**Spring boot and mybatis**

2017-07-25

이승진

**학습목표**

Spring boot + mybatis 기술을 사용하여 DB 조회, 수정, 삽입, 삭제 기능을 구현한다.

**목차**

[1. 배경지식 2](#_Toc489024268)

[1) ORM (Object Relational Mapping) 2](#_Toc489024269)

[2) JPA와 mybatis 비교 2](#_Toc489024270)

[3) mybatis mapper 4](#_Toc489024271)

[4) Auto Increment 필드 (Identity 필드) 4](#_Toc489024272)

[5) Referential Integrity Constraint 4](#_Toc489024273)

[6) 참조 무결성 제약조건 위반 피하기 7](#_Toc489024274)

[2. mybatis1 프로젝트 8](#_Toc489024275)

[1) 프로젝트 생성 8](#_Toc489024276)

[2) pom.xml 10](#_Toc489024277)

[3) application.properties 12](#_Toc489024278)

[3. DTO 클래스 구현 13](#_Toc489024279)

[1) Student.java 13](#_Toc489024280)

[2) Department.java 15](#_Toc489024281)

[4. Mapper 구현 16](#_Toc489024282)

[1) StudentMapper.java 16](#_Toc489024283)

[2) StudentMapper.xml 17](#_Toc489024284)

[3) mapper 구현 규칙 18](#_Toc489024285)

[4) auto increment 필드와 insert 22](#_Toc489024286)

[5) DepartmentMapper.java 23](#_Toc489024287)

[6) DepartmentMapper.xml 23](#_Toc489024288)

[5. 컨트롤러 구현 24](#_Toc489024289)

[1) StudentController 24](#_Toc489024290)

[6. 뷰 구현 26](#_Toc489024291)

[1) student/list.jsp 26](#_Toc489024292)

[2) student/edit.jsp 27](#_Toc489024293)

[7. 정적 컨텐츠 28](#_Toc489024294)

[1) res/common.css 28](#_Toc489024295)

[2) res/common.js 28](#_Toc489024296)

[3) index.jsp 28](#_Toc489024297)

[8. 실행 29](#_Toc489024298)

[1) 결과 소스코드 29](#_Toc489024299)

[2) 참조 무결성 제약조건 위반 에러 29](#_Toc489024300)

# 배경지식

## ORM (Object Relational Mapping)

ORM 에서 Object 는 객체지향 언어의 객체를 의미한다.

Ralational 은 관계형 데이터베이스(Relational Database)의 데이터를 의미한다.

Mapping이 의미하는 것은 객체지향 언어의 객체와 관계형 데이터를 서로 변환해 준다는 것이다.

ORM 이란?

관계형 데이터베이스에서 조회한 데이터를 Java 객체로 변환하여 리턴해 주고,

Java 객체를 관계형 데이터베이스에 저장해 주는

라이브러리 혹은 기술을 말한다.

Java ORM 기술로 유명한 것은

mybatis, Hibernate, JPA 이다.

mybatis와 Hibernate는 오픈소스 프로젝트이고 jar 라이브러리 형태로 제공된다.

JPA(Java Persistence API)는 제품의 이름이 아니고, API 표준의 이름이다.

JPA 표준 규격대로 만들어진 제품 중에서 유명한 것이 Hibernate 오픈소스 라이브러리이다.

우리가 사용하는 Spring JPA에 Hibernate 라이브러리가 포함되어 있다.

우리 나라의 전자 정부 표준 프레임웍에서 Spring mybatis를 채택하고 있기 때문에, 우리 나라 공공 프로젝트에서 mybatis를 사용하는 경우가 많다. 그렇지만 JPA가 좀 더 미래지향적인 기술이기 때문에 점점 JPA를 사용하는 경우가 늘어나고 있다.

## JPA와 mybatis 비교

MySQL, Oracle, SQL Server 등 DBMS 제품 마다 SQL 문법은 조금씩 다르다.

그래서 DBMS 제품을 교체하려면, SQL 문장도 수정해야 한다.

### JPA의 장점

SQL 명령을 구현할 필요가 없다. 그래서 DBMS 제품을 교체하더라도 소스코드를 수정할 필요가 없다.

자동으로 처리되는 부분이 많아서, 구현할 소스코드의 양이 상대적으로 적다.

관계형 데이터베이스가 아니더라도 적용할 수 있다.

### JPA의 단점

복잡한 조회 명령을 구현해야 할 때, 익숙한 SQL 명령으로 구현할 수가 없고, JPA의 고급 기능을 공부해야 한다.

### mybatis의 장점

익숙한 SQL 명령으로 구현할 수 있다.

SQL 문장을 그대로 사용하여 구현하기 때문에, SQL 문장에 익숙한 개발자에게 myBatis가 편하다.

DB 조회 결과를 복잡한 객체 구조로 변환해야 할 때 myBatis 기능이 좋다.

mybatis의 resultMap 기능이 바로 그것이다.

이 기능은 복잡한 보고서(report)를 출력해야 할 때, 특히 유용하다.

데이터베이스 성능 개선을 위해, 어떤 인덱스를 생성해야 하는지 파악하기 위해,

SQL 쿼리들을 분석해야 하는데, 이때 myBatis는 SQL 문장을 그대로 사용하기 때문에,

SQL 쿼리 분석하기 편하다.

### mybatis의 단점

구현할 소스코드의 양이 상대적으로 많다.

