

Method Delete

```
@Delete("DELETE FROM student WHERE npm=#{npm}")
void deleteStudent(@Param("npm") String npm);
```

Pertama membuat sebuah method *abstract* pada class interface *StudentMapper*. Gunakan anotasi *@Delete* untuk mengeksekusi query menghapus data dalam database. Parameter method tersebut menggunakan objek dari *student* dan menggunakan *@Param* untuk menangkap data dengan inisial "student".

```
void deleteStudent (String npm);
```

Tambahkan method *abstract deleteStudent* pada class interface *StudentService*. Pada parameter method tersebut menerima sebuah *String npm*.

```
@Override
public void deleteStudent (String npm)
{
    studentMapper.deleteStudent(npm);
    log.info("student " + npm + "deleted");
}
```

Pada class *StudentServiceDatabase* implementasikan method *deleteStudent* dari class interface *StudentService*. Dalam method tersebut gunakan method *deleteStudent* dari *studentMapper* untuk melakukan penghapusan data dari database.

```
@RequestMapping("/student/delete/{npm}")
public String delete (Model model, @PathVariable(value = "npm") String npm)
{
    StudentModel student = studentDAO.selectStudent(npm);
    if (student != null) {
        studentDAO.deleteStudent (npm);
        return "delete";
    }
    model.addAttribute("npm", npm);
    return "not-found";
}
```

Tambahkan method *delete* pada class *StudentController*. Dalam method tersebut terdapat kode untuk memvalidasi data *student* dengan NPM tersebut ada, jika ada maka akan me-*return* halaman *delete*, jika tidak ada maka akan me-*return* halaman *not-found*.

```
<!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>
<a th:href="/student/delete/' + ${student.npm}">Delete Data</a>
<a th:href="/student/update/' + ${student.npm}">Update Data</a>
<br/>
</div>
</body>
</html>
```

Tambahkan sebuah *link* untuk menghapus data *student* pada halaman *viewall*.

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

Berikut adalah halaman *not-found* jika data yang akan dihapus berdasarkan NPM yang diberikan tidak ditemukan.

```
<html>
<head>
<title>Delete</title>
</head>
<body>
<h2>Data berhasil dihapus</h2>
</body>
</html>
```

Berikut adalah halaman *delete* untuk menginformasikan bahwa data sudah berhasil dihapus, halaman ini muncul ketika NPM yang dimasukkan ada datanya.

All Students

No. 1

NPM = 4213

Name = zetra wibawa putra

GPA = 9.0

[Delete Data](#) [Update Data](#)

Berikut adalah data dari semua *student*.

← → ↻ localhost:8080/student/delete/4213

Data berhasil dihapus

Ketika diklik “Delete Data” maka data *student* tersebut terhapus.

Method Update

```
@Update("UPDATE student SET name=#{student.name}, gpa=#{student.gpa} WHERE npm=#{student.npm}")
void updateStudent(@Param("student") StudentModel student);
```

Pertama membuat sebuah method *abstract updateStudent* pada *class interface StudentMapper*. Gunakan anotasi *@Update* untuk mengeksekusi query meng-*update* data dalam database berdasarkan NPM yang diberikan. Parameter method tersebut menggunakan objek dari *student* dan menggunakan *@Param* untuk menangkap data dengan inisial “student”.

```
void updateStudent(StudentModel student);
```

Tambahkan method *abstract updateStudent* pada *class interface StudentService*. Pada parameter method tersebut menerima sebuah objek *student*.

```
@Override
public void updateStudent(StudentModel student) {
    // TODO Auto-generated method stub
    studentMapper.updateStudent(student);
    Log.info("student " + student.getNpm() + " updated");
}
}
```

Pada *class StudentServiceDatabase* implementasikan method *updateStudent* dari *class interface StudentService*. Dalam method tersebut gunakan method *updateStudent* dari *studentMapper* untuk melakukan penghapusan data dari database.

```
@RequestMapping("/student/update/{npm}")
public String update (Model model, @PathVariable(value= "npm") String npm) {
    StudentModel student = studentDAO.selectStudent(npm);
    if (student != null) {
        model.addAttribute("student", student);
        return "form-update";
    }
    model.addAttribute("npm", npm);
    return "not-found";
}
```

Tambahkan method *update* pada *class StudentController*. Dalam method tersebut terdapat kode untuk memvalidasi data *student* dengan NPM tersebut ada, jika ada maka akan me-*return* halaman *form-update*, jika tidak ada maka akan me-*return* halaman *not-found*.

