

## Method SelectAllStudents

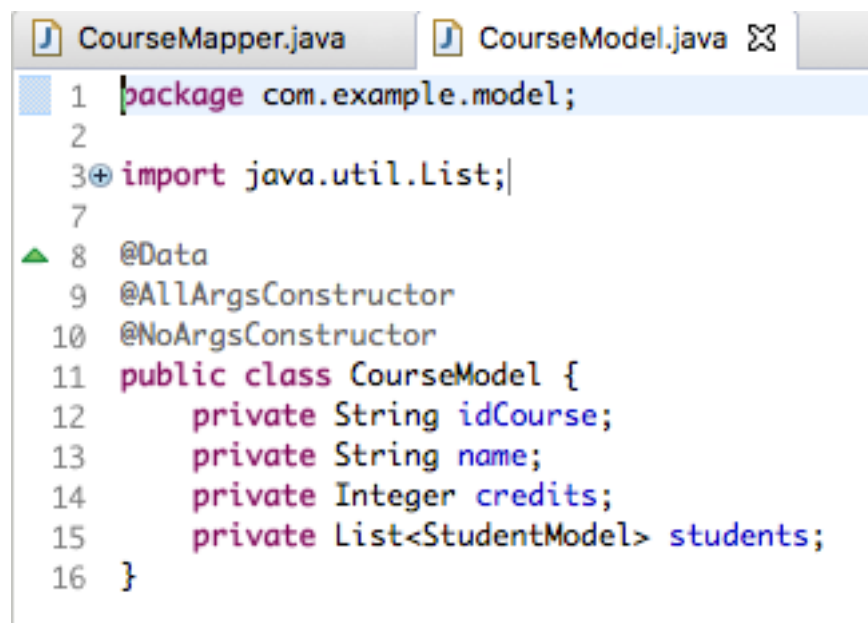
```
@Select("select npm, name, gpa from student")
@Results(value = { @Result(property = "npm", column = "npm"), @Result(property = "name", column = "name"),
    @Result(property = "gpa", column = "gpa"),
    @Result(property = "courses", column = "npm", javaType = List.class, many = @Many(select = "selectCourses")) })
List<StudentModel> selectAllStudents();
```

Saya menambahkan line @Result yang sama seperti pada method view, dimana nantinya tiap variable property akan diisi pada class studentModel. Untuk mendapatkan list mata kuliah yang diambil, dipanggil dengan method selectCourses.

```
<h3>Kuliah yang diambil</h3>
<ul th:each="course, iterationStatus: ${student.courses}">
    <li th:text="${course.name} + '-' + ${course.credits} + ' sks'">
        Nama kuliah-X SKS</li>
</ul>
```

Kemudian menambahkan line ini pada view.html, untuk me-loop list yang berisi course yang diambil.

## Method View pada Course



```
CourseMapper.java CourseModel.java
1 package com.example.model;
2
3 import java.util.List;
4
5
6
7
8 @Data
9 @AllArgsConstructor
10 @NoArgsConstructor
11 public class CourseModel {
12     private String idCourse;
13     private String name;
14     private Integer credits;
15     private List<StudentModel> students;
16 }
```

Pertama saya membuat sebuah model course, yang memiliki idCourse, name, credits, dan tentunya mahasiswa yang mengambil.

```
CourseMapper.java CourseModel.java CourseService.java
1 package com.example.service;
2
3 import com.example.model.CourseModel;
4
5 public interface CourseService {
6     CourseModel selectCourse(String id);
7 }
8
```

Kemudian saya membuat *interface* CourseService yang hanya berisi method selectSource yang menerima parameter id.

```
CourseMapper.java
1 package com.example;
2
3 import java.util.List;
4
5 @Mapper
6 public interface CourseMapper {
7
8     @Select("select id_course, name, credits from course where id_course = #{id_course}")
9     @Results(value = { @Result(property = "idCourse", column = "id_course"), @Result(property = "name", column = "name"),
10         @Result(property = "credits", column = "credits"),
11         @Result(property = "students", column = "id_course", javaType = List.class, many = @Many(select = "selectStudents")) })
12     CourseModel selectCourse(@Param("id_course") String id);
13
14     @Select("select student.npm, name, gpa " + "from student join studentcourse "
15         + "on student.npm = studentcourse.npm " + "where id_course = #{id_course}")
16     List<StudentModel> selectStudents(@Param("id_course") String id);
17 }
18
```

Selanjutnya terdapat courseMapper, di sana terdapat query yang akan dijalankan. Sama seperti method selectAllStudents, hasil query akan dipetakan ke CourseModel. Untuk mendapatkan mahasiswa yang mengambil mata kuliah yang dipilih, dilakukan query pada method selectStudents. Query bisa dilihat pada *screenshot* di atas.

```

CourseMapper.java CourseServiceDatabase.java
1 package com.example.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
10
11 @Slf4j
12 @Service
13 public class CourseServiceDatabase implements CourseService {
14     @Autowired
15     private CourseMapper courseMapper;
16
17     @Override
18     public CourseModel selectCourse(String id) {
19
20         log.info("select student with id {}", id);
21         return courseMapper.selectCourse(id);
22     }
23
24 }

```

Kemudian saya membuat courseServiceDatabase yang mengimplementasi CourseService. Terdapat method selectCourse milik courseMapper.

```

CourseController.java
1 package com.example.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
11
12 @Controller
13 public class CourseController {
14     @Autowired
15     CourseService courseDAO;
16
17     @RequestMapping("/course/view/{id}")
18     public String viewCoursePath(Model model, @PathVariable(value = "id") String id) {
19         CourseModel course = courseDAO.selectCourse(id);
20
21         if (course != null) {
22             model.addAttribute("course", course);
23             return "viewcourse";
24         } else {
25             model.addAttribute("id", id);
26             return "course-not-found";
27         }
28     }
29 }

```

Pada CourseController, path '/course/view/{id}' akan mengembalikan halaman yang sesuai dengan id yang dipanggil. Jika course ada, maka halaman viewcourse.html akan muncul. Jika tidak ada maka akan muncul halaman error.

```

StudentMapper.j CourseMapper.ja viewcourse.html not-found.ht
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4 <title>View Course by ID</title>
5 </head>
6 <body>
7 <h3 th:text="'ID course = ' + ${course.idCourse}">Course NPM</h3>
8 <h3 th:text="'Name = ' + ${course.name}">Course Name</h3>
9 <h3 th:text="'SKS = ' + ${course.credits}">Course Credits</h3>
10
11 <h3>Mahasiswa yang mengambil</h3>
12 <ul th:each="student, iterationStatus: ${course.students}">
13 <li th:text="${student.npm} + ' - ' + ${student.name}">
14 NPM-Nama</li>
15 </ul>
16 </body>
17 </html>

```

Pada halaman viewcourse, sama seperti viewall.html Bedanya adalah sudah disesuaikan karena akan mengembalikan courseModel dan akan mencetak list mahasiswa yang mengambil mata kuliah.

#### KESIMPULAN:

Pada minggu ini saya mempelajari bagaimana menyambungkan relasi many-to-many pada database. Untuk melakukan join pada kedua relasi, dibutuhkan method sendiri yang nantinya berupa suatu list yang memiliki loop sendiri.