관계형 데이터베이스에만 적용할 수 있다.

DBMS 제품을 교체하면 SQL 소스코드를 수정해야 한다.

Oracle, MS SQL Server, mySQL 등 DBMS 마다 SQL 문법이 약간씩 차이가 있다.

그래서 DBMS를 바꾸면 SQL 문도 수정해야 하는 불편함이 있다.

SQL 문을 사용하지 않는 Hibernate, JPA에는 이런 문제가 없다.

## mybatis mapper

데이터베이스 테이블에 대한 SELECT/INSERT/UPDATE/DELETE SQL 명령들을 mybatis mapper에 구현한다.

보통 데이터베이스 테이블 한 개당 mybatis mapper를 한 개를 구현한다.

mybatis mapper는 Java Interface 파일 한 개와, XML 파일 한 개로 구현된다.

DB 테이블에 대한 조회, 삽입, 수정, 삭제 SQL 명령을 mapper XML 파일에 구현한다.

그리고 이 명령을 호출하기 위한 Java 메소드를 mapper Java Interface 파일에 선언한다.

예를 들어 Student 테이블에 대한 mybatis mapper는,

StudentMapper Java 인터페이스를 포함하는 StudentMapper.java 파일과

Student 테이블에 대한 SQL 명령이 포함된 StudentMapper.xml 파일로 구현된다.

mapper 메소드를 호출하기 위한 Java 메소드를 Java Interface에 선언하기만 하면 된다.

이 메소드를 구현(implements)할 필요는 없다.

즉 StudentMapper Java Interface만 만들면 되고,

이 인터페이스를 구현(implements)한 Java 클래스를 구현할 필요는 없다.

StudentMapper 인터페이스를 구현한 Java 클래스를

mybatis spring이 자동으로 구현해 주기 때문이다.

## Auto Increment 필드 (Identity 필드)

Student 테이블의 기본키(primary key)는 id 필드이다.

MySQL에서 Student 테이블을 생성할 때, id 필드를 Auto Increment 필드로 지정하였다.

Auto Increment 필드의 값은 1부터 시작하는 일련번호이다.

테이블에 새 레코드를 insert 할 때, 이 필드의 값에 일련번호가 자동으로 부여된다.

Auto Increment 필드의 값인 자동으로 부여되기 때문에,

insert나 update SQL 문에서 이 필드의 값을 저장하는 것이 에러이다.

## Referential Integrity Constraint

Student 테이블의 departmentId 필드는 외래키(foreign key) 이다.

이 필드의 값은 department 테이블의 기본키인 id 필드값과 일치해야 한다.

Student 테이블과 Department 테이블을 조인할 때, departmentId 필드를 사용한다.

SELECT s.\*, d.departmentName

FROM Student s LEFT JOIN department d ON s.departmentId = d.id

Department 테이블에서 레코드를 한 개 삭제 하려고 할 때,

만약 Student 테이블의 어떤 레코드의 departmentId 필드 값이,

그 삭제하려는 Department 레코드의 id 필드 값과 일치한다면,

삭제는 실패하고 에러가 발생한다.

이 에러를 참조 무결성 제약조건 위반(referential intergity constraint violation)이라고 부른다.

쉽게 표현하자면, 국어국문학과 소속 학생들이 존재한다면, 국어국문학과를 삭제할 수 없다는 얘기다.

Register 테이블에 외래키인 studentId 필드가 들어있다.

그래서 Student 테이블의 레코드를 삭제하려 할 때, 참조 무결성 제약조건 위반 에러가 발생할 수 있다.

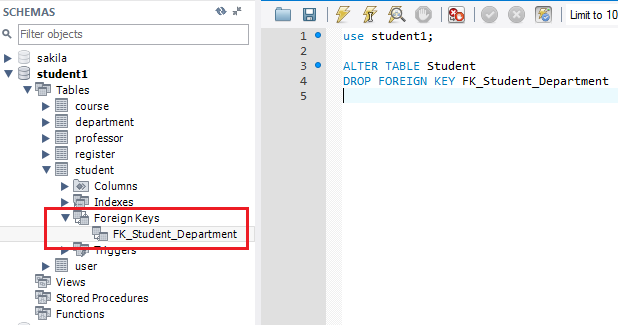
쉽게 표현하자면, 201132050 학생의 수강신청 내역이 존재한다면, 그 학생을 삭제할 수 없다는 얘기다.

데이터베이스가 참조 무결성 제약조건을 실시간 검사해 준다.

테이블을 생성할 때, 데이터베이스가 참조 무결성 제약조건을 설정해 주는 것이 바람직하다.

참조 무결성을 제약조건을 외래키 제약조건이라고도 부른다.

### 외래키 제약조건 제거하기



Student 테이블에 FK\_Student\_Department 이름의 외래키 제약조건이 이미 설정되어 있다.

