**이름으로 조회하기**

2022-06-05

이승진

**학습목표**

조회 조건으로 DB 조회하기

SQL LIKE 연산자

**목차**

[1. 개요 2](#_Toc87249658)

[1) LIKE 연산자 2](#_Toc87249659)

[2. 조회 기능 구현 4](#_Toc87249660)

[1) StudentMapper.java 수정 4](#_Toc87249661)

[2) StudentController.java 수정 5](#_Toc87249662)

[3) student/list1.jsp 생성 6](#_Toc87249663)

[4) 실행 과정 7](#_Toc87249664)

[5) 실습 11](#_Toc87249665)

# 개요

학생 이름으로 학생 목록을 조회하는 기능을 구현하자.

## LIKE 연산자

SQL의 LIKE 연산자는 문자열 패턴을 비교한다.

|  |
| --- |
| name LIKE '**문자열**%' |

name 필드 값의 시작 부분이 '**문자열**'로 시작하면 true 이다.

|  |
| --- |
| name LIKE '%**문자열**%' |

name 필드 값에 '**문자열**'이 들어있으면 true 이다.

|  |
| --- |
| name LIKE '%**문자열**' |

name 필드 값이 끝부분이 '**문자열**'로 끝나면 true 이다.

### 학생 이름으로 조회

|  |
| --- |
| SELECT s.\*, d.name departmentName  FROM student s LEFT JOIN department d ON s.departmentId = d.id  WHERE s.name LIKE '이%' |

학생 이름이 '이'로 시작하는 레코드를 조회하는 SQL

Graphical user interface

Description automatically generated with medium confidence

|  |
| --- |
| SELECT s.\*, d.name departmentName  FROM student s LEFT JOIN department d ON s.departmentId = d.id  WHERE s.name LIKE '%' |

전체 레코드가 조회된다.

Table

Description automatically generated with low confidence

# 조회 기능 구현

## StudentMapper.java 수정

### src/main/java/net/skhu/mapper/StudentMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Student;  @Mapper  public interface StudentMapper {  @Select("SELECT s.\*, d.name departmentName " +  "FROM student s JOIN department d ON s.departmentId = d.id")  List<Student> findAll();  @Select("SELECT s.\*, d.name departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id " +  "WHERE s.name LIKE #{name}")  List<Student> findByName(String name);  } |

노란색 코드가 추가되었다.

|  |
| --- |
| @Select("SELECT s.\*, d.name departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id " +  "WHERE s.name LIKE #{name}") |

SQL 명령이 길기 때문에 문자열 세 개로 나눠서 입력하였다.

이 세 문자열은 덧셈(+) 연산자에 의해서 하나로 합쳐지는데, 컴파일 할 때 합쳐진다.

주황색으로 표시한 공백이 없다면 두 문자열이 하나로 합쳐졌을 때 WHERE 앞에 공백이 없어서 SQL 문법 오류가 발생한다.

List<Student> findByName(String name); 메소드의 name 파라미터 변수 값이

SQL 명령의 #{name} 부분에 채워져서 실행된다.

이것은 #{name} 부분을 mybatis 파라미터라고 부른다.

## StudentController.java 수정

### src/main/java/net/skhu/controller/StudentController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | package net.skhu.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import net.skhu.mapper.StudentMapper;  @Controller  public class StudentController {  @Autowired StudentMapper studentMapper;  @RequestMapping("student/list")  public String list(Model model) {  model.addAttribute("students", studentMapper.findAll());  return "student/list";  }  @RequestMapping("student/list1")  public String list1(Model model, String srchText) {  if (srchText == null) srchText = "";  model.addAttribute("students", studentMapper.findByName(srchText + "%"));  model.addAttribute("srchText", srchText);  return "student/list1";  }  } |

노란색 코드가 추가되었다.

