mybatis 기초

2023-09-05

이승진

학습목표

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

목차

1. 배경지식

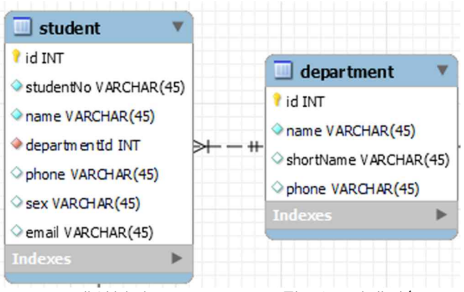
1) Auto Increment 필드 (Identity 필드)

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 값이 자동으로 부여되기 때문에,
insert나 update SQL 문에서 이 필드의 값을 저장하지 않아야 한다.

2) Referential Integrity Constraint



student 테이블의 departmentId 필드는 외래키(foreign key) 이다.
이 필드의 값은 department 테이블의 기본키인 id 필드값과 일치해야 한다.

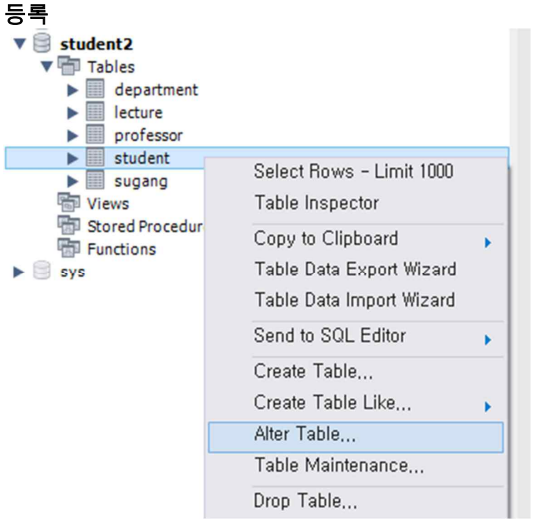
소프트웨어공학과 소속 학생들이 존재한다면, 소프트웨어공학과를 삭제할 수 없어야 한다.

department 테이블에서 소프트웨어공학과 레코드를 삭제 하려고 할 때,
  student 테이블에서 departmentId 값이 소프트웨어공학과 id 값과 일치하는 레코드가 있다면,
 삭제는 실패하고 에러가 발생한다.

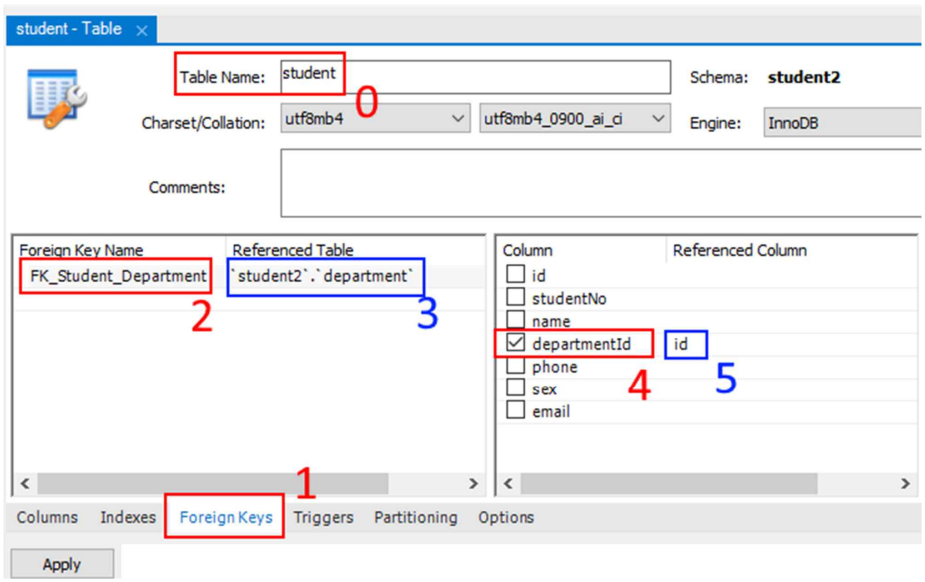
이 에러를 참조 무결성 제약조건 위반(referential intergity constraint violation) 에러 이라고 부른다.

3) 외래키 제약조건 등록/삭제

등록



mysql workbench에서 student 테이블을 우클릭하고 Alter Table 클릭



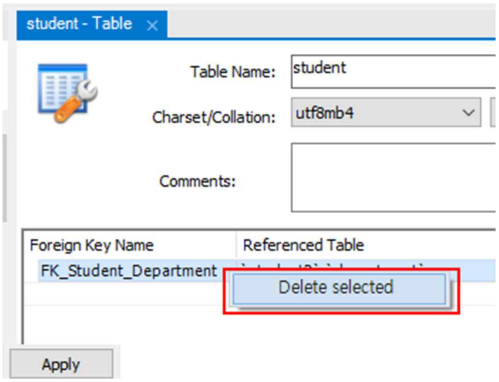
빨간색 부분은 student 테이블과 관련된 부분이고
파란색 부분은 department 테이블과 관련된 부분이다.

0) student 테이블 수정(Alter Table) 화면에서
1) Foreign Keys 탭을 선택한다. 이 탭에서 student 테이블의 외래키를 등록 삭제 할 수 있다.
2) 새로 만들 외래키의 이름을 입력한다.
   student 테이블의 departmentId 외래키가 department 테이블의 id 필드와 일치해야 하므로
   FK_Student_Department 형태의 이름을 부여하였다.
3) student2 데이터베이스의 department 테이블을 선택한다.

4) student 테이블의 외래키를 선택한다 (departmentId 필드가 외래키)
5) departmentId 필드가 가르키는 department 필드의 id 필드를 선택한다.

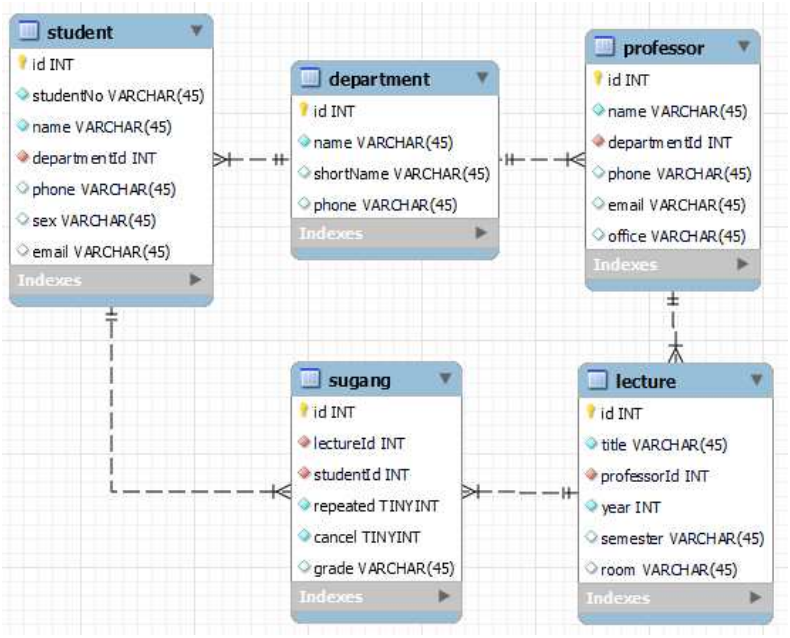
마지막으로 Apply 버튼을 클릭하면, 외래키 제약조건이 등록된다.

