

WRITE UP

Tutorial 5 - Menggunakan Database serta Relasi Database dalam Project Spring Boot

Pada tutorial kali ini, kita belajar cara menggunakan database pada project spring. Mirip dengan tutorial 4, tapi bedanya disini kita belajar cara membuat project dengan 2 collections/table berbeda. Kedua table ini nantinya akan saling berinteraksi dan disinilah kita belajar beberapa syntax yang berhubungan dengan ini.

Persiapan

Sebelum memulai, seperti objectives dari tutorial ini, kita akan buat dulu table baru yang berisi mata kuliah yang ada.

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.2343 seconds.)

```
ALTER TABLE `course` ADD PRIMARY KEY(`id_course`);
```

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
<input type="checkbox"/> 1	id_course	varchar(20)	utf8_general_ci		No	None		Change
<input type="checkbox"/> 2	name	varchar(45)	utf8_general_ci		Yes	NULL		Change
<input type="checkbox"/> 3	credits	double			Yes	NULL		Change

☐ Check all With selected: Browse Change Drop Primary Un

Remove from central columns

Print view Propose table structure Track table Move columns Improve

Add 1 column(s) after credits Go

[+ Indexes](#)

Kemudian, table tersebut akan di populate dengan data-data dummy, sql ada di pdf tutorial5.

				id_course	name	credits
<input type="checkbox"/>		Edit		Copy		Delete
				CSC123	PSP	4
<input type="checkbox"/>		Edit		Copy		Delete
				CSC124	SDA	3
<input type="checkbox"/>		Edit		Copy		Delete
				CSC125	DDP 1	4
<input type="checkbox"/>		Edit		Copy		Delete
				CSC126	MPKT	6

Seperti yang kita pelajari saat kuliah basis data, kita perlu buat relasi antara Student dengan Course. Relasi ini bisa digambarkan dengan 1 tambahan table lagi untuk dibuat.

	#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
<input type="checkbox"/>	1	npm	varchar(20)	utf8_general_ci		No	None		Change
<input type="checkbox"/>	2	id_course	varchar(30)	utf8_general_ci		No	None		Change

☐ Check all
 With selected:
 Browse
 Change
 Drop
 Primary
 Unique

Kemudian kita populasikan berdasarkan data student yang ada
Data student

				npm	name	gpa
<input type="checkbox"/>		Edit		Copy		Delete
				12345678	Martha Bucks	3.65
<input type="checkbox"/>		Edit		Copy		Delete
				1506689345	Bella Nadhifah	4

Populasi data StudentCourse.

npm	id_course
1506689345	CSC123
1506689345	CSC125
12345678	CSC123

Table diatas menggambarkan bahwa npm 1506689345 (bernama Bella) mengambil kuliah dengan id CSC123 dan CSC125.

Pengerjaan Tutorial

Kita akan lanjutkan pengerjaan dari tutorial 4 kemarin.

Tutorial dimulai dengan membuat model baru bernama CourseModel untuk menampung data-data course. Prinsipnya sama seperti StudentModel

```

CourseModel.java StudentControll Student
1 package com.example.model;
2
3 import java.util.List;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
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 }
17

```

Karena definisinya setiap Student bisa ambil 0 atau 1 atau lebih mata kuliah, maka di struktur data student kita perlu diupdate seperti ini

```

0 @AllArgsConstructor
1 @NoArgsConstructor
2 public class StudentModel
3 {
4     private String npm;
5     private String name;
6     private double gpa;
7     private List<CourseModel> courses;
8
9 }
10

