Pada tutorial ini saya mempelajari bagaimana caranya memetakan hasil dari query ke dalam suatu class model menggunakan anotasi @Results dan @Result. Anotasi @Result berfungsi untuk mengatur data yang diinginkan untuk masuk ke dalam suatu class model. Karena bisa saja nama atribut dalam suatu class model tidak sama dengan nama kolom pada table database. Serta mempermudah dalam memetakan atribut yang membutuhkan query berbeda dengan menggunakan anotasi @One ataupun @Many. Sendangkan @Results sendiri digunakan jika hasil pemetaan yang diinginkan terdapat lebih dari satu.

```
@Select("select npm, name, gpa from student where npm = #{npm}")
@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"))
})
StudentModel selectStudent(@Param("npm") String npm);

@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();
```

1. Saya mengubah method selectAllStudents pada StudentMapper dengan memetakan hasil sesuai yang dibutuhkan. Karena terdapat atribut courses pada StudentModel yang nantinya akan menyimpan informasi kelas apa saja yang diikuti oleh mahasiswa. Sehingga perlu dilakukan pemetaan menggunakan anotasi @Result dan @Many karena atribut courses bisa menyimpan 1 atau lebih kelas.

```
@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();
```

Setelah itu saya mengubah viewall html dengan melakukan iterasi pada atribut courses untuk mendapatkan informasi kelas yang diikuti mahasiswa.

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

2. Pertama tama saya membuat interface CourseMapper yang nantinya akan melakukan query SELECT untuk mendapatkan informasi kelas yang dicari dengan melakukan pemetaan menggunakan @Results @Result dan @Many. Terdapat dua method yang saya buat pada kelas CourseMapper, yaitu selectCourse untuk mendapatkan kelas yang dicari serta selectStudents untuk mengetahui siapa saja mahasiswa yang mengikuti kelas yang dicari.

```
@Mapper
public interface CourseMapper
{
    @Select("SELECT id_course, name, credits FROM COURSE WHERE id_course = #{id_course}")
    @Results(value = {
            @Result(property = "id_course", 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("id_course") String id_course);

    @Select("SELECT STUDENT.npm, STUDENT.name " +
            "FROM STUDENT JOIN STUDENTCOURSE " +
            "ON STUDENT.npm = STUDENTCOURSE.npm " +
            "WHERE STUDENTCOURSE.id_course = #{id_course}")
    List<StudentModel> selectStudents(@Param("id_course") String id_course);
```

Setelah itu saya membuat interface CourseService yang memiliki method selectCourse yang akan diimplementasikan oleh CourseServiceDatabase. CourseServiceDatabase akan meng-override method selectCourse dan

mengimplementasikannya dengan menggunakan atribut kelas CourseMapper dan memanggil method selectCourse pada CourseMapper. Method ini memiliki tipe kembalian CourseModel.

```
public interface CourseService
{
    CourseModel selectCourse(String id_course);
}
```

```
@Service
public class CourseServiceDatabase implements CourseService
{
    @Autowired
    private CourseMapper courseMapper;

    @Override
    public CourseModel selectCourse(String id_course)
    {
        return courseMapper.selectCourse(id_course);
    }
}
```

Setelah itu saya membuat class CourseController yang memiliki method view yang dijalankan ketika menerima mapping kehalaman "/course/view/{id_course}".

```
@RequestMapping("/course/view/{id_course}")
public String view(Model model, @PathVariable(value = "id_course") String id_course)
{
    model.addAttribute( attributeName: "course", courseDAO.selectCourse(id_course));
    return "view-course";
}
```