

Tutorial Delete







localhost:8080/student/add

Problem Editor

NPM





Name

GPA



localhost:8080/student/add/submit?npm=1506757724&name=Ernest&gpa=3.

Data berhasil ditambahkan



localhost:8080/student/viewall

All Students

[Delete Data](#)

No. 1

NPM = 123

Name = Chanek

GPA = 3.6

[Delete Data](#)

No. 2

NPM = 124

Name = Chanek Jr.

GPA = 3.4

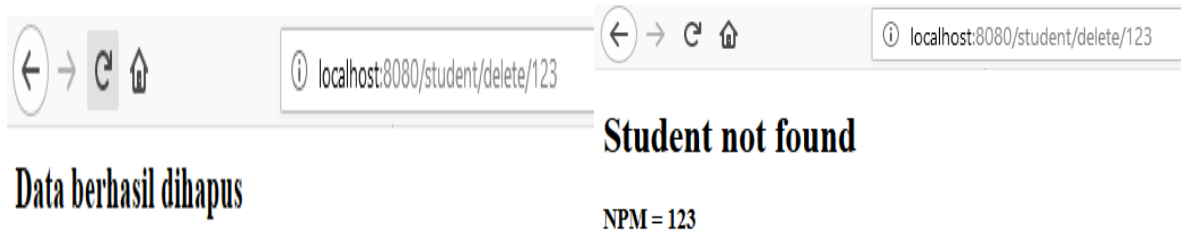
[Delete Data](#)

No. 3

NPM = 1506757724

Name = Ernest

GPA = 3.2



Method-method tutorial delete

1. Method deleteStudent pada class StudentMapper

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

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

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

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

Method ini memiliki anotasi Delete yang akan berhubungan dengan penggunaan operasi "DELETE" pada *database*. *Query* yang ada memiliki fungsi menghapus data dari student dengan npm tertentu. Jika method deleteStudent dipanggil dengan parameter String npm, *query* yang ada pada anotasi akan bekerja dan melakukan operasi "DELETE" pada *database* sesuai dengan parameter "npm" yang diberikan.

2. Method deleteStudent pada class StudentService

```
7 public interface StudentService
8 {
9     StudentModel selectStudent (String npm);
10
11
12     List<StudentModel> selectAllStudents ();
13
14
15     void addStudent (StudentModel student);
16
17
18     void deleteStudent (String npm);
19
20
21     void updateStudent(StudentModel student);
22 }
```

3. Method deleteStudent pada class StudentServiceDatabase

```

@Slf4j
@Service
public class StudentServiceDatabase implements StudentService {
    @Autowired
    private StudentMapper studentMapper;

    @Override
    public StudentModel selectStudent(String npm) {
        log.info("select student with npm {}", npm);
        return studentMapper.selectStudent(npm);
    }

    @Override
    public List<StudentModel> selectAllStudents() {
        log.info("select all students");
        return studentMapper.selectAllStudents();
    }

    @Override
    public void addStudent(StudentModel student) {
        studentMapper.addStudent(student);
    }

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

    @Override
    public void updateStudent(StudentModel student) {
        log.info("student" + student.getNpm() + "updated");
        studentMapper.updateStudent(student);
    }
}

```

Method ini menerima parameter String npm yang datanya akan digunakan untuk memanggil method deleteStudent yang ada pada class studentMapper.

4. Method deleteStudent pada class StudentController

```

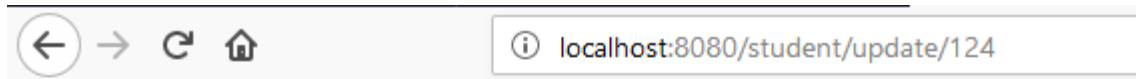
@RequestMapping("/student/delete/{npm}")
public String delete(Model model, @PathVariable(value = "npm") String npm) {
    StudentModel student = studentDAO.selectStudent(npm);

    if (student != null) {
        studentDAO.deleteStudent(npm);
        return "delete";
    } else {
        return "not-found";
    }
}

```

Method ini berfungsi untuk melakukan cek apakah student dengan npm yang diinput ada di dalam database atau tidak. Npm yang akan dicari diambil dari url dengan menggunakan RequestMapping. Student dicari dengan menggunakan method selectStudent jika student ditemukan maka method deleteStudent pada class StudentService dipanggil dengan parameter npm tersebut dan menampilkan delete.html sedangkan jika npm student tidak ditemukan pada *database*, maka akan ditampilkan not-found.html

Tutorial Update

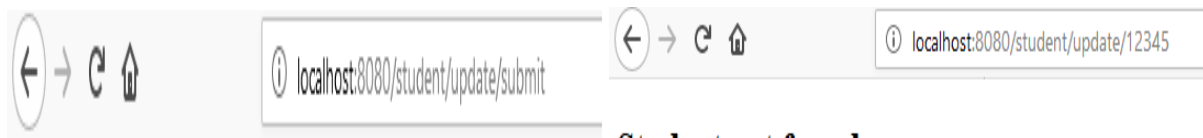


Problem Editor

NPM

Name

GPA



Data berhasil diupdate

Student not found

NPM = 12345

Method-method tutorial update

1. Method updateStudent pada class StudentMapper

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

    @Select("select npm, name, gpa from student")
    List<StudentModel> selectAllStudents ();