```

Lalu seperti kata tutorial5, kita harus membenarkan error yang muncul dengan update constructor StudentModel yang sebelumnya dipakai di controller

```

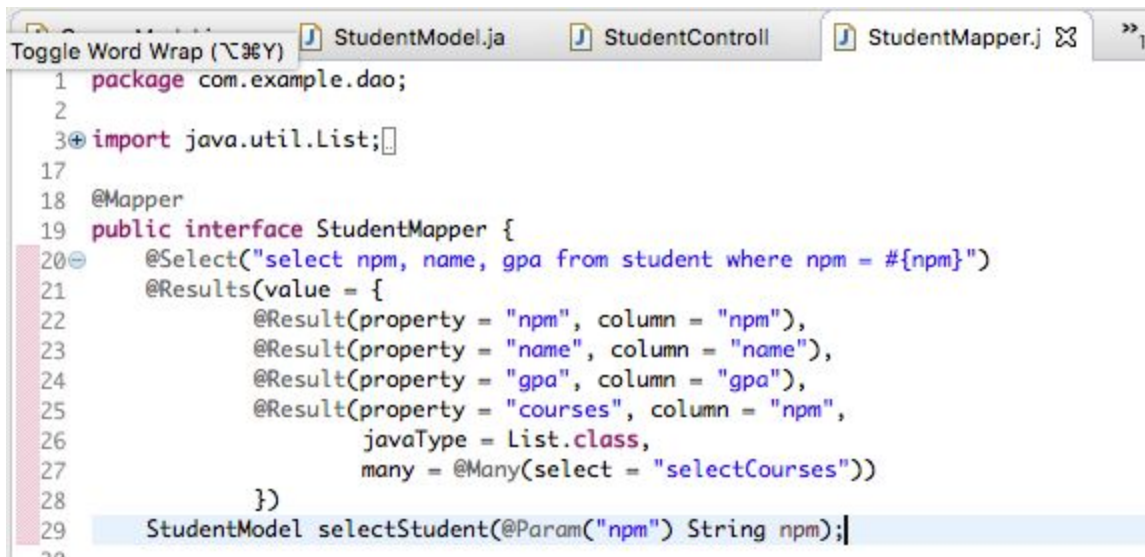
CourseModel.java StudentModel.java StudentControll view.html
54 @RequestMapping("/student/add/submit")
55 public String addSubmit (
56     @RequestParam(value = "npm", required = false) String npm,
57     @RequestParam(value = "name", required = false) String name,
58     @RequestParam(value = "gpa", required = false) double gpa)
59 {
60     StudentModel student = new StudentModel (npm, name, gpa, null);
61     studentDAO.addStudent (student);
62
63     return "success-add";
64 }
--

```

Kita akan buat method untuk query mata kuliah di StudentMapper seperti ini.

```
@Select("select course.id_course, name, credits " + "from studentcourse join course "
+ "on studentcourse.id_course = course.id_course " + "where studentcourse.npm = #{npm}")
List<CourseModel> selectCourses(@Param("npm") String npm);
```