## student/list1.jsp 생성

### src/main/webapp/WEB-INF/views/student/list1.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  <style>  div.container { width: 900px; margin: 50px auto; }  thead th { background-color: #eee; }  table{ border-collapse: collapse; width: 100%; }  td, th { padding: 4px; border: 1px solid lightgray; }  td:nth-child(4) { text-align: center; }  input { padding: 5px; font-size: 10pt; }  button { margin: 10px ; padding: 0.4em 2em; }  </style>  </head>  <body>  <div class="container">  <h1>학생목록</h1>  <form>  <label>이름</label>  <input type="text" name="srchText" value="${ srchText }" placeholder="검색조건" />  <button type="submit">조회</button>  </form>    <table class="table table-bordered table-condensed">  <thead>  <tr>  <th>학번</th>  <th>이름</th>  <th>학과</th>  <th>전화</th>  <th>성별</th>  <th>이메일</th>  </tr>  </thead>  <tbody>  <c:forEach var="student" items="${ students }">  <tr>  <td>${ student.studentNo }</td>  <td>${ student.name }</td>  <td>${ student.departmentName }</td>  <td>${ student.phone }</td>  <td>${ student.sex }</td>  <td>${ student.email }</td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

<input type="text" name="srchText" value="${ srchText }" placeholder="검색조건" />

위 input 태그에 입력된 검색 키워드가

srchText 이름의 request parameter가 되어 서버에 전달된다.

이 request parameter는, 아래 액션 메소드의 srchText 변수에 채워져서 전달된다

@RequestMapping("student/list1")

public String list1(Model model, String srchText) {

## 실행 과정

(1)

http://localhost:8088/student/list1

웹브라우저에서 위 URL을 입력하면, 웹서버에 위 URL이 GET 방식으로 요청된다.

이때 request parameter는 없다.

(2)

|  |
| --- |
| @RequestMapping("student/list1")  public String list1(Model model, String srchText) {  if (srchText == null) srchText = "";  model.addAttribute("students", studentMapper.findByName(srchText + "%"));  model.addAttribute("srchText", srchText);  return "student/list1";  } |

서버에서 list1 액션 메소드가 호출된다.

request parameter가 없으므로, srchText 파라미터 변수의 값은 null 이다.

(3)

if (srchText == null) srchText = "";

srchText 변수에 "" 문자열이 대입된다.

(4)

studentMapper.findByName(srchText + "%")

findByName 메소드가 호출된다. 파라미터 값은 "%" 이다.

(5)

SELECT s.\*, d.name departmentName FROM student s LEFT JOIN department d ON s.departmentId = d.id WHERE s.name LIKE '%'

SQL 명령이 실행된다.

student 테이블의 모든 레코드가 조회된다.

조회된 학생 목록이 List<Student> 타입의 객체에 채워져 리턴된다.

(6)

model.addAttribute("students", studentMapper.findByName(srchText + "%"));

조회된 학생 목록 객체가 "students" 이름의 model attribute가 되어 뷰에 전달된다.

(7)

model.addAttribute("srchText", srchText);

srchText 변수의 값이 "srchText" 이름의 model attribute가 되어 뷰에 전달된다.

이 변수의 값은 "" 문자열이다.

(8)

return "student/list1";

student/list1.jsp 뷰 파일이 실행된다.

<input type="text" name="srchText" value="${ srchText }" placeholder="검색조건" />

value="${ srchText }" 부분에 출력되는 값은 value="" 이다.

<c:forEach var="student" items="${ students }">

...

</c:forEach>

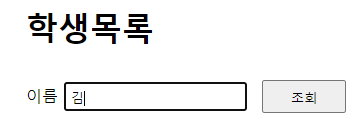
전체 학생 목록이 출력된다.

Table

Description automatically generated

서버에서 출력된 내용이 웹브라우저 창에 그려진다.

위 화면에서 이름 부분에 다음과 같이 입력하고 조회 버튼을 누르자.



(1)

<form>

<label>이름</label>

<input type="text" name="srchText" value="${ srchText }" placeholder="검색조건" />

<button type="submit">조회</button>

</form>

위 submit 버튼이 눌려졌으므로, HTTP request가 서버에 요청되어야 한다.

form 태그에 URL이 명시되지 않았으므로 웹브라우저창의 현재 URL이 서버에 요청된다.

form 태그에 method="POST"가 없으므로 GET 방식으로 요청된다.

입력폼에 입력된 값이 request parameter가 된다.

|  |  |
| --- | --- |
| request parameter 이름 | request parameter 값 |
| srchText | 김 |

GET 방식 요청이므로 request parameter가 URL의 query string이 된다.

