Writeup Tutorial 5

Kenny Reida Dharmawan
1506757472, APAP - C.

Hal Baru yang Dipelajari

Dalam tutorial kali ini, saya mempelajari bagaimana cara melakukan query kompleks dimana saya perlu melakukan operasi join untuk mendapatkan apa yang saya perlukan, saya juga belajar bahwa pada DAO di spring, saya bisa melakukan custom mapping dari data yang saya dapatkan dari query, memberikan kemampuan untuk mengabungkan beberapa query dalam satu operasi, dan mengganti cara menampilkan data yang berbeda apa yang dikembalikan oleh query.

Penjelasan Implementasi

Implementasi selectAllStudents

Dalam melakukan implementasi selectAllStudents, pertama saya menambahkan dekorator @Results pada mapper selectAllStudents di dao.StudentMapper, kode yang saya tambahkan terlihat seperti pada gambar dibawah ini.

```
32  @Select("select npm, name, gpa from student")
33  @Results(value = {
34    @Result(property="npm", column="npm"),
35    @Result(property="name", column="name"),
36    @Result(property="gpa", column="gpa"),
37    @Result(property="courses", column="npm",
38      javaType = List.class,
39      many=@Many(select="selectCourses"))
40  })
41  List<StudentModel> selectAllStudents ();
42
```

Gambar 1.1.
Method mapper selectAllStudents() yang terletak di com.example.dao.StudentMapper

Method diatas akan melakukan query select yang akan mengambil data dari kolom npm, name, dan gpa dari tabel student, setelah berhasil mendapatkan data-data yang ada, maka setiap data tersebut akan dimapping ulang oleh dekorator @Results dengan

value yang sesuai dengan attribut value pada parameternya, dimana value tersebut adalah objek yang berisi setiap kolom yang akan dikembalikan, attribut property pada parameter dekorator @Result menandakan nama variable yang akan dikembalikan, dan atribut column menandakan nama variabel dari data hasil query. Pada dekorator @Result terakhir, saya menambahkan property courses, dimana isinya adalah hasil query dari method selectCourses, dalam mendapatkan hasil query ini, saya memasukan parameter npm dari data hasil query nya dengan cara menambahkan attribut column yang mendeklarasikan bahwa dalam memanggil method selectCourses, @Result akan meneruskan variabel yang ada di atribut column (disini npm) kedalam parameter pemanggilan method selectCourses. Setelah berhasil melakukan query, dan memapping setiap data yang ada di query sesuai dengan yang dibutuhkan (dan juga melakukan query selectCourses untuk setiap data), maka method ini akan mengembalikan data berupa List yang berisikan data bertipe StudentModel.

Setelah berhasil melakukan query dan mendapatkan data, selanjutnya saya melakukan sedikit perubahan pada template viewall, dimana kodenya terlihat seperti gambar dibawah ini.

```
<!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 1.2.
Template viewall.html

Pada kode diatas, terlihat saya menambahkan tulisan Kuliah yang diambil, beserta sebuah unordered list yang berguna untuk menampilkan daftar nama mata kuliah yang diambil dan jumlah kredit yang diambil dalam bentuk baris tidak teratur (unordered list).

Implementasi view Course

Untuk implementasi ini, saya mulai dengan melakukan implementasi method selectStudents() pada class StudentMapper, method ini berguna untuk mengambil daftar mahasiswa yang mengambil suatu mata kuliah, implementasinya dapat dilihat pada gambar dibawah ini.

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

Gambar 2.1.
Implementasi method selectStudents()

Dari gambar diatas, dapat dilihat saya melakukan sebuah query select ke database yang mengambil data npm, name, dan gpa seorang student dari hasil join table studentcourse dan student, dimana saya meminta data jika dan hanya jika id_course mata kuliah tersebut sesuai dengan id_course yang diberikan parameter.

Setelah melakukan implementasi method diatas, saya menambahkan method mapper selectCourse, yang berguna untuk mendapatkan data suatu mata kuliah berdasarkan id_course mata kuliah tersebut, kode dapat dilihat pada gambar dibawah ini.

```
  @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);
```

Gambar 2.2.
Implementasi method selectCourse()

Pada kode diatas, saya melakukan query select untuk mendapatkan semua mata kuliah dan meminta data dari kolom id_course, name, dan credits, dari setiap data

yang saya dapat, data tersebut akan dimapping sesuai dengan aturan yang ada pada setiap dekorator @Result, saya juga menambahkan kolom tambahan students yang berisi data hasil query method selectStudents, hal ini mirip dengan implementasi selectStudent, akan tetapi saya menyesuaikan data yang diminta dan tabel yang dituju dengan data dari view course.

Setelah membuat method selectCourse di DAO StudentMapper, selanjutnya saya menambahkan method selectCourse di service saya, kodenya ada pada gambar dibawah ini.

```
53
54    @Override
55    public CourseModel selectCourse(String id_course) {
56      log.info("select course with id {}", id_course);
57      return studentMapper.selectCourse(id_course);
58    }
59  }
```

Gambar 2.3.
Implementasi service selectCourse()

Service diatas merupakan service yang cukup standar, dimana ketika dipanggil akan melakukan logging dan memanggil sekaligus mengembalikan data yang dikembalikan oleh method selectCourse dari studentMapper.

Setelah membuat service, hal yang saya lakukan selanjutnya adalah membuat Controller dan Template dari view-course, kode serta penjelasannya dapat dilihat dibawah ini.

```
131
132     @RequestMapping("/course/view/{id_course}")
133     public String viewCourse (Model model, @PathVariable(value = "id_course") String id_course)
134     {
135       CourseModel course = studentDAO.selectCourse (id_course);
136
137       if (course != null) {
138         model.addAttribute ("course", course);
139         return "view-course";
140       } else {
141         model.addAttribute ("id_course", id_course);
142         return "not-found-course";
143       }
144     }
145 }
```

Gambar 2.4.
Controller viewCourse()

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <head>
      <title>View Course by ID</title>
  </head>
  <body>
    <h3 th:text="'ID = ' + ${course.id_course}">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}" >
      NPM - Nama Mahasiswa
    </li>
  </ul>
  </body>
</html>
```

Gambar 2.5.
Template view-course.html

```
<!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="'ID = ' + ${id_course}">Course ID</h3>
  </body>
</html>
```

Gambar 2.6.
Template not-found-course.html

Dari Gambar 2.4, terlihat kode controller viewCourse(), controller ini berguna untuk menangkap setiap request yang mengarah ke /course/view/{id_course}, pertama tama setelah mendapatkan request, controller akan mengecek apakah course yang diminta memang ada, jika ada, maka controller akan mengembalikan view dari template view-course dengan data course yang diminta, jika tidak ada maka controller akan mengembalikan view not-found-course dengan data berupa id_course yang tidak ditemukan.

Pada template view-course.html (Gambar 2.5.), saya mendisplay data dari course yang diminta dan juga list mahasiswa yang mengambil mata kuliah ini dengan data npm dan nama mahasiswanya. Ketika mata kuliah yang diminta tidak ada, maka template not-found-course.html (Gambar 2.6.) akan ditampilkan dengan informasi bahwa Course yang diminta tidak ada dan menampilkan id course yang diminta.