이 외래키 제약조건을 삭제하는 명령은 다음과 같다.

|  |
| --- |
| ALTER TABLE Student  DROP FOREIGN KEY FK\_Student\_Department; |

### 외래키 제약조건 생성하기

Student 테이블의 departmentId 필드와 Department 테이블의 id 필드 사이에

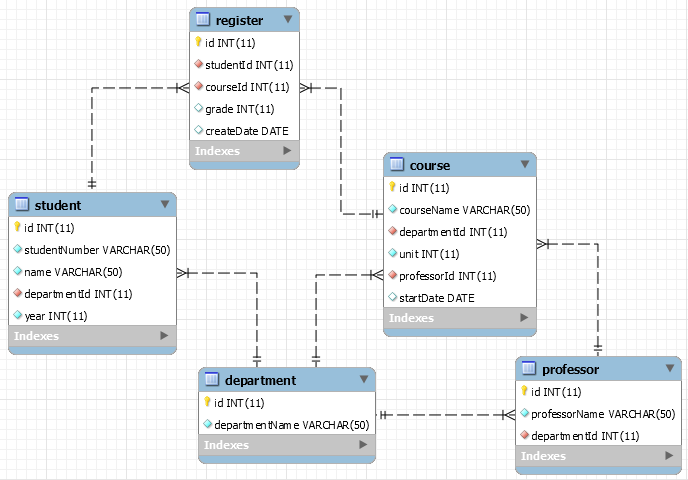
외래키 제약조건을 생성하는 명령은 다음과 같다.

|  |
| --- |
| ALTER TABLE Student  ADD CONSTRAINT FK\_Student\_Department  FOREIGN KEY (departmentId) REFERENCES Department(id); |

제약조건의 이름은 FK\_Student\_Department 이다.

외래키는 Student 테이블의 departmentId 필드이다.

이 필드는 Department 테이블의 id 필드를 참조(references)한다.



register 테이블은 수강신청 레코드를 저장하는 테이블이다.

register 테이블의 studentId 필드는 foreign key 이다.

studentId 필드의 값은 이름 그대로 Student 테이블의 id 필드 값이다.

studentId 필드는 수강신청을 한 학생(Student)을 가르킨다.

register 테이블의 courseId 도 foreign key 이다.

수강신청 대상 강좌(course)의 id 이다.

학생(student) 레코드를 삭제하려면, 먼저 그 학생의 수강신청register) 레코드들부터 삭제해야 한다.

그렇지 않고 학생 레코드부터 삭제하면 참조 무결성 위반 에러가 발생한다.

## 참조 무결성 제약조건 위반 피하기

참조 무결성 제약조건 위반 에러를 피하는 방법은 다음과 같다.

### 먼저 삭제하기

Department 테이블의 레코드를 삭제하기 전에, 먼저 그 레코드를 참조하는 Student 레코드들을 전부 삭제한다.

예:

|  |
| --- |
| DELETE FROM Student WHERE departmentId = 2;  DELETE FROM Department WHERE id = 2; |

### Cascade Delete 옵션

외래키 제약 조건을 생성할 때, Casecade Delete 옵션을 지정할 수 있다.

이 옵션이 지정된 경우에는, Department 테이블의 레코드를 삭제할 때,

그 레코드를 참조하는 Student 레코드들이 전부 자동으로 삭제된다.

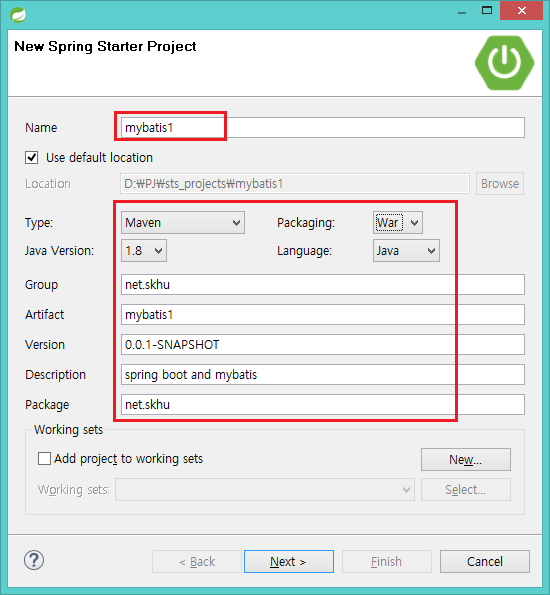
Casecade Delete 옵션 지정 방법

|  |
| --- |
| ALTER TABLE Student  ADD CONSTRAINT FK\_Student\_Department  FOREIGN KEY (departmentId) REFERENCES Department(id)  ON DELETE CASCADE; |

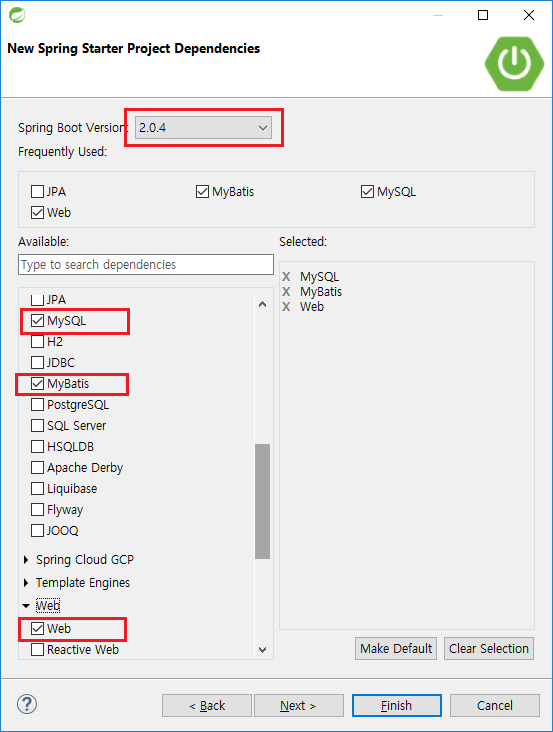
# mybatis1 프로젝트

## 프로젝트 생성

메뉴: File - New - Spring Starter Project



Packaging: War



## pom.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | <?xml version="1.0" encoding="UTF-8"?>  <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>net.skhu</groupId>  <artifactId>mybatis1</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>war</packaging>  <name>mybatis1</name>  <description>spring boot and mybatis</description>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.4.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.mybatis.spring.boot</groupId>  <artifactId>mybatis-spring-boot-starter</artifactId>  <version>1.3.2</version>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <scope>runtime</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