(2)

|  |
| --- |
| @RequestMapping("student/list1")  public String list1(Model model, String srchText) {  if (srchText == null) srchText = "";  model.addAttribute("students", studentMapper.findByName(srchText + "%"));  model.addAttribute("srchText", srchText);  return "student/list1";  } |

서버에서 list1 액션 메소드가 호출된다.

srchText 이름의 request parameter 값이 srchText 파라미터 변수에 채워져 전달된다.

이 값은 "김" 이다.

(3)

studentMapper.findByName(srchText + "%")

findByName 메소드가 호출된다. 파라미터 값은 "김%" 이다.

(4)

SELECT s.\*, d.name departmentName FROM student s LEFT JOIN department d ON s.departmentId = d.id WHERE s.name LIKE '김%'

SQL 명령이 실행된다.

student 테이블에서 이름이 "김"으로 시작하는 레코드들이 조회된다.

조회된 학생 목록이 List<Student> 타입의 객체에 채워져 리턴된다.

(5)

model.addAttribute("students", studentMapper.findByName(srchText + "%"));

조회된 학생 목록 객체가 "students" 이름의 model attribute가 되어 뷰에 전달된다.

(6)

model.addAttribute("srchText", srchText);

srchText 변수의 값이 "srchText" 이름의 model attribute가 되어 뷰에 전달된다.

이 변수의 값은 "김" 문자열이다.

(7)

return "student/list1";

student/list1.jsp 뷰 파일이 실행된다.

<input type="text" name="srchText" value="${ srchText }" placeholder="검색조건" />

value="${ srchText }" 부분에 출력되는 값은 value="김" 이다.

<c:forEach var="student" items="${ students }">

...

</c:forEach>

이름이 "김"으로 시작하는 학생 목록이 출력된다.

Graphical user interface, table

Description automatically generated

서버에서 출력된 내용이 웹브라우저 창에 그려진다.

웹브라우저창의 URL은 방금 요청한 URL 이다.

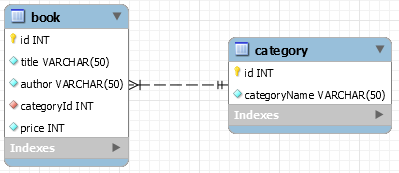
GET 방식 요청이므로 request parameter가 URL의 query string이 되었다.

## 실습

### book3 데이터베이스

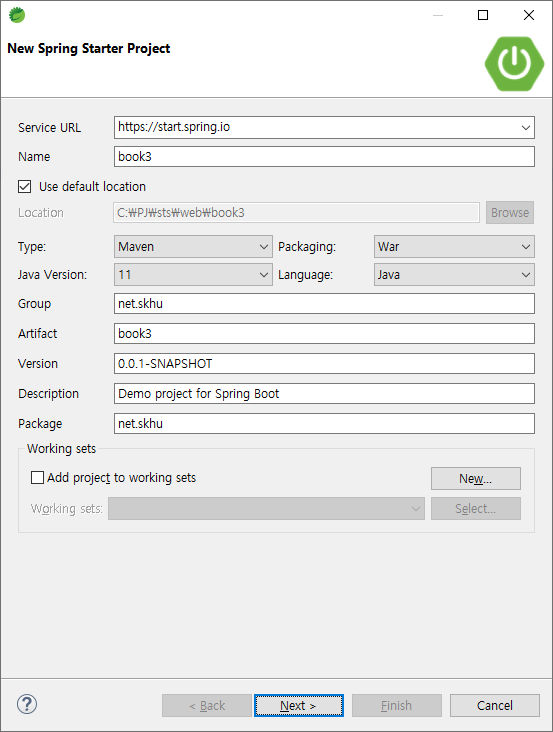
book3.sql 파일을 import 하자.

이전 강의노트 참고 (01 MySQL 설치)



book 테이블의 categoryId 필드가 외래키이다.

### book3 프로젝트를 만들자.



메뉴: File - New - Spring Starter Project

mybatis1 프로젝트 생성 강의노트 참고 (02 데이터목록조회)

### application.properties

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/book3?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

데이터베이스 이름: book3

### book 테이블 조회

(1) 전체 목록 조회를 구현하자.

http://localhost:8088/book/list

(2) 제목으로 조회를 구현하자.

http://localhost:8088/book/list1