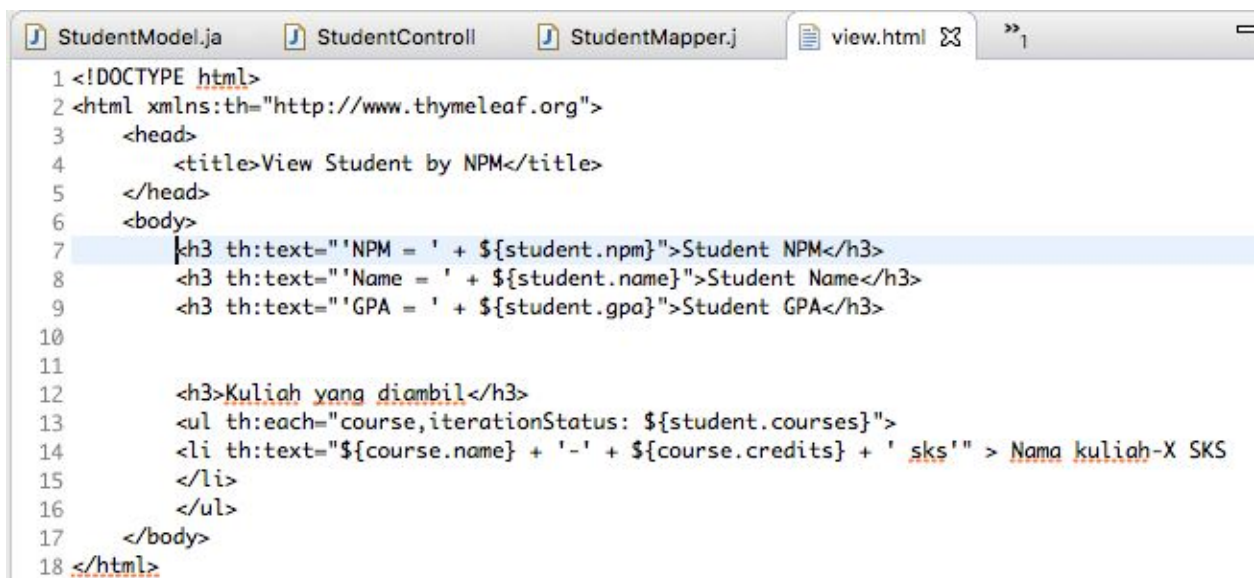
Selanjutnya, kita perlu mengambil list of course yang diambil oleh mahasiswa, dan memasukkannya ke struktur data list of course yang sebelumnya sudah kita buat. Ini perlu update query yang sebelumnya kita buat saat select student, diupdate menjadi seperti ini.



```
1 package com.example.dao;
2
3 import java.util.List;
4
17 @Mapper
18 public interface StudentMapper {
19     @Select("select npm, name, gpa from student where npm = #{npm}")
20     @Results(value = {
21         @Result(property = "npm", column = "npm"),
22         @Result(property = "name", column = "name"),
23         @Result(property = "gpa", column = "gpa"),
24         @Result(property = "courses", column = "npm",
25             javaType = List.class,
26             many = @Many(select = "selectCourses"))
27     })
28     StudentModel selectStudent(@Param("npm") String npm);
29 }
```

Perhatikan bahwa pada query tersebut otomatis akan memasukkan hasilnya kedalam courses (seperti yang didefinisikan) dan proses selectnya menggunakan selectCourses yang sebelumnya kita buat.

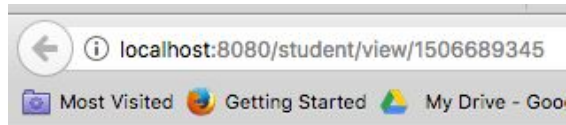
Untuk menampilkannya, kita perlu menambahkan pada view.html baris code berikut untuk space dari list mata kuliah tersebut.



```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3     <head>
4         <title>View Student by NPM</title>
5     </head>
6     <body>
7         <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
8         <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
9         <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
10
11         <h3>Kuliah yang diambil</h3>
12         <ul th:each="course, iterationStatus: ${student.courses}">
13             <li th:text="${course.name} + ' - ' + ${course.credits} + ' sks'" > Nama kuliah-X SKS
14         </li>
15     </ul>
16 </body>
17 </html>
```


Perhatikan bahwa kode tersebut juga menggunakan each (seperti forloop) untuk menampilkan semua dalam list course mahasiswa tersebut.

Jalankan program, buka <localhost:8080/student/view/1506689345> dan <localhost:8080/student/view/12345678> (data student yang ada di database).



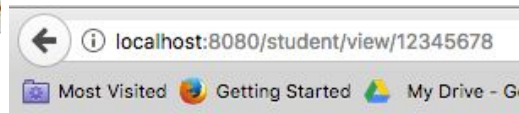
NPM = 1506689345

Name = Bella Nadhifah

GPA = 4.0

Kuliah yang diambil

- PSP-4 sks
- DDP 1-4 sks



NPM = 12345678

Name = Martha Bucks

GPA = 3.65

Kuliah yang diambil

- PSP-4 sks

Latihan

1. Ubah method `selectAllStudents` pada kelas `StudentMapper` agar halaman `viewall` menampilkan semua student beserta daftar kuliah yang diambil.