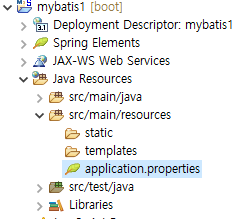
자동 생성된 pom.xml 파일의 내용에서, 노란색으로 표시한 부분만 추가되었다.

pom.xml 파일 전체를 수정하지 말고, 노란색 부분만 복사해서 붙여넣자.

|  |
| --- |
| <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

뷰 파일(\*.jsp)에서 JSTL 확장 태그를 사용하기 위해 필요한 라이브러리를 등록하는 maven dependency 이다.

## application.properties



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student1  spring.datasource.username=user1  spring.datasource.password=test123  mybatis.type-aliases-package=net.skhu.dto |

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/ |

뷰 파일(\*.jsp)이 위차할 폴더를 지정한다.

|  |
| --- |
| spring.mvc.view.suffix=.jsp |

뷰 파일의 확장자를 지정한다.

|  |
| --- |
| spring.datasource.driver-class-name=com.mysql.jdbc.Driver |

JDBC 드라이버 클래스의 이름를 지정한다.

위 이름은 MySQL JDBC 드라이버 클래스이다.

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/student1 |

데이터베이스 서버 IP 와 데이터베이스 이름을 지정한다.

|  |
| --- |
| spring.datasource.username=user1  spring.datasource.password=test123 |

데이터베이스에 연결할 계정을 지정한다.

|  |
| --- |
| mybatis.type-aliases-package=net.skhu.dto |

데이터베이스의 조회 결과 데이터를 담을 클래스들의 패키지를 지정한다.

mybatis mapper XML 파일에서 select 태그의 resultType으로 등록된 클래스들의 패키지를 지정한다.

예: <select id="findById" resultType="Student">

select 태그의 resultType으로 등록된 Student 클래스의 패키지는 net.skhu.dto 이어야 한다.

# DTO 클래스 구현

## Student.java

### src/main/java/net/skhu/dto/Student.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58 | package net.skhu.dto;  public class Student {  int id;  String studentNumber;  String name;  int departmentId;  int year;  String departmentName;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getStudentNumber() {  return studentNumber;  }  public void setStudentNumber(String studentNumber) {  this.studentNumber = studentNumber;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public int getDepartmentId() {  return departmentId;  }  public void setDepartmentId(int departmentId) {  this.departmentId = departmentId;  }  public int getYear() {  return year;  }  public void setYear(int year) {  this.year = year;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  } |

Student 테이블에서 조회한 데이터를 채울 DTO (Data Transfer Object) 이다.

Student 테이블에 departmentName 필드는 없지만,

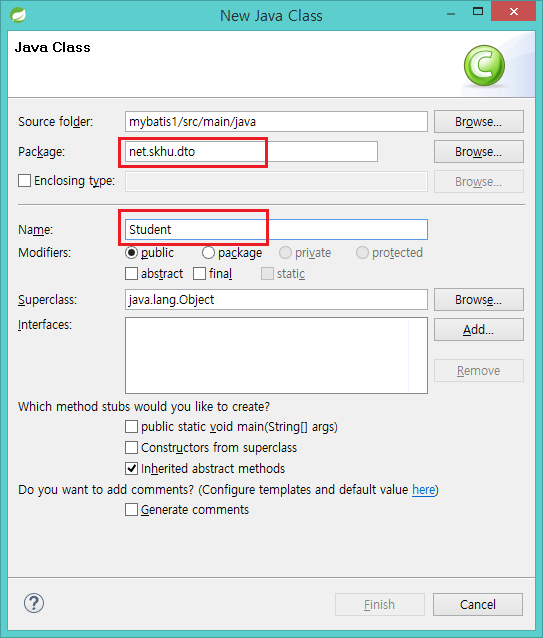
Student 테이블에서 조회할 때, Department 테이블과 조인하여 departmentName 필드도 조회하기 때문에,

그 조회 결과를 채우기, 위해 departmentName 멤버 변수가 Student 클래스에 추가되었다.

예;

|  |
| --- |
| SELECT s.\*, d.departmentName  FROM Student s JOIN department d ON s.departmentId = d.id |

### 클래스 생성할 때 패키지 지정하기



## Department.java

### src/main/java/net/skhu/dto/Department.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.dto;  public class Department {  int id;  String departmentName;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  } |

Department 테이블에서 조회한 데이터를 채울 DTO (Data Transfer Object) 이다.

# Mapper 구현

## StudentMapper.java

### src/main/java/net/skhu/mapper/StudentMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import net.skhu.dto.Student;  @Mapper  public interface StudentMapper {  Student findOne(int id);  Student findByStudentNumber(String studentNumber);  List<Student> findAll();  void insert(Student student);  void update(Student student);  void delete(int id);  } |

DB 의 Student 테이블에 대한 조회, 삽입, 수정, 삭제 SQL 명령을 StudentMapper.xml 파일에 구현한다.