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit (
    @RequestParam(value = "npm", required = false) String npm,
    @RequestParam(value = "name", required = false) String name,
    @RequestParam(value = "gpa", required = false) double gpa) {
    StudentModel student = new StudentModel(npm, name, gpa);
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Tambahkan method *updateSubmit* pada *class StudentController*. Method tersebut untuk menyimpan data *student* yang sudah dimasukkan pada *form*.

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

Berikut adalah halaman *not-found* jika data yang akan dihapus berdasarkan NPM yang diberikan tidak ditemukan.

```

<body>
  <h1 class="page-header">Problem Editor</h1>
  <form action="/student/update/submit" method="post">
    <div>
      <label for="npm">NPM</label>
      <input type="text" name="npm" readonly="true" th:value="{student.npm}" />
    </div>
    <div>
      <label for="name">Name</label>
      <input type="text" name="name" th:value="{student.name}"/>
    </div>
    <div>
      <label for="gpa">GPA</label>
      <input type="text" name="gpa" th:value="{student.gpa}"/>
    </div>
    <div>
      <button type="submit" name="action" value="save">Save</button>
    </div>
  </form>

```

Berikut adalah halaman *form-update* untuk mengubah data *student* berdasarkan NPM, halaman ini muncul ketika NPM yang diberikan *valid*.

```

<html>
  <head>
    <title>Update</title>
  </head>
  <body>
    <h2>Data berhasil di Update</h2>
  </body>
</html>

```

Berikut adalah halaman *success-update*, halaman ini muncul ketika data berhasil diubah dan disimpan kedalam database.

← → ↻ localhost:8080/student/viewall	
All Students	
No. 1	
NPM = 1	
Name = Cinta Kamu	
GPA = 110.0	
Delete Data	Update Data
<hr/>	
No. 2	
NPM = 4213	
Name = zetra wibawa putra	
GPA = 9.0	
Delete Data	Update Data
<hr/>	

Berikut adalah data dari seluruh data *student* yang ada di database, dimana halaman ini diakses lewat localhost:8080/student/viewall.

← → ↻ ⓘ localhost:8080/student/update/4213

Problem Editor

NPM	4213
Name	zetra wibawa putra
GPA	9.0
<input type="button" value="Save"/>	

← → ↻ ⓘ localhost:8080/student/update/4213

Problem Editor

NPM	4213
Name	zetra wibawa putra hendi
GPA	9.0
<input type="button" value="Save"/>	

Berikut adalah halaman pengubahan data *student*, kasus ini mengganti nama dari “zetra wibawa putra” menjadi “zetra waibawa putra hendi”.

← → ↻ ⓘ localhost:8080/student/update/submit

Data berhasil di Update

Ketika sesudah di submit maka data berhasil di Update.

Method Object

```
@PostMapping("/student/update/submit")
public String updateSubmitObject (@ModelAttribute StudentModel student) {
    studentDAO.updateStudent(student);
    return "success-update";
}
```

Pada *class StudentController* tambahkan sebuah method *updateSubmitObject* (tanpa mengubah method *updateSubmit*). Method ini sama dengan method *updateSubmit* perbedaannya adalah method *updateSubmitObject* mengubah sebuah data dari database dengan sebuah objek dari *form* halaman yang diinputkan atau tanpa mendefinisikan *@RequestParam* berdasarkan masing-masing variabel yang dimasukkan.

```
<form th:action="@{/student/update/submit}" th:object="${student}" method="post">
    <div>
        <label for="npm">NPM</label>
        <input type="text" name="npm" readonly="true" th:value="${student.npm}" th:field="*{npm}" />
    </div>
    <div>
        <label for="name">Name</label>
        <input type="text" name="name" th:value="${student.name}" th:field="*{name}" />
    </div>
    <div>
        <label for="gpa">GPA</label>
        <input type="text" name="gpa" th:value="${student.gpa}" th:field="*{gpa}" />
    </div>
    <div>
        <button type="submit" name="action" value="save">Save</button>
    </div>
</form>
</body>
```

Berikut adalah halaman *form-update* yang sudah diubah kode pada *form*-nya.

Soal

1. Dengan melakukan validasi masing-masing variabel yang didapatkan @RequestParam dari hasil *input form* tersebut. Validasi ini penting atau dibutuhkan ketika ada suatu masukan membutuhkan *value* yang khusus atau bersifat opsional.
2. Menggunakan method POST ketika data yang akan dikirim tersebut adalah data yang penting atau bersifat *private*, untuk menjamin keamanan data tersebut maka pengiriman methodnya menggunakan method POST dibandingkan dengan method GET. GET memungkinkan pengguna dapat memasukkan sebuah data di URL secara langsung. Iya perlu penanganan yang berbeda.
3. Bisa, karena pada @RequestMapping dapat menerima lebih satu jenis method POST dan GET.