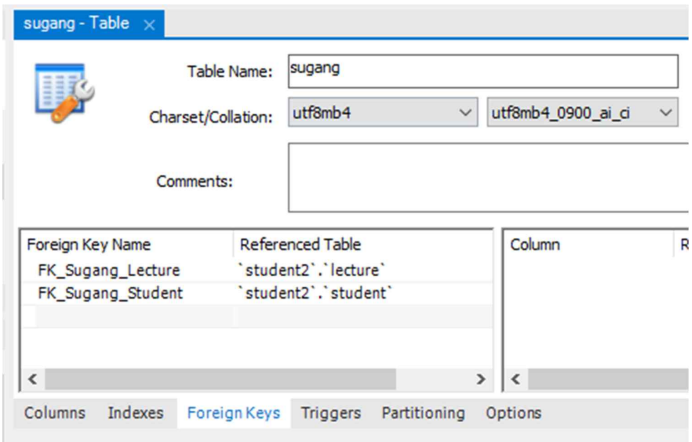
삭제



삭제할 외래키를 우클릭하고, "Delete selected" 메뉴를 클릭하고
마지막으로 Apply 버튼을 클릭하면, 외래키 제약조건이 제거된다.

4) sugang 테이블의 외래키 제약조건





sugang 테이블에는 두 개의 외래키가 있다.
  sugang 테이블의 lectudeId 외래키는 lecture 테이블의 id 기본키를 가르킨다.
  sugang 테이블의 studentId 외래키는 student 테이블의 id 기본키를 가르킨다.

sugang 테이블의 레코드 한 개는,
  studentId 외래키가 가르키는 학생이
  lectureId 외래키가 가르키는 강좌를 수강신청하고 있음을 의미한다.

그래서 sugang 테이블에 수강신청 레코드가 있는 student 레코드를 삭제하려고 하면
참조 무결성 제약조건 위반 에러(외래키 제약조건 위반 에러)가 발생한다.

5) 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.

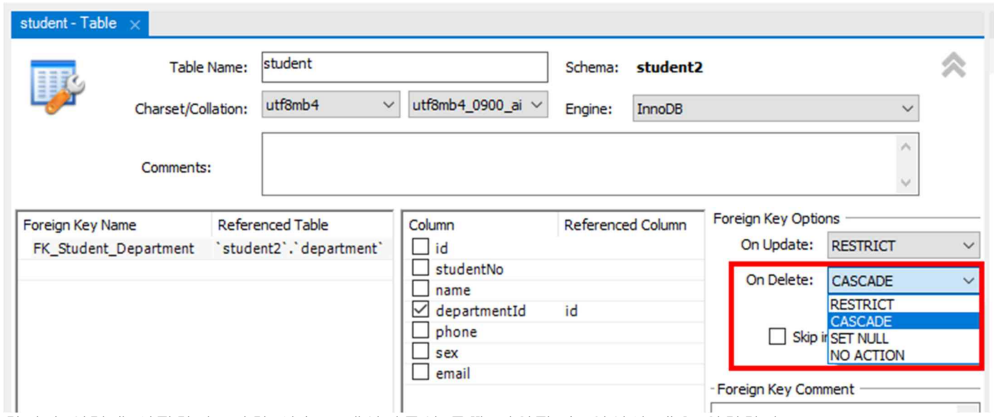
예:

```
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



하지만 이렇게 설정하면, 자칫 실수로 데이터들이 몽땅 지워질 수 있어서 매우 위험하다.

6) 삭제 관련 조언

일단 DB에 한 번 등록된 데이터는, 최대한 삭제하지 않는 것이 좋다.
그 데이터가 필요할 때가 언제가 올 수 있기 때문이다.

예를 들어 학생이 수강신청을 취소할 때, 수강 레코드를 삭제하기 보다,
취소 필드를 true로 변경하자.
정상적인 수강 레코드를 조회할 때, 취소 필드가 false인 레코드들만 조회하면 된다.

예를 들어 학생이 자퇴하거나 졸업했을 때, 학생 레코드를 삭제하기 보다,
학적상태 필드를 '자퇴'나 '졸업'으로 변경하자.
재학중인 학생 레코드를 조회할 때, '학적상태 필드값이 '재학'인 레코드들만 조회하면 된다.

7) 인덱스(index)

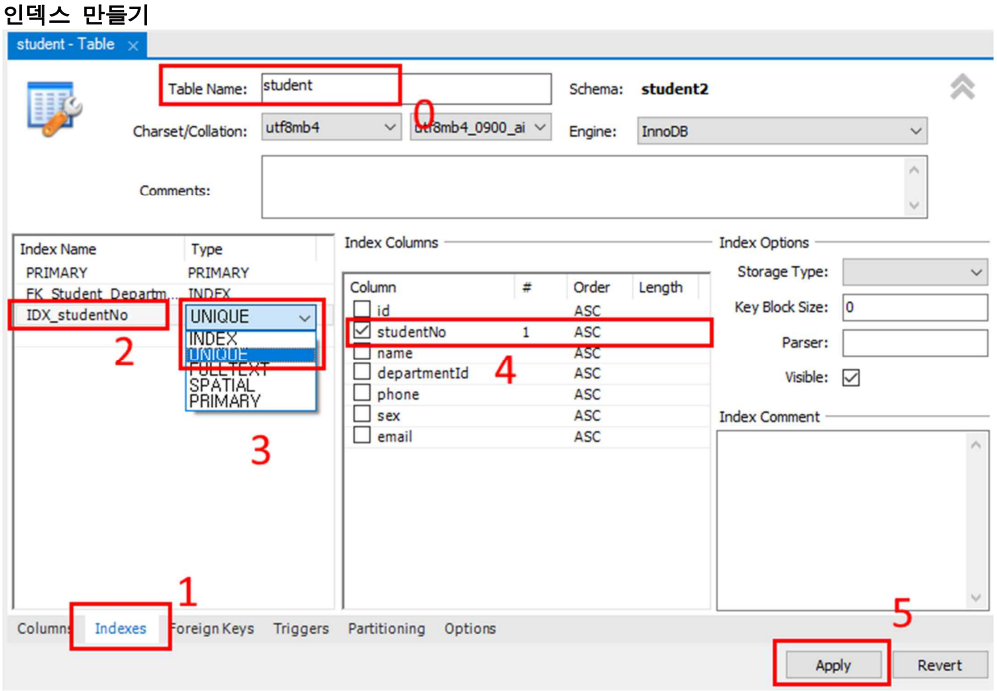
방금 위에서 조언한 방식의 조회를 빠르게 하기 위해서, 인덱스를 만들자.

수강 테이블의 취소 필드에 인덱스를 만들면, 취소 필드가 true/false인 레코드들만 빠르게 조회할 수 있다.
학적 테이블의 학적상태 필드에 인덱스를 만들면,
학적상태 필드가 재학/졸업/자퇴인 레코드들을 빠르게 조회할 수 있다.

예를 들어, 학번으로 학생을 조회하는 경우가 비교적 많다.
student 테이블의 studentNo 필드에 인덱스를 만들면, 학번으로 학생을 조회를 빠르게 할 수 있다.

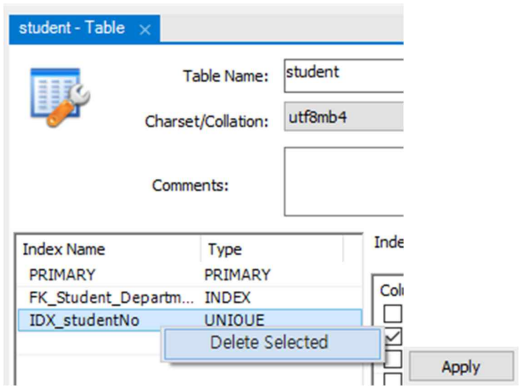
필드에 인덱스가 있고 없고에 따라 조회 속도 차이가 매우 크다.

인덱스 만들기



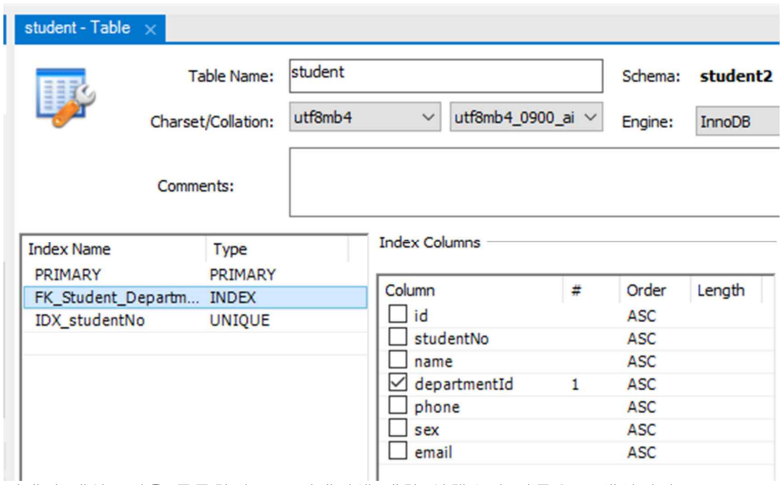
0) student 테이블 수정(Alter Table) 화면에서
1) Indexes 탭을 선택한다. 이 탭에서 인덱스를 등록/삭제할 수 있다.
2) 인덱스 이름을 입력한다. studentNo 필드에 인덱스를 만들 거라서 IDX_studentNo 이름을 부여하였다.
3) 보통 인덱스라면 INDEX 항목을 선택하면 된다.
   값의 중복이 없는 필드에 대한 인덱스라면 UNIQUE 항목을 선택하면 된다.
   학번은 값의 중복이 없어야 하기 때문에, UNIQUE 항목을 선택하자.
4) 인덱스를 만들 필드를 선택한다.
5) 마지막으로 Apply 버튼을 클릭하면, 인덱스가 등록된다.

인덱스 삭제



삭제할 인덱스를 우클릭하고, Delete selected 메뉴를 클릭한다.
마지막으로 Apply 버튼을 클릭하면 인덱스가 제거된다.

외래키 인덱스 자동 생성



외래키 제약조건을 등록하면, 그 외래키에 대한 인덱스가 자동으로 생성된다.
따라서 departmentId 외래키로 조회할 때, 빠르게 조회할 수 있게 된다.

복수 필드 인덱스

예를 들어서, 학번으로 조회하는 경우도 많고, 학생 이름으로 조회하는 경우도 많다면,
학번 필드에 대한 인덱스 한 개, 학생 이름에 대한 인덱스 또 한 개,
이렇게 두 개의 인덱스를 만드는 것이 좋다.

2. lombok

1) lombok이란?

lombok 은 getter, setter 메소드를 자동으로 생성해 주는 도구이다.
lombok 을 사용하면, getter, setter 메소드를 구현하지 않아도 되니 편하다.
equals 메소드, hashCode 메소드, toString 메소드도 자동으로 생성해 준다.

클래스에 @Data 어노테이션을 붙여주기만 하면 된다.
getter, setter, equals, hashCode, toString 메소드들이 자동 생성된다.

바이트코드 파일에 생성됨 (*.class)

그런데 소스 코드 파일이 아니고, 컴파일되어 생성되는 *.class 파일에,
위 메소드들이 자동 생성된다.

@Data 어노테이션

이 어노테이션을 클래스 앞에 붙이면,
getter, setter, equals, hashCode, toString 메소드가 자동 구현된다.
예:

```
@Data
public class Student {
   int id;
   String name;
}
```

- 모든 멤버 변수에 대해서 getter setter 가 자동으로 구현된다.
- 모든 멤버 변수 값이 같은지 비교하는 equals 메소드가 자동으로 구현된다.
- 모든 멤버 변수 값을 바탕으로 해시 값을 계산하는 hashCode 메소드가 자동으로 구현된다.
- 모든 멤버 변수 값을 문자열로 표현하는 toString 메소드가 자동으로 구현된다.

만약 자동으로 구현되는 메소드에서 제외되어야 할 멤버 변수가 있다면 다음과 같이 구현한다.
예:

```
@Data
@EqualsAndHashCode(exclude="temp")
@ToString(exclude="temp")
public class Student {
   int id;
   String name;
   int temp;
}
```

equals, hashCode, toString 메소드가 자동으로 구현될 때, temp 멤버 변수는 무시된다.

예:

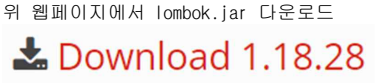
```
@Data
@EqualsAndHashCode(exclude={"temp1","temp2"})
@ToString(exclude={"temp1","temp2"})
public class Student {
   int id;
   String name;
   int temp1;
   String temp2;
}
```

equals, hashCode, toString 메소드가 자동으로 구현될 때, temp1, temp2 멤버 변수는 무시된다.

2) lombok 설치

다운로드

https://projectlombok.org/download.html
위 웹페이지에서 lombok.jar 다운로드

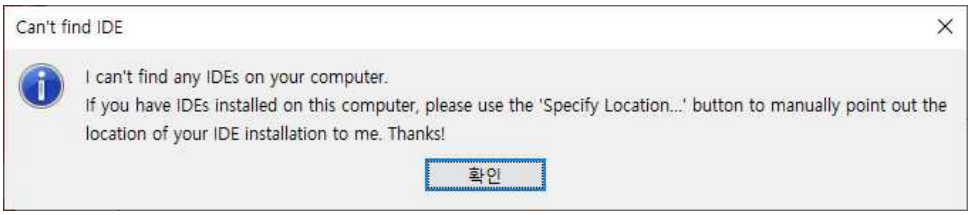


lombok.jar 실행

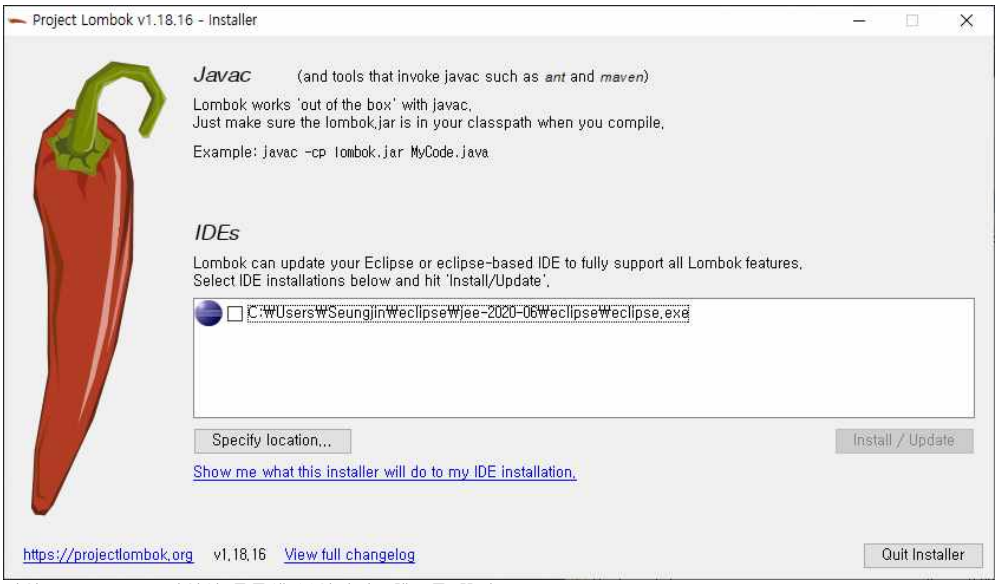
먼저 Spring tool Suite를 종료하고 설치해야 한다.

명령 프롬프트 창에서, 다운로드된 lombok.jar 파일이 있는 디렉토리로 이동하여, 다음 명령을 실행한다.