그리고 이 명령을 호출하기 위한 메소드를 StudentMapper 인터페이스에 선언한다.

StudentMapper 인터페이스의 메소드들의 리턴 타입, 이름, 파마티러 타입에 주목하자.

StudentMapper 인터페이스의 메소드들의 이름은 StudentMapper.xml 파일의 태그들의 id 애트리뷰트 값과 일치해야 한다.

StudentMapper 인터페이스의 메소드들의 파라미터는 StudentMapper.xml 파일의 태그들의 mybatis 파라미터와 잋치해야 한다.

StudentMapper 인터페이스의 메소드들의 리턴 타입은 StudentMapper.xml 파일의 태그들의 resultType 애트리뷰트 값과 일치해야 한다.

## StudentMapper.xml

### src/main/java/net/skhu/mapper/StudentMapper.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="net.skhu.mapper.StudentMapper">  <select id="findOne" resultType="Student">  SELECT \* FROM Student WHERE id = #{id}  </select>    <select id="findByStudentNumber" resultType="Student">  SELECT \* FROM Student WHERE studentNumber = #{studentNumber}  </select>  <select id="findAll" resultType="Student">  SELECT s.\*, d.departmentName  FROM Student s LEFT JOIN department d ON s.departmentId = d.id  </select>  <insert id="insert" useGeneratedKeys="true" keyProperty="id">  INSERT Student (studentNumber, name, departmentId, year)  VALUES (#{studentNumber}, #{name}, #{departmentId}, #{year})  </insert>  <update id="update">  UPDATE Student SET  studentNumber = #{studentNumber},  name = #{name},  departmentId = #{departmentId},  year = #{year}  WHERE id = #{id}  </update>  <delete id="delete">  DELETE FROM Student WHERE id = #{id}  </delete>  </mapper> |

DB 의 Student 테이블에 대한 조회, 삽입, 수정, 삭제 SQL 명령을 StudentMapper.xml 파일에 구현한다.

이 파일은 StudentMapper.java 파일과 동일한 폴더에 있어야 한다.

## mapper 구현 규칙

### namespace 일치

StudentMapper.xml

|  |
| --- |
| <mapper namespace="net.skhu.mapper.StudentMapper"> |

namespace 애트리뷰트 값은 StudentMapper 인터페이스의 이름과 패키지와 정확하게 일치해야 한다.

그렇지 않은 경우 StudentMapper bean을 생성할 수 없다는 에러가 발생한다.

StudentMapper.java

|  |
| --- |
| package net.skhu.mapper;  @Mapper  public interface StudentMapper {  . . .  } |

### 메소드 이름 일치

StudentMapper.xml

|  |
| --- |
| <select id="findOne" resultType="Student">  <select id="findByStudentNumber" resultType="Student">  <select id="findAll" resultType="Student">  <insert id="insert" useGeneratedKeys="true" keyProperty="id">  <update id="update">  <delete id="delete"> |

id 애트리뷰트 값은 StudentMapper 인터페이스의 메소드 이름과 일치해야 한다.

StudnetMapper.java

|  |
| --- |
| Student findOne(int id);  Student findByStudentNumber(String studentNumber);  List<Student> findAll();  void insert(Student student);  void update(Student student);  void delete(int id); |

### resultType: 패지키

application.properties

|  |
| --- |
| mybatis.type-aliases-package=net.skhu.dto |

mybatis mapper XML 파일에서 select 태그의 resultType으로 등록된 클래스들의 패키지를 지정한다.

StudentMapper.xml

|  |
| --- |
| <select id="findById" resultType="Student"> |

select 태그의 resultType으로 등록된 Student 클래스의 패키지는 net.skhu.dto 이어야 한다.

### resultType: 레코드 한 개 리턴

|  |
| --- |
| <select id="findById" resultType="Student"> |

id 애트리뷰트 값은 StudentMapper 인터페이스의 메소드 이름과 일치해야 한다.

resultType 애트리뷰트 값은 메소드의 리턴 타입과 일치해야 한다.

이 SQL 명령은 Student 레코드 한 개를 조회한다.

조회 결과는 Student 자바 객체에 자동으로 채워져서 리턴된다.

DB 조회 결과가 자바 객체에 자동으로 채워질 때,

조회 결과 컬럼 제목과 자바 객체의 set 메소드 이름이 일치해야 한다.

StudentMapper.java

|  |
| --- |
| Student findOne(int id);  Student findByStudentNumber(String studentNumber); |

StudentMapper.xml

|  |
| --- |
| <select id="findOne" resultType="Student">  <select id="findByStudentNumber" resultType="Student"> |

### resultType: 레코드 여러 개 리턴

|  |
| --- |
| <select id="findAll" resultType="Student"> |

findAll 태그의 SQL 명령은 Student 레코드 여러 개를 조회한다.

조회 결과 레코드 각각은 Student 객체에 채워지고, Student 객체는 List<Student> 객체에 채워져서 리턴된다.

그래서 findAll 자바 메소드의 리턴 타입은 List<Studnet> 이다.

|  |
| --- |
| List<Student> findAll(); |

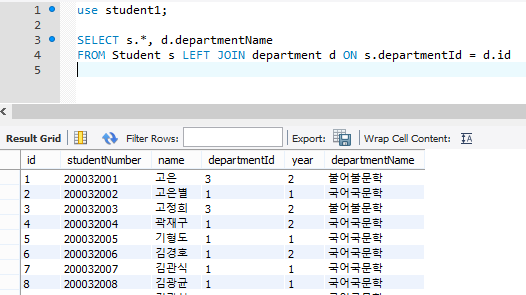
findAll 메소드의 리턴 타입은 List<Student> 이지만, findAll 태그의 resultType은 Student 임에 주의하자.

