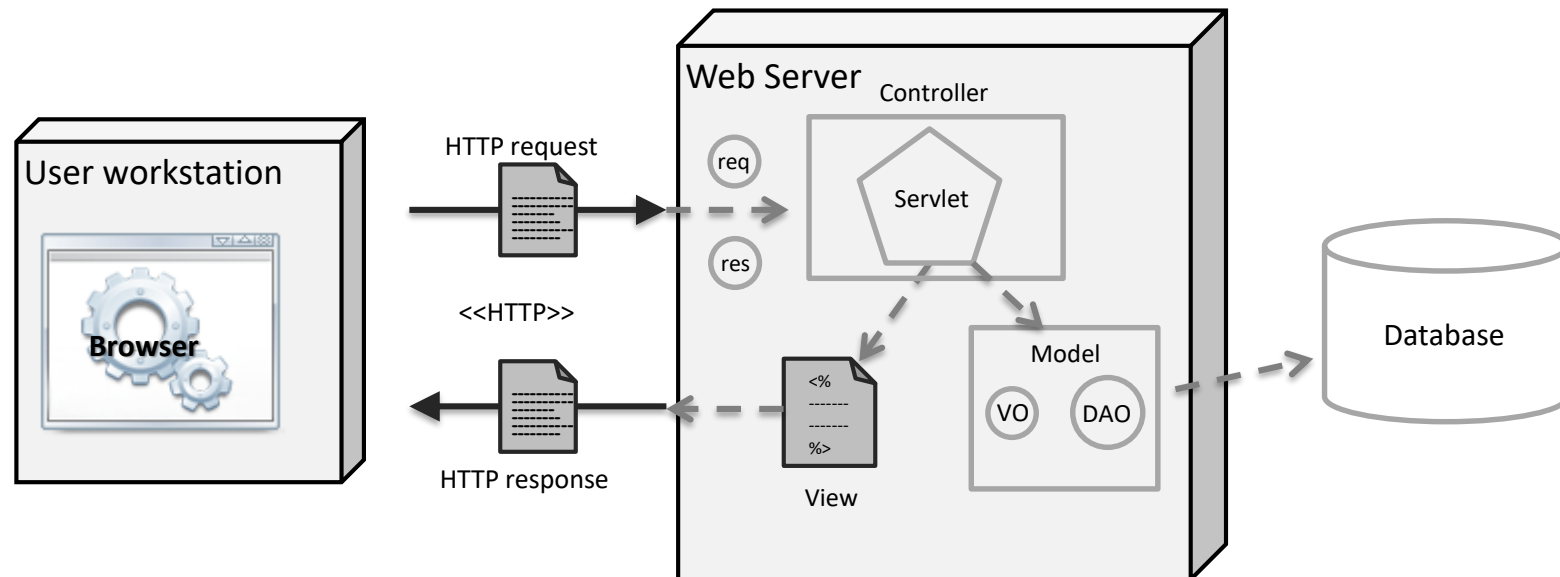
서블릿 – 요청, 응답 처리 컨트롤러(사장)

데이터베이스 – 요리 재료

모델 – DAO(Data Access Object) : 데이터를 처리하는 메서드 구현(요리사)

VO(Value Object) : 데이터베이스에서 조회한 데이터를 담을 틀(요리 그릇)

뷰 – 데이터를 브라우저의 형태로 사용자에게 보여줌(최종 요리)



DAO패턴 이란?

- 요점은 서블릿에서 바로 DB에 접근하는 것이 아니라 서블릿과 데이터베이스 사이에 DAO를 두는 것

서블릿 실행 시 커넥션 생성 - 서블릿이 요청을 받음 - 요청 받은 결과를 만들기 위한 데이터 처리를 하는 DAO 메서드를 호출 - DAO메서드는 생성된 커넥션을 통해 데이터베이스와 통신해 데이터를 받고 처리해서 결과를 VO 클래스 객체들에 담아서 내놓음 - 서블릿은 이 객체(데이터) 들을 받아 JSP에 뿌려줌 - JSP는 객체를 표현하는 페이지 코드만 있으면 됨

장점

- 유지보수가 더 쉬워짐
- 비즈니스 로직과 data access 로직을 독립적으로 갈라놓음(DB가 변하면 DAO메서드만 바꾸면 됨)
- 재사용이 가능하고 시스템 변화에 대해 유연해짐
- front-end 개발과 back-end 개발이 나누어짐

Chapter

08 스크립팅 요소 제거(EL, JSTL)

1. EL
2. JSTL

1. EL(Expression Language)

EL(Expression Language)

`${...}` 형태로 기술하는 언어

JSP 의 4가지 기본객체(page, request, session, application)가 제공하는 scope의 속성을 사용 가능 – 4가지 기본 객체 호출 가능

자바 에서처럼 문자열+문자열의 연산은 불가

페이지에 member 객체를 바인딩해서 보낸 경우 객체의 이름을 추가하려면

자바 문법이면 member.getName() 을 통해 출력해야 하지만

EL은 `${member.name}` 으로 가능 – 편리성 증대

2. JSTL

JSTL(JSP Standard Tag Library)

표준화된 태그 라이브러리 들을 제공함으로써 보다 편리하게 개발할 수 있도록 해주는 기능

간단한 태그로 기능구현까지 가능하게 함

본 교안에서는 Core 태그만 사용할 것

태그 종류는

XML processing, Formatting, Database access, Funtions 까지 총 5개가 존재

JSTL(JSP Standard Tag Library)

무슨 라이브러리를 사용 하든 taglib 지시자를 항상 페이지 맨 위에 설정해야 함

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

위와 같은 코드를 써주고

D:\projects\apache-tomcat-9.0.6\webapps\examples\WEB-INF\lib 경로에 있는 jar 파일들을

WEB-INF 속 lib 폴더에 넣어줘야 JSTL 코드를 사용할 수 있음

<c:out>, <c:if>, <c:choose>, <c:forEach> 등의 형식으로 사용

JSTL(JSP Standard Tag Library)

c:set : 변수 선언 시 사용

`<c:set var="b" value="20" scope="request"></c:set>` request에 b라는 파라미터를 20값으로 저장

c:if : if 조건문

`<c:if test="{a le 10}">` a가 10보다 작으면 뒤의 내용 실행, 크면 실행 안함

c:forEach : for문과 비슷한 반복문

`<c:forEach var="i" begin="1" end="{a}">` 밑의 내용을 1부터 i씩 더해 a번까지 실행

`<c:forEach var="list" items="{lists}">` lists에서 list 아이템을 하나씩 전부 꺼냄

c:choose : 자바의 switch 문과 비슷한 조건문

`<c:choose>`

`<c:when test="{a==10}">` a가 10이면 이 내용을 실행. 스위치 구문과 똑같은 방식으로 작성

JSTL(JSP Standard Tag Library)

Custom Tag

태그 파일/ 클래스 등을 통해 JSP 안의 내용을 해당 코드로 채울 수 있음

주로 CSS 적용 시 (Header, Footer)나 때로는 페이지 전체를 태그로 처리하는 경우도 있음

WEB-INF 폴더 안에 tags 폴더를 만들고 확장자를 .tag 명으로 작성해 놓고 JSP 페이지에서 지시자 코드에서 불러서 사용

<%@ taglib prefix="jk" tagdir="경로" %> 형태로 선언