```
java -jar lombok.jar
```

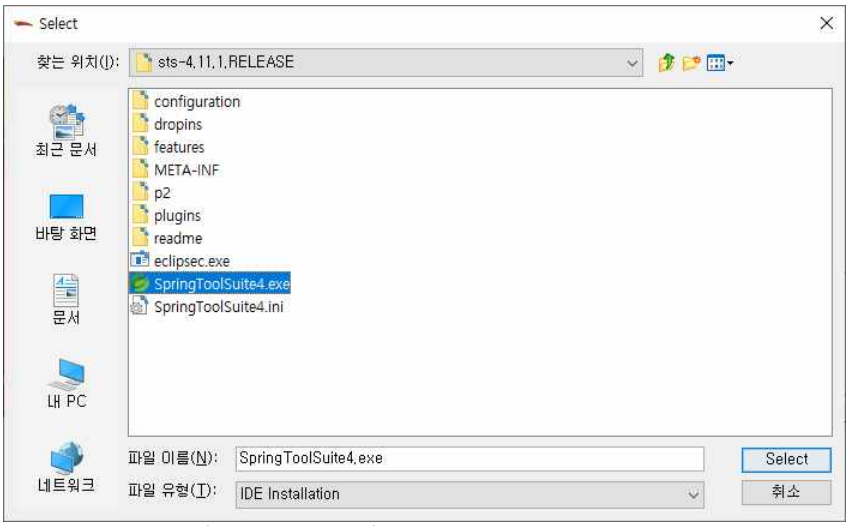


만약 위 대화 상자가 나타나면 '확인' 버튼을 클릭하여 닫는다.
위 대화 상자가 나타나지 않을 수도 있다.

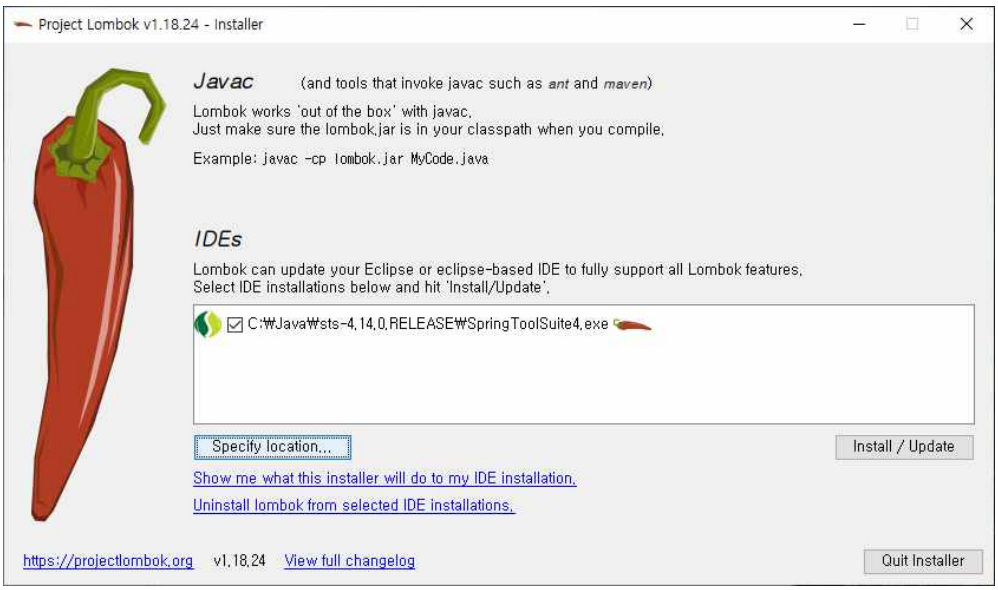


만약 eclipse.exe 파일이 목록에 보인다면, 체크를 끈다.

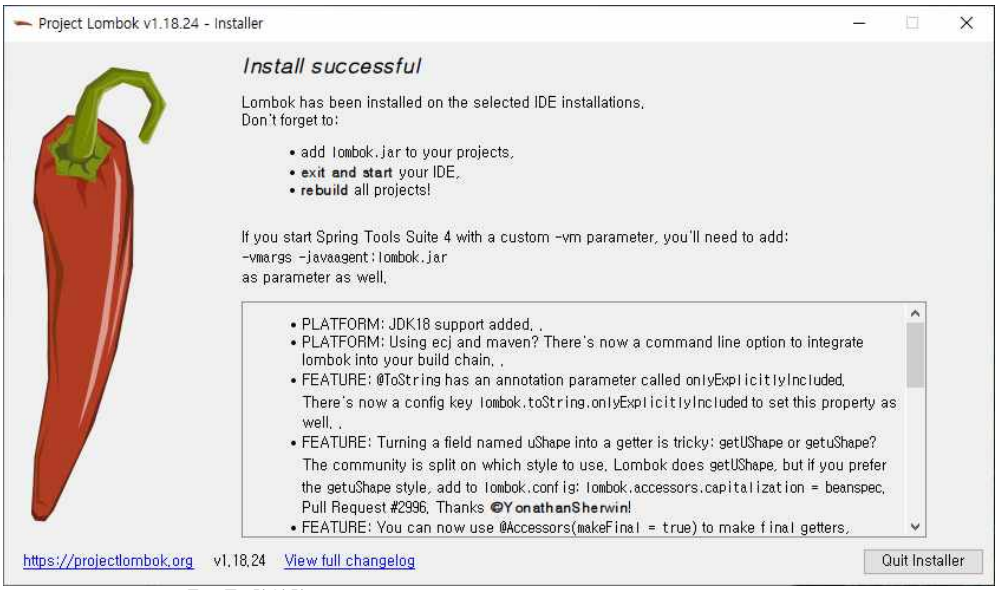
'Specify location...' 버튼을 클릭하여 SpringToolSuite4.exe 파일을 찾는다.



SpringToolSuite4.exe (Spring tool suite) 실행 파일을 찾아서 선택한다.

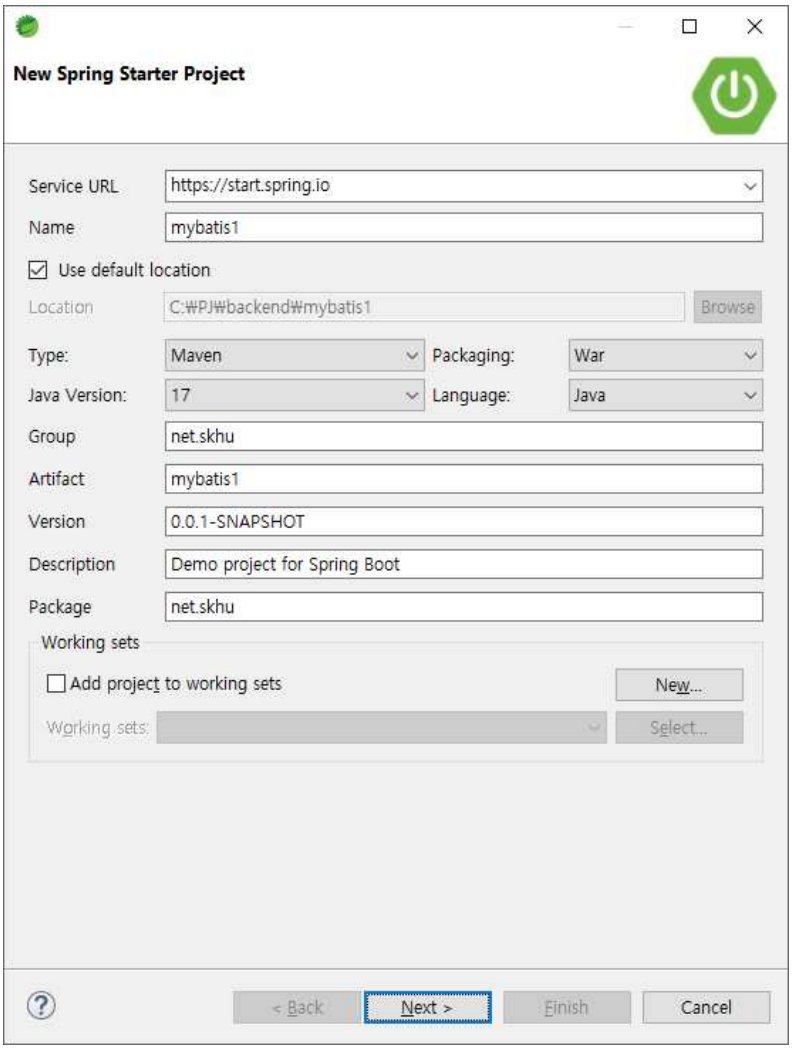


SpringToolSuite4.exe 파일이 목록에 보인다.
SpringToolSuite4.exe 파일을 체크하고, 'Install / Update' 버튼을 클릭해서 설치하자.

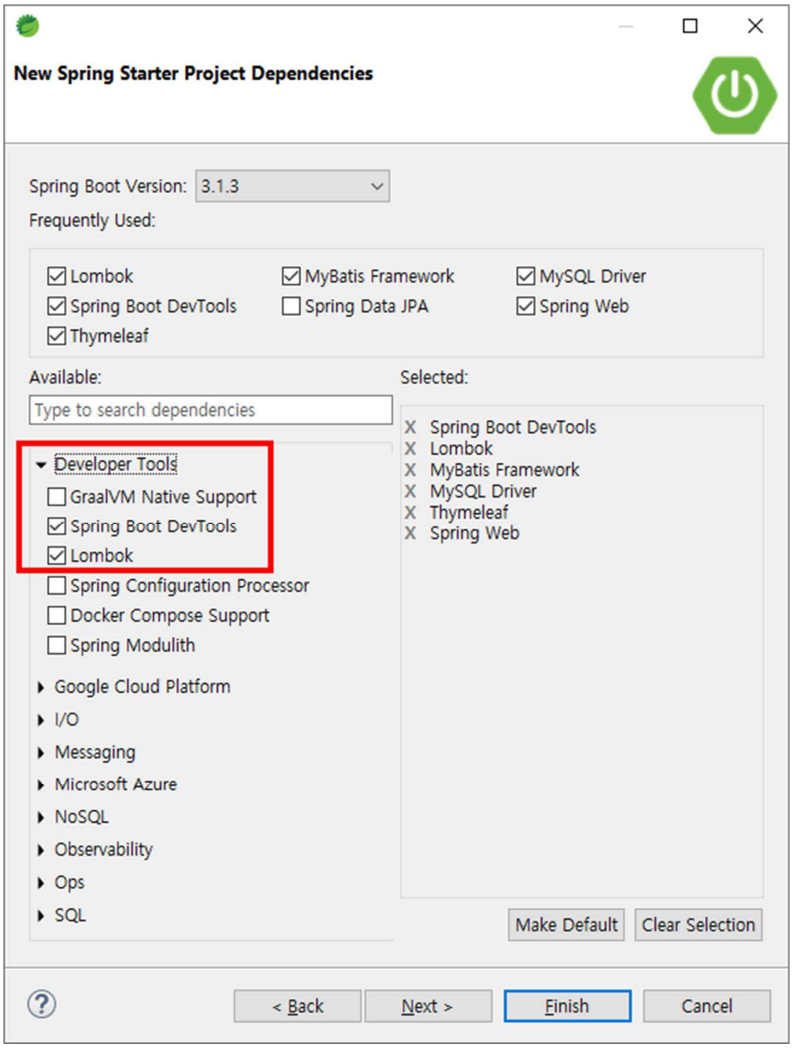


Install successful 문구를 확인하고,
Quit Installer 클릭

3. **mybatis1** 프로젝트



Name	mybatis1
Java version	17
Package	net.skhu



Spring Booot Version: 3.0.# 버전을 선택함.

체크할 항목들

Developer Tools 항목 아래
  Spring Boot DevTools
  Lombok

SQL 항목 아래
  MySQL Driver
  MyBatis Framework

Web 항목 아래
  Spring Web

Template Engines 항목 아래
  Thymeleaf

1) application.properties

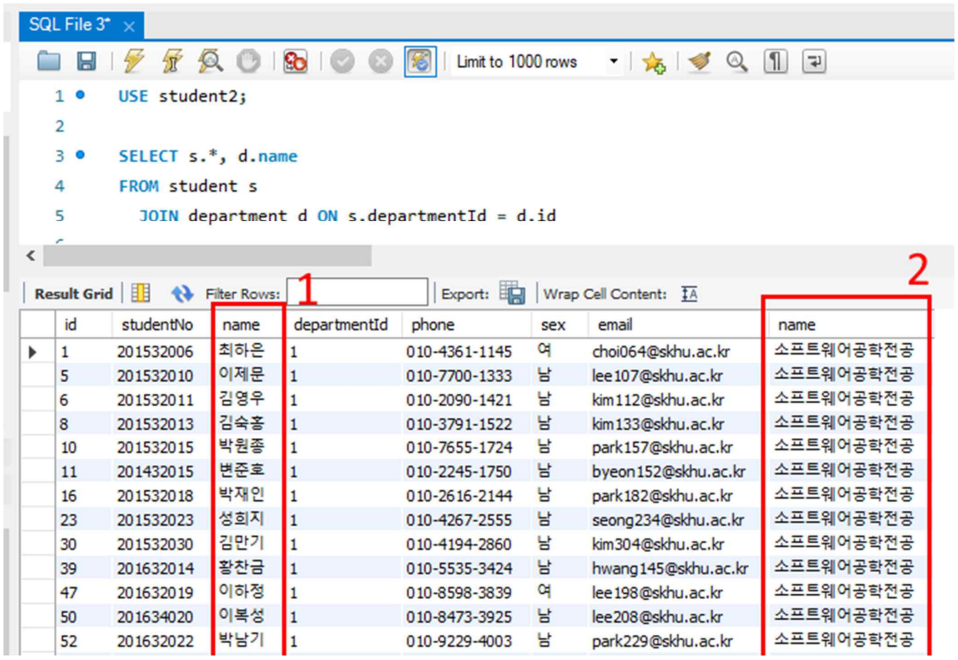
```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/student2?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul
spring.datasource.username=user1
spring.datasource.password=skhuA+4.5
server.port=8088
```

application.properties는 spring Boot 설정 파일이다.

2) 소속 학과명 조회

학생 목록을 출력할 때, 소속 학과명도 출력해야 한다.
그런데 student 테이블에 학과명 필드가 없으니,
다음과 같이 department 테이블과 조인하여 학과명 필드도 조회해야 한다.