Pada dasarnya, student sudah punya struktur data yang mampu menyimpan course yang diambilnya. Tapi kita perlu menambahkan query seperti tadi, untuk bagaimana caranya, saat `select all students`, data-data courses yang diambil student tersebut tersimpan kedalam list. Caranya mirip dengan yang `selectStudent` tadi.

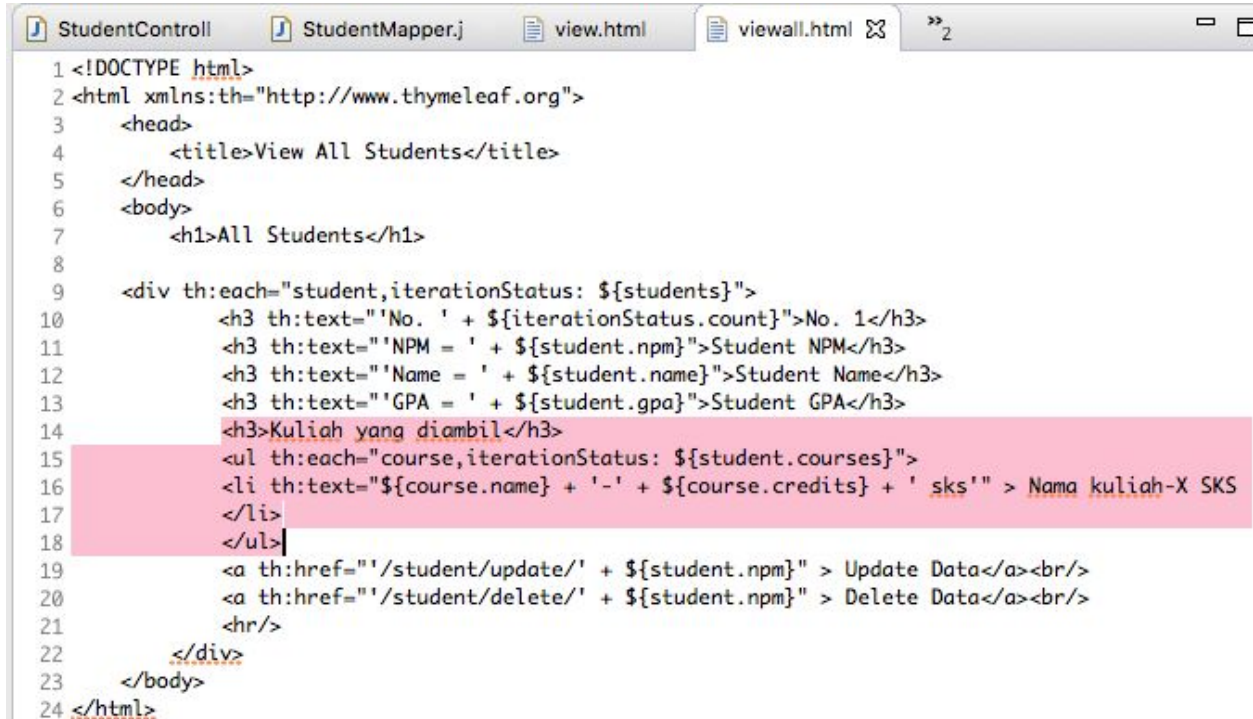
```
@Select("select npm, name, gpa from student")
@Results(value = {
    @Result(property = "npm", column = "npm"),
    @Result(property = "name", column = "name"),
    @Result(property = "gpa", column = "gpa"),
    @Result(property = "courses", column = "npm",
        javaType = List.class,
       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

Kita akan melakukan tambahan `@Results` di method `selectAllStudents()`. Pada baris `@Result(property="npm", column = "npm")`, isian `property` menunjukkan nama variable pada kelas `CourseModel`, dan pada isian `column` menunjukkan nama kolom di tabel yang tadi kita buat. Kemudian pada `javaType = List.class` ditunjukkan bahwa nanti hasilnya akan disimpan

kedalam List (ex: ArrayList). `@Many(select = "selectCourses")` menunjukkan nama method di `studentMapper` yang akan digunakan untuk mendapatkan coursanya.

Dengan demikian, setiap `Student` yang kita peroleh dari `selectAllStudents()` akan memiliki list of `courses` yang sudah di fetch dari query tersebut (prinsipnya mirip dengan yang `selectStudent()` sebelumnya).

Selanjutnya, kita hanya perlu menampilkannya di `viewall.html` dengan menambah beberapa baris kode seperti sebelumnya di `view.html`.

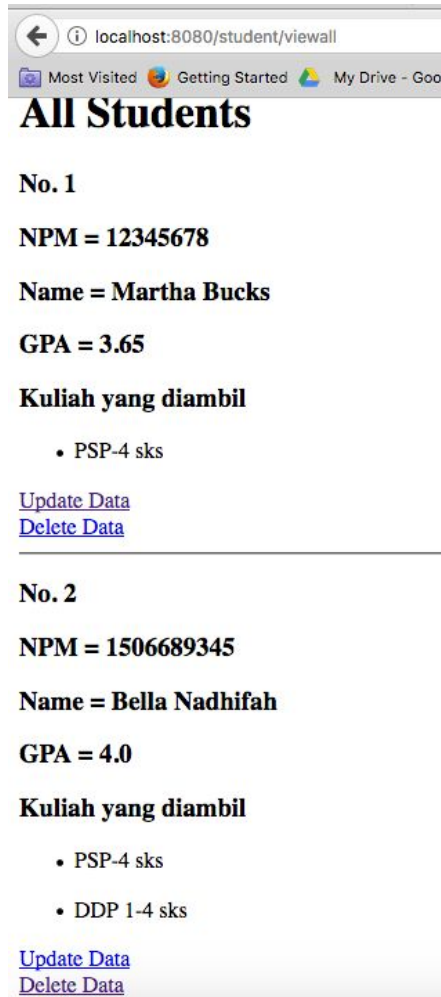