### 조회 결과 컬럼명 일치

DB 조회 결과가 resultType 자바 객체에 자동으로 채워질 때,

조회 결과 컬럼 제목과 resultType 자바 객체의 set 메소드 이름이 일치해야 한다.

조회 결과의 예



|  |  |
| --- | --- |
| 조회 결과 컬럼명 | resultType자바 객체의 set 메소드 이름 |
| id | setId |
| studentNumber | setStudentNumber |
| name | setName |
| departmentId | setDepartmentId |
| year | setYear |
| departmentName | setDepartmentName |

### mybatis 파라미터: 파라미터 한 개

StudentMapper.java

|  |
| --- |
| Student findOne(int id);  Student findByStudentNumber(String studentNumber);  void delete(int id); |

StudentMapper.xml

|  |
| --- |
| <select id="findOne" resultType="Student">  SELECT \* FROM Student WHERE id = #{id}  </select>    <select id="findByStudentNumber" resultType="Student">  SELECT \* FROM Student WHERE studentNumber = #{studentNumber}  </select>  <delete id="delete">  DELETE FROM Student WHERE id = #{id}  </delete> |

#{id}, #{studentNumber} 부분이 mybatis 파라미터이다.

메소드를 호출할 때 전달된 파라미터 값이, SQL 문장의 mybatis 파라미터 부분에 채워져서 SQL 문장이 실행된다.

mybatis 파라미터로 전달할 값이 한 개이고,

값의 타입이 int, long, float, double, boolean 등 기본 자료형이나

String, Date, Time, Timestamp 클래스 객체인 경우에는

위와 같은 방법으로 구현한다.

여기서 Java 파라미터 변수 이름은 중요하지 않다.

중요한 것은 파라미터가 핸 개이고,

파라미터 타입이 int, long, float, double, boolean 등 기본 자료형이거나

String, Date, Time, Timestamp 클래스 중 하나의 객체이어야 한다는 점이다.

### mybatis 파라미터: 파라미터 여러 개

StudentMapper.java

|  |
| --- |
| void insert(Student student);  void update(Student student); |

StudentMapper.xml

|  |
| --- |
| <insert id="insert" useGeneratedKeys="true" keyProperty="id">  INSERT Student (studentNumber, name, departmentId, year)  VALUES (#{studentNumber}, #{name}, #{departmentId}, #{year})  </insert>  <update id="update">  UPDATE Student SET  studentNumber = #{studentNumber},  name = #{name},  departmentId = #{departmentId},  year = #{year}  WHERE id = #{id}  </update> |

#{...} 부분이 mybatis 파라미터이다.

여기서 Java 파라미터 변수 이름은 중요하지 않다.

중요한 것은 Java 파라미터 변수의 타입이 DTO 클래스이어야 하고

이 클래스의 get 메소드 이름과 mybatis 파라미터의 이름이 일치해야 한다는 점이다.

|  |  |
| --- | --- |
| Student 클래스의 get 메소드 | mybatis 파라미터 |
| getId | #{id} |
| getStudentNumber | #{studentNumber} |
| getName | #{name} |
| getDepartmentId | #{departmentId} |
| getYear | #{year} |

## auto increment 필드와 insert

Student 테이블에 새 레코드를 insert 할 때, auto increment 필드인 id 필드 값은, 자동으로 부여된다.

그래서 insert SQL 문에 id 필드값은 지정하지 않았다.

|  |
| --- |
| INSERT Student (studentNumber, name, departmentId, year)  VALUES (#{studentNumber}, #{name}, #{departmentId}, #{year}) |

위 insert SQL 문이 실행되어 새 레코드가 저장된 후에,

값이 자동 부여된, 그 새 레코드의 id 필드값이 Student 객체의 id 속성에 자동으로 채워진다.

예:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Student student = new Student();  student.setStudentNumber("201132091");  student.setName("홍길동");  student.setDepartmentId(1);  student.setYear(1);  System.out.println(student.getId());  studentMapper.insert(student);  System.out.println(student.getId()); |

줄7에서 출력되는 id 값은 0 이다.

Student 객체의 id 멤버 변수에 아직 아무것도 대입되지 않았기 때문이다.

줄9에서 출력되는 id 값은, 줄8에서 저장된 새 레코드의 id 필드값이다.

이 값은 0 이 아니다.

|  |
| --- |
| <insert id="insert" useGeneratedKeys="true" keyProperty="id">  INSERT Student (studentNumber, name, departmentId, year)  VALUES (#{studentNumber}, #{name}, #{departmentId}, #{year})  </insert> |

위 insert 태그에서 노란색으로 칠한 부분이 의미하는 것은,

새 레코드의 id 필드값을 Student 객체의 id 속성에 대입해 달라는 것이다.

따라서 노란색으로 칠한 부분을 빼먹으면, 새 레코드의 id 필드값을 알기 어렵다.

많은 레코드가 동시에 insert 되는 경우에, 방금 전에 내가 insert 한 레코드가 무엇인지 구별하기 어렵기 때문이다.