    @Insert("INSERT INTO student (npm, name, gpa) VALUES (#{npm}, #{name}, #{gpa})")
    void addStudent (StudentModel student);

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

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

Method ini memiliki anotasi Update yang akan berhubungan dengan penggunaan operasi "UPDATE" pada *database*. Query yang ada memiliki fungsi mengupdate data dari student

dengan npm tertentu. Jika method `updateStudent` dipanggil dengan parameter `StudentModel student`, *query* yang ada pada anotasi akan bekerja dan melakukan operasi "UPDATE" pada *database* sesuai dengan parameter "student" yang diberikan.

2. Method `updateStudent` pada class `StudentService`

```
7 public interface StudentService
8 {
9     StudentModel selectStudent (String npm);
10
11
12     List<StudentModel> selectAllStudents ();
13
14
15     void addStudent (StudentModel student);
16
17
18     void deleteStudent (String npm);
19
20
21     void updateStudent(StudentModel student);
22 }
```

3. Method `updateStudent` pada class `StudentServiceDatabase`

```
@Slf4j
@Service
public class StudentServiceDatabase implements StudentService {
    @Autowired
    private StudentMapper studentMapper;

    @Override
    public StudentModel selectStudent(String npm) {
        log.info("select student with npm {}", npm);
        return studentMapper.selectStudent(npm);
    }

    @Override
    public List<StudentModel> selectAllStudents() {
        log.info("select all students");
        return studentMapper.selectAllStudents();
    }

    @Override
    public void addStudent(StudentModel student) {
        studentMapper.addStudent(student);
    }

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

    @Override
    public void updateStudent(StudentModel student) {
        log.info("student" + student.getNpm() + "updated");
        studentMapper.updateStudent(student);
    }
}
```

Method ini menerima parameter `StudentModel student` yang datanya akan digunakan untuk memanggil method `updateStudent` yang ada pada class `studentMapper`.

4. Method `update` dan `updateSubmit` pada class `StudentController`

```

@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";
    } else {
        return "not-found";
    }
}

```

Method ini berfungsi untuk melakukan cek apakah student dengan npm yang diinput ada di dalam database atau tidak. Npm yang akan dicari diambil dari url dengan menggunakan RequestMapping. Student dicari dengan menggunakan method selectStudent jika student ditemukan maka akan menampilkan form-update.html sedangkan jika npm student tidak ditemukan pada *database*, maka akan ditampilkan not-found.html

```

@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) {

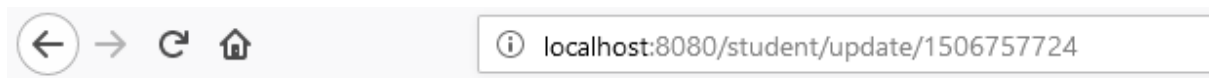
    StudentModel student = studentDAO.selectStudent(npm);
    student.setName(name);
    student.setGpa(gpa);
    studentDAO.updateStudent(student);

    return "success-update";
}

```

Method ini akan berfungsi ketika url yang ditampilkan adalah “/student/update/submit” dan method POST. Method akan menerima parameter String npm, String name, dan double gpa yang didapatkan dari method yang ada pada url. Dimana data ini berasal dari form-update.html yang dipanggil oleh method updateStudent pada class StudentControler. Selanjutnya akan dilakukan pencarian terhadap npm tersebut di database dengan menggunakan method selectStudent, lalu dilakukan perubahan terhadap atribut name dan gpa student menggunakan method set yang dimiliki oleh objek student. Method ini lalu memanggil method updateStudent pada class StudentService dengan parameter student yang telah diubah, lalu menampilkan success-update.html

Tutorial Object sebagai parameter



Problem Editor

NPM
Name
GPA



Data berhasil diupdate

Mengubah method updateSubmit pada StudentController

```
@RequestMapping(value = "/student/update/submit", method = RequestMethod.POST)
public String updateSubmit(StudentModel student) {

    studentDAO.updateStudent(student);
    return "success-update";
}
```

PERTANYAAN

1. Jika menggunakan Object sebagai parameter pada form POST, bagaimana caranya melakukan validasi input yang optional dan input yang required? Apakah validasi diperlukan?

Validasi diperlukan, validasi dapat dilakukan pada Controller dengan mengambil attribute dari model objek

2. Menurut anda, mengapa form submit biasanya menggunakan POST method dibanding GET method? Apakah perlu penanganan berbeda di header atau body method di controller jika form di post dikirim menggunakan method berbeda?

Karena method POST tidak menampilkan isi dari form di url sehingga tidak sembarang orang bisa mengganti nilai dari form tersebut.

Perlu penanganan berbeda.

- 3. Apakah mungkin satu method menerima lebih dari satu jenis request method, misalkan menerima GET sekaligus POST?**

Tidak mungkin, karena satu method hanya bisa menerima satu request method.