```
SELECT s.*, d.name
FROM student s
  JOIN department d ON s.departmentId = d.id
```



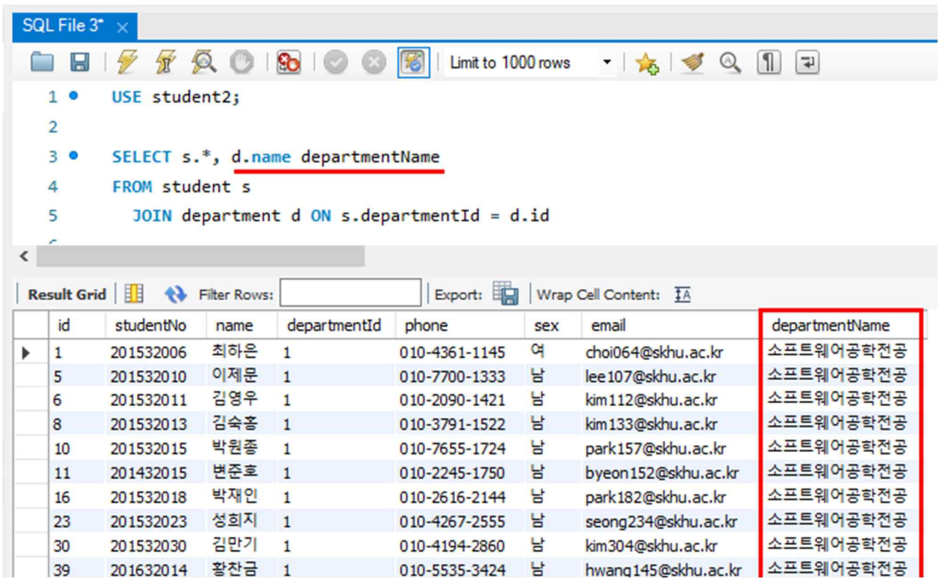
id	studentNo	name	departmentId	phone	sex	email	name
1	201532006	최하은	1	010-4361-1145	여	choi064@skhu.ac.kr	소프트웨어공학전공
5	201532010	이제문	1	010-7700-1333	남	lee107@skhu.ac.kr	소프트웨어공학전공
6	201532011	김영우	1	010-2090-1421	남	kim112@skhu.ac.kr	소프트웨어공학전공
8	201532013	김숙홍	1	010-3791-1522	남	kim133@skhu.ac.kr	소프트웨어공학전공
10	201532015	박원종	1	010-7655-1724	남	park157@skhu.ac.kr	소프트웨어공학전공
11	201432015	변준호	1	010-2245-1750	남	byeon152@skhu.ac.kr	소프트웨어공학전공
16	201532018	박재인	1	010-2616-2144	남	park182@skhu.ac.kr	소프트웨어공학전공
23	201532023	성희지	1	010-4267-2555	남	seong234@skhu.ac.kr	소프트웨어공학전공
30	201532030	김만기	1	010-4194-2860	남	kim304@skhu.ac.kr	소프트웨어공학전공
39	201632014	황찬금	1	010-5535-3424	남	hwang145@skhu.ac.kr	소프트웨어공학전공
47	201632019	이하정	1	010-8598-3839	여	lee198@skhu.ac.kr	소프트웨어공학전공
50	201634020	이복성	1	010-8473-3925	남	lee208@skhu.ac.kr	소프트웨어공학전공
52	201632022	박남기	1	010-9229-4003	남	park229@skhu.ac.kr	소프트웨어공학전공

그런데, 위 조회 결과 출력해서, name 컬럼이 두 개 이다.
  1번 name 컬럼은 student 테이블의 name 필드이다.
  2번 name 컬럼은 department 테이블의 name 필드이다.

이렇게 조회 결과 컬럼명에 중복이 있을 때, DB에서 SQL을 실행할 때는 에러가 발생하지 않지만,
조회 결과를 Java 코드가 받아올 때는 에러가 발생한다.

그래서 조회 결과 컬럼명에 중복이 없도록 다음과 같이 SQL 명령을 수정해야 한다.