## DepartmentMapper.java

### src/main/java/net/skhu/mapper/DepartmentMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import net.skhu.dto.Department;  @Mapper  public interface DepartmentMapper {  List<Department> findAll();  } |

## DepartmentMapper.xml

### src/main/java/net/skhu/mapper/DepartmentMapper.xml

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="UTF-8" ?>  <!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="net.skhu.mapper.DepartmentMapper">  <select id="findAll" resultType="Department">  SELECT \* FROM Department  </select>  </mapper> |

# 컨트롤러 구현

## StudentController

### src/main/java/net/skhu/controller/StudentController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66 | package net.skhu.controller;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RequestParam;  import net.skhu.dto.Department;  import net.skhu.dto.Student;  import net.skhu.mapper.DepartmentMapper;  import net.skhu.mapper.StudentMapper;  @Controller  @RequestMapping("/student")  public class StudentController {  @Autowired StudentMapper studentMapper;  @Autowired DepartmentMapper departmentMapper;  @RequestMapping("list")  public String list(Model model) {  List<Student> students = studentMapper.findAll();  model.addAttribute("students", students);  return "student/list";  }  @RequestMapping(value="create", method=RequestMethod.GET)  public String create(Model model) {  Student student = new Student();  List<Department> departments = departmentMapper.findAll();  model.addAttribute("student", student);  model.addAttribute("departments", departments);  return "student/edit";  }  @RequestMapping(value="create", method=RequestMethod.POST)  public String create(Model model, Student student) {  studentMapper.insert(student);  return "redirect:list";  }  @RequestMapping(value="edit", method=RequestMethod.GET)  public String edit(Model model, @RequestParam("id") int id) {  Student student = studentMapper.findOne(id);  List<Department> departments = departmentMapper.findAll();  model.addAttribute("student", student);  model.addAttribute("departments", departments);  return "student/edit";  }  @RequestMapping(value="edit", method=RequestMethod.POST)  public String edit(Model model, Student student) {  studentMapper.update(student);  return "redirect:list";  }  @RequestMapping("delete")  public String delete(Model model, @RequestParam("id") int id) {  studentMapper.delete(id);  return "redirect:list";  }  } |

@Autowired StudentMapper studentMapper;

StudentMapper 인터페이스를 구현한 Java 클래스를, mybatis spring이 자동으로 구현해 주고,

그 클래스의 객체를 한 개 생성하여, studentMapper 멤버 변수에 자동으로 대입(autowired)해 준다.

그래서 studentMapper 멤버 변수를 선언했을 뿐이고, 어떤 객체도 대입해 주지 않았지만,

mybatis spring이 자동으로 생성해준 객체가 이 멤버 변수에 자동으로 대입되어 있기 때문에,

액션 메소드에서 이 객체를 사용할 수 있다.

# 뷰 구현

## student/list.jsp

### src/main/webapp/WEB-INF/views/student/list.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <c:url var="R" value="/" />  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"  rel="stylesheet" media="screen">  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  <script src="${R}res/common.js"></script>  <link rel="stylesheet" href="${R}res/common.css">  </head>  <body>  <div class="container">  <h1>학생목록</h1>  <div class="pull-right mb5">  <a href="create" class="btn btn-info">  <span class="glyphicon glyphicon-user"></span> 학생등록</a>  </div>  <table class="table table-bordered mt5">  <thead>  <tr>  <th>id</th>  <th>학번</th>  <th>이름</th>  <th>학과</th>  <th>학년</th>  </tr>  </thead>  <tbody>  <c:forEach var="student" items="${ students }">  <tr data-url="edit?id=${ student.id }">  <td>${ student.id }</td>  <td>${ student.studentNumber }</td>  <td>${ student.name }</td>  <td>${ student.departmentName }</td>  <td>${ student.year }</td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

## student/edit.jsp

### src/main/webapp/WEB-INF/views/student/edit.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>  <%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>  <c:url var="R" value="/" />  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"  rel="stylesheet" media="screen">  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>  <script src="${R}res/common.js"></script>  <link rel="stylesheet" href="${R}res/common.css">  </head>  <body>  <div class="container">  <h1>학생 ${ student.id > 0 ? "수정" : "등록" }</h1>  <hr />  <form:form method="post" modelAttribute="student">  <div class="form-group">  <label>학번:</label>  <form:input path="studentNumber" class="form-control w200" />  </div>  <div class="form-group">  <label>이름:</label>  <form:input path="name" class="form-control w200" />  </div>  <div class="form-group">  <label>학과</label>  <form:select path="departmentId" class="form-control w200"  itemValue="id" itemLabel="departmentName" items="${ departments }" />  </div>  <div class="form-group">  <label>학년:</label>  <form:input path="year" class="form-control w200" />  </div>  <hr />  <div>  <button type="submit" class="btn btn-primary">  <span class="glyphicon glyphicon-ok"></span> 저장</button>  <c:if test="${ student.id > 0 }">  <a href="delete.do?id=${ student.id }" class="btn btn-danger" data-confirm-delete>  <i class="glyphicon glyphicon-remove"></i> 삭제</a>  </c:if>  <a href="list.do" class="btn btn-info">목록으로</a>  </div>  </form:form>  </div>  </body>  </html> |

