Tutorial 5 - Menggunakan Database serta Relasi Database dalam Project

Fitur *Select All Students*

1. Pada *StudentMapper,* saya melakukan modifikasi pada *method* selectAllStudents menjadi seperti berikut:

```
@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 ();
```

Gambar 1. *Method* selectAllStudents yang telah dimodifikasi.

Modifikasi ini berguna untuk meng-*assign value* hasil dari *query*, terutama *list* dari *course* yang diambil mahasiswa tersebut, ke *object StudentModel* yang ada pada di dalam *List.*

2. Pada *viewall*, saya melakukan modifikasi berupa tambahan untuk menampilkan *course* yang diambil oleh mahasiswa. Berikut *viewall* yang sudah dimodifikasi:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View All Students</title>
    </head>
    <body>
        <h1>All Students</h1>

        <div th:each="student,iterationStatus: ${students}">
            <h3 th:text="'No. ' + ${iterationStatus.count}">No. 1</h3>
            <h3 th:text="'NPM = ' + ${student.npm}">Student NPM</h3>
            <h3 th:text="'Name = ' + ${student.name}">Student Name</h3>
            <h3 th:text="'GPA = ' + ${student.gpa}">Student GPA</h3>
            <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>
            <a th:href="'/student/update/' + ${student.npm}" > Update Data</a><br/>
            <a th:href="'/student/delete/' + ${student.npm}" > Delete Data</a><br/>
            <hr/>
        </div>
    </body>
</html>
```

Gambar 2. *viewall* yang telah dimodifikasi.

Perubahan yang dicakup adalah sebuah *list* yang mengiterasi dan menampilkan semua *course* yang di-*enroll* mahasiswa yang ada di dalam *List.*

Menampilkan *Course* dan Mahasiswa yang Mengambilnya

1. Pertama, saya membuat dua buah *method* pada *Mapper*, yaitu selectStudents dan selectCourse sebagai berikut:

```
@Select("select student.npm, student.name, student.gpa " +
        "from studentcourse join student " +
        "on studentcourse.npm = student.npm " +
        "where studentcourse.id_course = #{idCourse}")
List<StudentModel> selectStudents (@Param("idCourse") String idCourse);
```

Gambar 3. *Method selectStudents*

```
@Select("SELECT course.id_course, course.name, course.credits "
        + "FROM course WHERE course.id_course = #{idCourse}")
@Results(value = {
            @Result(property="idCourse", column="id_course"),
            @Result(property="name", column="name"),
            @Result(property="credits", column="credits"),
            @Result(property="students", column="id_course",
                    javaType = List.class,
                    many=@Many(select="selectStudents"))
        })
CourseModel selectCourse (@Param("idCourse") String idCourse);
```

Gambar 4. *Method selectCourse*

Method selectStudents mengambil seluruh *student* yang mengambil mata kuliah sesuai dengan idCourse pada *parameter*.

Method selectCourse memilih suatu *course* dengan idCourse sesuai pada *parameter* dan lalu meng-*assign* atribut dari *object CourseModel* kembalian dengan hasilnya.

2. Kemudian, saya menamahkan *method-method* tersebut pada *Service* seperti gambar berikut:

```
List<StudentModel> selectStudents (@Param("idCourse") String idCourse);

CourseModel selectCourse (@Param("idCourse") String idCourse);
```

Gambar 5. *Method* pada *StudentService*

```
@Override
public List<StudentModel> selectStudents (@Param("idCourse") String idCourse) {
    return studentMapper.selectStudents(idCourse);
};

@Override
public CourseModel selectCourse (@Param("idCourse") String idCourse) {
    return studentMapper.selectCourse(idCourse);
}
```

Gambar 6. *Method* pada *StudentServiceDatabase*

3. Kemudian pada *StudentController*, saya menambahkan sebuah *method* dengan nama *viewCourse* sebagai berikut:

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

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

Gambar 7. *Method viewCourse* pada *StudentController*

Method ini menerima *parameter* sebuah *Model* dan sebuah String idCourse yang berasal dari *PathVariable*. *Method* pertama melakukan pengecekan terhadap keberadaan *course* dengan idCourse tersebut dengan memanfaatkan *selectCourse*. Prinsip kerjanya sama, yaitu mengecek nilai dari *object CourseModel* yang dikembalikan. Apabila *null,* berarti *course* tidak ada, sehingga *user* akan di-*redirect* ke

page error berupa *not-found-course*. Apabila tidak *null,* akan ditampilkan *view course* tersebut.

4. Untuk menampilkan fitur-fitur tersebut, saya menambahkan dua buah *view* sebagai berikut:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>View Course by Course ID</title>
    </head>
    <body>
        <h3 th:text="'ID = ' + ${course.idCourse}">Course ID</h3>
        <h3 th:text="'Nama = ' + ${course.name}">Course Name</h3>
        <h3 th:text="'SKS = ' + ${course.credits}">Course Credits</h3>
        <h3>Mahasiswa yang mengambil</h3>
            <ul th:each="student,iterationStatus: ${course.students}">
               <li th:text="${student.npm} + '-' + ${student.name} + ' sks'" >
                   NPM - Nama Mahasiswa
                </li>
            </ul>
    </body>
</html>
```

Gambar 8. *View view-course*.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Course not found</title>
    </head>
    <body>
        <h1>Course not found</h1>
        <h3 th:text="'Course ID = ' + ${idCourse}">Course ID</h3>
    </body>
</html>
```

Gambar 9. *View not-found-course*.

Kedua *view* ini memiliki prinsip yang sama seperti dua buah *view* yang mirip, yaitu *view* dan *not-found*. Kedua *view* ini hanya mendapatkan sejumlah modifikasi kecil untuk menyesuaikan dan *case* yang dibutuhkan.

Refleksi *Tutorial*

Tutorial ini mengeksplor lebih dalam mengenai penggunaan *database* dalam *Spring*, khususnya mengenai hubungan antara dua buah *entity* dan *class* yang bersangkutan seperti pada contohnya pada kasus ini adalah *studentcourse* yang merupakan sebuah relasi penghubung *entity student* dengan *course*. Konten dari *tutorial* ini membantu saya dalam memahami konsep MVC serta sejumlah *best practice* dalam penggunaan MVC, khususnya mengenai *layering*.

Selain itu, *tutorial* ini juga memperkenalkan dan mengajari saya mengenai *library mybatis* beserta *syntax-syntax* baru, khususnya penggunaan anotasi *Result* dalam melakukan *assigning* dari *value* hasil *query* ke-dalam atribut-atribut dalam suatu *object* kembalian.