```
SELECT s.*, d.name departmentName
FROM student s
  JOIN department d ON s.departmentId = d.id
```



id	studentNo	name	departmentId	phone	sex	email	departmentName
1	201532006	최하은	1	010-4361-1145	여	choi064@skhu.ac.kr	소프트웨어공학전공
5	201532010	이제문	1	010-7700-1333	남	lee107@skhu.ac.kr	소프트웨어공학전공
6	201532011	김영우	1	010-2090-1421	남	kim112@skhu.ac.kr	소프트웨어공학전공
8	201532013	김숙홍	1	010-3791-1522	남	kim133@skhu.ac.kr	소프트웨어공학전공
10	201532015	박원종	1	010-7655-1724	남	park157@skhu.ac.kr	소프트웨어공학전공
11	201432015	변준호	1	010-2245-1750	남	byeon152@skhu.ac.kr	소프트웨어공학전공
16	201532018	박재인	1	010-2616-2144	남	park182@skhu.ac.kr	소프트웨어공학전공
23	201532023	성희지	1	010-4267-2555	남	seong234@skhu.ac.kr	소프트웨어공학전공
30	201532030	김만기	1	010-4194-2860	남	kim304@skhu.ac.kr	소프트웨어공학전공
39	201632014	황찬금	1	010-5535-3424	남	hwang145@skhu.ac.kr	소프트웨어공학전공

학과명 컬럼의 제목이 departmentName 으로 수정되었다.

3) Department DTO 클래스

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

```java
package net.skhu.dto;

import lombok.Data;

@Data
public class Department {
    int id;
    String name;
    String shortName;
    String phone;
}
```

department 테이블에서 조회한 데이터를 채우기 위한 DTO 객체이다.
department 테이블의 필드명이, id, name, shortName, phone 이다.

@Data 어노테이션

@Data 어노테이션은 lombok 기능을 사용하기 위한 어노테이션이다.
@Data 어노테이션을 클래스 앞에 붙이면,
lombok에 의해서 getter, setter, equals, hashCode, toString 메소드가 자동 구현된다.

4) DepartmentMapper.java

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

```java
package net.skhu.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Select;
import net.skhu.dto.Department;

@Mapper
public interface DepartmentMapper {

    @Select("SELECT * FROM department")
    List<Department> findAll();
}
```

department 테이블에 대한 SELECT/INSERT/UPDATE/DELETE SQL 명령들을 구현하기 위한 것이
DepartmentMapper interface 이다.

class가 아니고 interface임에 주의하자.

interface 이므로, 메소드 본문을 구현하지 않는다.
이 mapper interface를 구한 자식 클래스와 각 메소드의 본문은 spring data mybatis가 자동으로 구현해 준다.

5) Student DTO 클래스

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