# 정적 컨텐츠

## res/common.css

**src/main/webapp/res/common.css**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | body { font-family: 굴림체; }  thead th { background-color: #eee; }  tr:hover td { background-color: #ffe; cursor: pointer; }  input.form-control.w100 { width: 100px; }  input.form-control.w200 { width: 200px; }  input.form-control.w300 { width: 300px; }  input.form-control.w400 { width: 400px; }  select.form-control.w100 { width: 100px; }  select.form-control.w200 { width: 200px; }  select.form-control.w300 { width: 300px; }  select.form-control.w400 { width: 400px; }  .mt5 { margin-top: 5px; }  .mb5 { margin-bottom: 5px; }  .mr20 { margin-right: 20px; }  .ml20 { margin-left: 20px; } |

## res/common.js

**src/main/webapp/res/common.js**

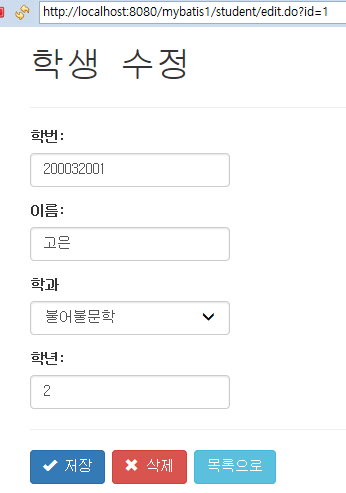
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | $(**function**() {    $("[data-url]").click(**function**() {  **var** url = $(**this**).attr("data-url");  location.href = url;  })  $("[data-confirm-delete]").click(**function**() {  **return** confirm("삭제하시겠습니까?");  })  }) |

## index.jsp

**src/main/webapp/index.jsp**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  </head>  <body>  <h1>mybatis</h1>  <a href="student/list">student/list</a> <br />  </body>  </html> |

# 실행



## 결과 소스코드

mybatis1\_src1.zip

## 참조 무결성 제약조건 위반 에러

위 화면에서 **삭제** 버튼을 클릭하자.

StudentMapper.xml 파일에 정의된 아래 SQL 명령이 실행된다.

DELETE FROM Student WHERE id = #{id}

그런데 '고은' 학생을 참조하는 register 테이블의 레코드가 존재하기 때문에,

참조 무결성 제약조건 위반 에러가 발생한다.

이클립스 console 창에 자세한 에러 메시지가 출력된다.

이 에러 메시지의 시작 부분을 자세히 살펴보자.

|  |
| --- |
| ### Error updating database. Cause: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Cannot delete or update a parent row: a foreign key constraint fails (`student1`.`register`, CONSTRAINT `FK\_Register\_Student` FOREIGN KEY (`studentId`) REFERENCES `student` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION)  ### The error may involve defaultParameterMap  ### The error occurred while setting parameters  ### SQL: DELETE FROM Student WHERE id = ?  ### Cause: com.mysql.jdbc.exceptions.jdbc4.MySQLIntegrityConstraintViolationException: Cannot delete |

**MySQLIntegrityConstraintViolationException:**

무결성 제약조건 위반 에러가 발생했다는 뜻

**Cannot delete or update a parent row: a foreign key constraint fails**

외래키 제약조건 때문에 레코드를 삭제할 수 없다는 뜻

**(`student1`.`register`, CONSTRAINT `FK\_Register\_Student` FOREIGN KEY (`studentId`) REFERENCES `student` (`id`)**

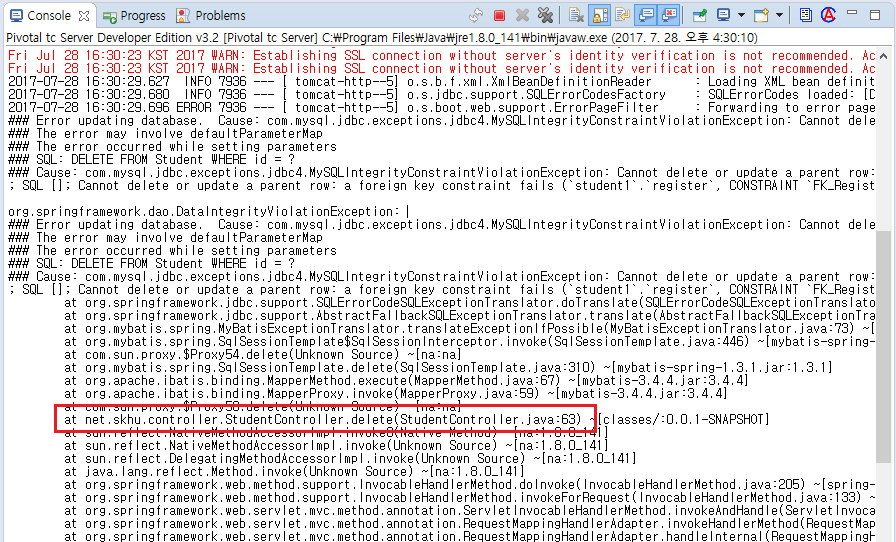
student1 데이터베이스의 register 테이블에 FK\_Register\_Student 제약조건이 있다는 뜻.

이 제약조건은 register 테이블의 studentId 필드가 외래키이고,

이 외래키가 student 테이블의 id 필드를 가르킨다는 뜻

**### SQL: DELETE FROM Student WHERE id = ?**

에러가 발생한 SQL 명령이다.



에러가 발생한 소스코드 위치를 에러 메시지에서 볼 수 있다.

StudentController.java 파일의 줄 63 에서 에러가 발생했다.

|  |
| --- |
| studentMapper.delete(id); |