```

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <title>View All Students</title>
5   </head>
6   <body>
7     <h1>All Students</h1>
8
9     <div th:each="student, iterationStatus: ${students}">
10       <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
11       <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
12       <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
13       <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
14       <h3>Kuliah yang diambil</h3>
15       <ul th:each="course, iterationStatus: ${student.courses}">
16         <li th:text="${course.name} + '-' + ${course.credits} + ' sks'" > Nama kuliah-X SKS
17       </li>
18     </ul>
19     <a th:href="/student/update/" + ${student.npm}" > Update Data</a><br/>
20     <a th:href="/student/delete/" + ${student.npm}" > Delete Data</a><br/>
21   </div>
22 </body>
23 </html>

```

Kita test sekarang dengan buka `/student/viewall`, dan bisa!

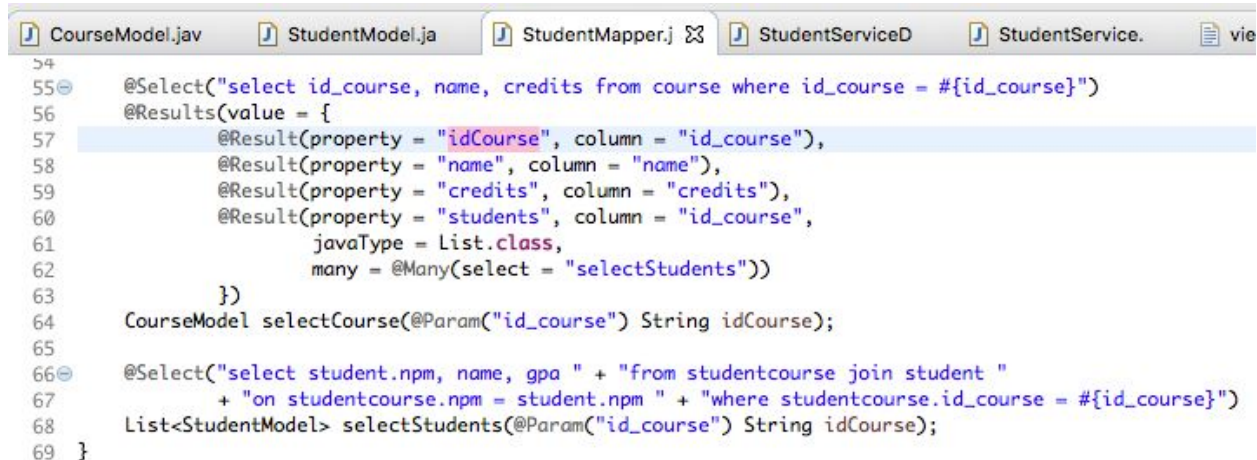


2. Buatlah view pada halaman `http://localhost:8080/course/view/{id}` untuk Course sehingga dapat menampilkan data course beserta Student yang mengambil.

Pertama, kita perlu melakukan reverse action saat kita ingin tampilkan list course yang diambil oleh mahasiswa.

Sebelumnya kita harus **memastikan** dulu terdapat struktur data **List<StudentModel> students**; di CourseModel untuk menampung data student yang mengambil mata kuliah tersebut.

Setelah itu, kita buat dulu query selectStudents dengan query join di table studentCourse, mirip dengan selectCourses cuman tinggal kita reverse saja parameter dan wherenya