```
1   package net.skhu.dto;
2
3   import lombok.Data;
4
5   @Data
6   public class Student {
7       int id;
8       String studentNo;
9       String name;
10      int departmentId;
11      String phone;
12      String sex;
13      String email;
14
15      String departmentName;
16  }
```

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

student 테이블에는, id, studentNo, name, departmentId, phone, sex, email 필드만 있고,
departmentName 필드는 없다.

그런데 student 데이블에서 조회할 때, 학과명도 조회해야 하고,
학과명의 조회결과 컬럼명이 departmentName 이다.
이 조회결과 학과명 컬럼 값을 채우기 위한 속성이 departmentName 이다.

즉 DTO는 테이블의 내용을 채우기 위한 객체라기 보다는
SELECT SQL 문의 조회 결과를 채우기 위한 객체이다.

6) Student Mapper 구현

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

```java
package net.skhu.mapper;

import java.util.List;

import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Mapper;
import org.apache.ibatis.annotations.Options;
import org.apache.ibatis.annotations.Select;
import org.apache.ibatis.annotations.Update;

import net.skhu.dto.Student;

@Mapper
public interface StudentMapper {

    @Select("SELECT * FROM student WHERE id = #{id}")
    Student findOne(int id);

    @Select("""
        SELECT s.*, d.name departmentName
        FROM student s LEFT JOIN department d ON s.departmentId = d.id """)
    List<Student> findAll();

    @Insert("""
        INSERT student (studentNo, name, departmentId, phone, sex, email)
        VALUES (#{studentNo}, #{name}, #{departmentId}, #{phone}, #{sex}, #{email}) """)
    @Options(useGeneratedKeys=true, keyProperty="id")
    void insert(Student student);

    @Update("""
        UPDATE student SET
          studentNo= #{studentNo},
          name = #{name},
          departmentId = #{departmentId},
          phone = #{phone},
          sex = #{sex},
          email = #{email}
        WHERE id = #{id} """)
    void update(Student student);

    @Delete("DELETE FROM student WHERE id = #{id}")
    void delete(int id);
}
```

DB 의 student 테이블에 대한 조회, 삽입, 수정, 삭제 SQL 명령을 StudentMapper에 구현한다.

interface

StudentMapper는 클래스가 아니고 java interface임에 주의하자.
interface의 메소드는 강제로 public abstract 이다. 그래서 public을 붙이지 않아도 된다.

findOne

```
@Select("SELECT * FROM student WHERE id = #{id}")
Student findOne(int id);
```

id 파라미터 변수의 값이, SQL 문장의 #{id} 부분에 채워진 후, SQL 명령이 DB에서 실행된다.

이 메소드의 리턴값이 Student 임에 주목하자.
"SELECT * FROM student WHERE id = #{id}" SQL 명령이 조회한 데이터를 Student 객체에 채워서 리턴한다.
이때, 조회 결과 컬럼 이름과, Student 객체의 속성 이름이 일치해야 한다.
일치하는 이름이 없는 데이터들은 채워지지 않고 무시된다. 에러 발생 안함.
대소문자까지 일치해야 한다.

findAll

```
@Select("""
    SELECT s.*, d.name departmentName
    FROM student s LEFT JOIN department d ON s.departmentId = d.id """ )
List<Student> findAll();
```

SQL 명령이 긴 편이기 때문에 여러줄 문자열 문법을 사용하여 구현하였다.
이 문법은 Java 15 이상 버전에서만 지원된다.

이 메소드의 리턴값이 List<Student> 임에 주목하자.

위 SQL 명령이 조회한 레코드 각각을 Student 객체에 채우고,
그렇게 데이터가 채워진 Student 객체들을 List 객체에 채워서 리턴한다.

insert

```
@Insert("""
    INSERT student (studentNo, name, departmentId, phone, sex, email)
    VALUES (#{studentNo}, #{name}, #{departmentId}, #{phone}, #{sex}, #{email}) """ )
@Options(useGeneratedKeys=true, keyProperty="id")
void insert(Student student);
```

student 객체의 속성값이, INSERT SQL 명령의 #{...} 부분에 채워진 후,
그렇게 값이 채워진 SQL 명령이 DB에서 실행된다.

Java 객체의 속성명과, #{...} 부분의 이름이 일치해야 속성값이 채워질 수 있다.
#{...} 부분의 이름과 일치하는 Java 객체의 속성이 없을 경우에, 에러가 발생한다.

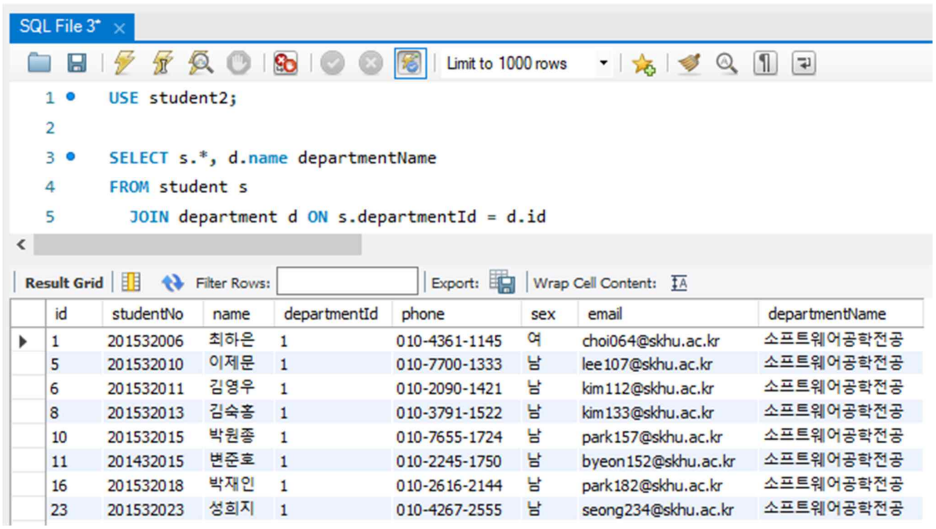
@Options(useGeneratedKeys=true, keyProperty="id")
INSERT할 테이블의 기본키(primary key) 필드 이름이 "id" 이고,
이 필드의 값은 새 레코드가 INSERT될 때, DB에서 자동으로 생성된다는 선언이다. (Auto Increment 필드)

메모 포함[혬남1]: get메소드로 읽고, 채울때는 set메소드를 통해 채운다.

조회 결과 컬럼명 일치

SELECT SQL 명령의 조회 결과가 Java 객체에 자동으로 채워질 때,
조회 결과 컬럼명과 Java 객체의 set 메소드 이름이 일치해야 한다.

조회 결과의 예



조회 결과 컬럼명	자바 객체의 set 메소드 이름
id	setId
studentNo	setStudentNo
name	setName
departmentId	setDepartmentId
phone	setPhone
sex	setSex
email	setEmail
departmentName	setDepartmentName

Student DTO 클래스의 setter 메소드들은 lombok에 의해 자동 구현된다.

메모 포함[혬남2]: 필기시험 가능성

mybatis 파라미터: 파라미터 한 개

StudentMapper.java

```
@Select("SELECT * FROM student WHERE id = #{id}")
Student findOne(int id);

@Delete("DELETE FROM student WHERE id = #{id}")
void delete(int id);
```

#{id} 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 한 개이고,
파라미터 타입이 int, long, float, double, boolean 등 기본 자료형이거나
String, Date, Time, Timestamp 클래스 중 하나이어야 한다는 점이다.

mybatis 파라미터: 파라미터 여러 개

StudentMapper.java