```
54
55 @Select("select id_course, name, credits from course where id_course = #{id_course}")
56 @Results(value = {
57     @Result(property = "idCourse", column = "id_course"),
58     @Result(property = "name", column = "name"),
59     @Result(property = "credits", column = "credits"),
60     @Result(property = "students", column = "id_course",
61         javaType = List.class,
62         many = @Many(select = "selectStudents"))
63 })
64 CourseModel selectCourse(@Param("id_course") String idCourse);
65
66 @Select("select student.npm, name, gpa " + "from studentcourse join student "
67     + "on studentcourse.npm = student.npm " + "where studentcourse.id_course = #{id_course}")
68 List<StudentModel> selectStudents(@Param("id_course") String idCourse);
69 }
```

Jangan lupa seperti screenshot diatas, kita buat saat selectCourse kita melakukan assignment list student yang dimiliki course tersebut ke dalam list students di data model CourseModel.

Perhatikan bahwa pada method selectStudents (yang bawah), adalah untuk mendapatkan course tertentu berdasarkan idcourse yang sudah di join dengan setiap student yang mengambil course tersebut. Dengan demikian, kita bisa mendapatkan data-data student yang mengambil course tertentu berdasarkan idcourse yang dikirimkan.

Method selectCourse (yang atas) akan mengembalikan CourseModel. Prinsipnya sama dengan yang selectStudent sebelumnya, kita mapping property dan column yang dibutuhkan, kemudian disimpan ke dalam List (pada variable javaType), dan menggunakan method selectStudents untuk mendapatkan data-data student nya.

Kita buat service method baru untuk select course by idCourse. Jangan lupa menambahkannya di interfacenya dan di class yang implnya.


```
public interface StudentService
{
    StudentModel selectStudent (String npm);

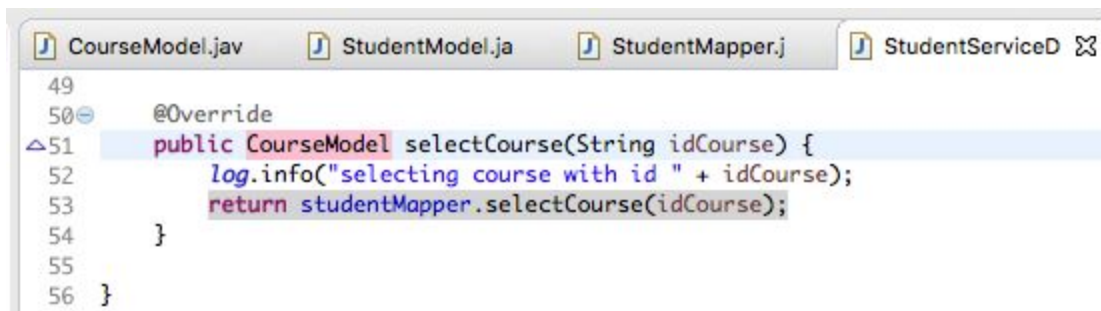
    List<StudentModel> selectAllStudents ();

    void addStudent (StudentModel student);

    void deleteStudent (String npm);

    void updateStudent (StudentModel student);

    CourseModel selectCourse(String idCourse);
}
```



Kita buat method controller baru untuk view course di controller.

```
@RequestMapping("/course/view/{id_course}")
public String viewCoursePath (Model model,
    @PathVariable(value = "id_course") String idCourse)
{
    CourseModel course = studentDAO.selectCourse(idCourse);

    if (course != null) {
        model.addAttribute("course", course);
        return "viewcourse";
    } else {
        model.addAttribute("idCourse", idCourse);
        return "not-foundcourse";
    }
}
```

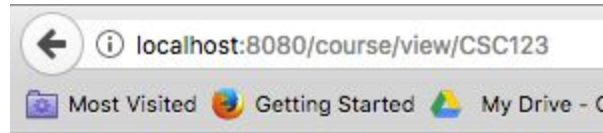
Last but not least, kita perlu buat html nya untuk menampilkan data students yang sudah kita retrieve tadi.

```
viewcourse.html CourseModel.jav StudentModel.ja StudentControll
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <title>View Student by NPM</title>
5   </head>
6   <body>
7     <h3 th:text="'IDCourse = ' + ${course.idCourse}">IDCourse</h3>
8     <h3 th:text="'Name = ' + ${course.name}">Course Name</h3>
9     <h3 th:text="'Credits = ' + ${course.credits}">Course Credits</h3>
10
11
12     <h3>Kuliah yang diambil</h3>
13     <ul th:each="student, iterationStatus: ${course.students}">
14       <li th:text="${student.npm} + ' - ' + ${student.name}" > NPM STUDENT
15     </li>
16   </ul>
17 </body>
18 </html>
```

BONUS!! Jangan lupa untuk membuat notfound in case ternyata coursanya tidak ditemukan.

```
viewcourse.html not-foundcourse StudentControll StudentS
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3   <head>
4     <title>Course not found</title>
5   </head>
6   <body>
7     <h1>Course not found</h1>
8     <h3 th:text="'IDCourse = ' + ${idCourse}">Student NPM</h3>
9   </body>
10 </html>
```

Kita akan lakukan testing.



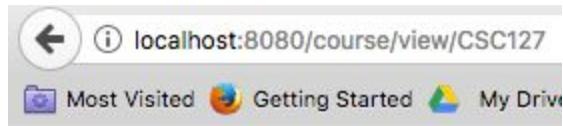
IDCourse = CSC123

Name = PSP

Credits = 4

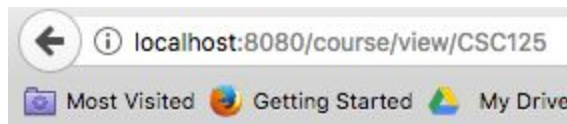
Kuliah yang diambil

- 12345678 - Martha Bucks
- 1506689345 - Bella Nadhifah



Course not found

IDCourse = CSC127



IDCourse = CSC125

Name = DDP 1

Credits = 4

Kuliah yang diambil

- 1506689345 - Bella Nadhifah

Dan berhasil!

What I have learned

Pada tutorial kali ini, saya belajar tentang tata cara membuat service yang ada 2 table. Disini saya belajar penggunaan anotasi `@Results` dan `@Result` untuk melakukan penggabungan 2 query yang didapat. Dengan demikian bertambah kembali ilmu penggunaan spring yang bisa memanfaatkan lebih dari 1 table dan 1 model sekaligus.