```
@Insert("""
    INSERT student (studentNo, name, departmentId, phone, sex, email)
    VALUES (#{studentNo}, #{name}, #{departmentId}, #{phone}, #{sex}, #{email}) """)
@Options(useGeneratedKeys=true, keyProperty="id")
void insert(Student student);

@Update("""
    UPDATE student SET
      studentNo= #{studentNo},
      name = #{name},
      departmentId = #{departmentId},
      phone = #{phone},
      sex = #{sex},
      email = #{email}
    WHERE id = #{id} """)
void update(Student student);
```

#{...} 부분이 mybatis 파라미터이다.

여기서 Java 파라미터 변수 이름은 중요하지 않다.
중요한 것은 Java 파라미터 변수의 타입이 Java 클래스이어야 하고
이 클래스의 get 메소드 이름과 mybatis 파라미터의 이름이 일치해야 한다는 점이다.

Student 클래스의 get 메소드	mybatis 파라미터
getId	#{id}
getStudentNo	#{studentNo}
getName	#{name}
getDepartmentId	#{departmentId}
getPhone	#{phone}
getSex	#{sex}
getEmail	#{email}

Student DTO 클래스의 getter 메소드들은 lombok에 의해 자동 구현된다.

메모 포함[헴남3]: 필기시험 가능성

콤마 주의

```
@Update("""
    UPDATE student SET
      studentNo= #{studentNo},
      name = #{name},
      departmentId = #{departmentId},
      phone = #{phone},
      sex = #{sex},
      email = #{email}
    WHERE id = #{id} """)
void update(Student student);
```

UPDATE SQL 명령에서 콤마(,) 문자 부분에 주의하자.
마지막 #{email} 뒤에는 콤마 문자가 없어야 한다.

7) 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("""
    INSERT student (studentNo, name, departmentId, phone, sex, email)
    VALUES (#{studentNo}, #{name}, #{departmentId}, #{phone}, #{sex}, #{email}) """ )
@Options(useGeneratedKeys=true, keyProperty="id")
void insert(Student student);
```

@Options(useGeneratedKeys=true, keyProperty="id")
INSERT할 테이블의 기본키(primary key) 필드 이름이 "id" 이고,
이 필드의 값은 새 레코드가 INSERT될 때, DB에서 자동으로 생성된다는 선언이다. (Auto Increment 필드)

새 INSERT가 완료되면 그렇게 자동으로 생성된 id 필드 값이, student 객체의 id 속성에 채워져 있다.
그래서 방금 INSERT된 레코드의 id 필드 값을 알 수 있게 된다.

예:

```
1   Student student = new Student();
2   student.setStudentNo("201132091");
3   student.setName("홍길동");
4   student.setDepartmentId(1);
5   student.setPhone("010-111-1111");
6   student.setSex("남");
7   student.setPhone("h@skhu.ac.kr");
8
9   System.out.println(student.getId());
10  studentMapper.insert(student);
11  System.out.println(student.getId());
```

줄7에서 출력되는 id 속성값은 0 이다.
Student 객체의 id 속성에 아직 아무것도 대입되지 않았기 때문이다.

줄8에서 새 레코드가 INSERT 된 후, 그 새 레코드에 자동으로 부여된 id 필드의 값이
student 객체의 id 속성에 채워진다.
이렇게 채워진 값이 줄9에서 출력된다.
이 값은 당연히 0 이 아니다.

/// 이부분 다시 읽어보기 바람 ~

8) Student Controller 구현

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

```java
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.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
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;

    @GetMapping("list")
    public String list(Model model) {
        List<Student> students = studentMapper.findAll();
        model.addAttribute("students", students);
        return "student/list";
    }

    @GetMapping("create")
    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";
    }

    @PostMapping("create")
    public String create(Model model, Student student) {
        studentMapper.insert(student);
        return "redirect:list";
    }

    @GetMapping("edit")
    public String edit(Model model, int id) {
        Student student = studentMapper.findOne(id);
        List<Department> departments = departmentMapper.findAll();
        model.addAttribute("student", student);
        model.addAttribute("departments", departments);
        return "student/edit";
    }

    @PostMapping("edit")
    public String edit(Model model, Student student) {
        studentMapper.update(student);
        return "redirect:list";
    }

    @GetMapping("delete")
    public String delete(Model model, int id) {
        studentMapper.delete(id);
        return "redirect:list";
    }
}
```

메모 포함[혬남4]: 필기시험 가능성
Student이름 값 Student객체의 목록이 전달된다.
호출하기 위한 uurl은 GetMapping에 적힌 url
Return 에 있는 것은 view의 이름 이다.

```
@Autowired StudentMapper studentMapper;
```

StudentMapper interface의 자식 클래스와 메소드를 spring이 자동으로 구현해 주고
그렇게 구현된 클래스의 객체를 자동 생성해 주고,
그 객체를 studentMapper 멤버 변수에 대입해준다.

9) 정적 컨텐츠

src/main/resources/static/common.css

```
div.container { width: 800px; margin: 10px auto; font-size: 10pt; }

.btn { padding: 0.4em 1em; border: 1px solid gray;
       border-radius: 0.5em; background: linear-gradient(#fff, #ddd);
       text-decoration: none; color: black;
       display: inline-block; }
.btn:active {
      -ms-transform: translateY(2px);
      -webkit-transform: translateY(2px);
      transform: translateY(2px);
      background: #ccc; }

table.list { border-collapse: collapse; width: 100%; }
table.list td { padding: 4px; border: 1px solid gray; }
table.list th { padding: 4px; border: 1px solid gray;  background-color: #eee; }

input { padding: 4px; }
select { padding: 4px; }
```

10) student/list 뷰 구현

src/main/resources/templates/student/list.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="/common.css" />
  <style>
    a.btn { float: right; margin-top: -40px; }
  </style>
</head>
<body>
<div class="container">
  <h1>학생목록</h1>
  <a href="create" class="btn">학생등록</a>
  <table class="list">
    <thead>
      <tr>
        <th>id</th>
        <th>학번</th>
        <th>이름</th>
        <th>학과</th>
        <th>전화</th>
        <th>성별</th>
        <th>이메일</th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="st : ${ students }">
        <td th:text="${ st.id }"></td>
        <td><a th:text="${ st.studentNo }" th:href="${ 'edit?id=' + st.id }"></a></td>
        <td th:text="${ st.name }"></td>
        <td th:text="${ st.departmentName }"></td>
        <td th:text="${ st.phone }"></td>
        <td th:text="${ st.sex }"></td>
        <td th:text="${ st.email }"></td>
      </tr>
    </tbody>
  </table>
</div>
</body>
</html>
```

```
<tr th:each="st : ${ students }">
  ...
</tr>
```

students는 학생 목록이다. (뷰에 전달된 model attribute 데이터)

model.addAttribute("students", ...);

이 학생 목록의 학생 각각을 st 변수에 대입하며, <tr> 태그가 반복 출력된다.

```
<a th:text="${ st.studentNo }" th:href="${ 'edit?id=' + st.id }"></a>
```

th:text="값"

값 부분은 이 태그 사이트 텍스트로 출력된다.

th:href="URL"

이 태그의 href="URL" 애트리뷰트로 출력된다.

예를 들어 st 변수에 대입된 학생 객체가 id는 3 이고, studentNo는 '201014199' 이라면,

위 소스코드에 의해서 다음과 같은 <a> 태그가 출력된다.

```
<a href="edit?id=3">201014199</a>
```

11) student/edit 뷰 구현

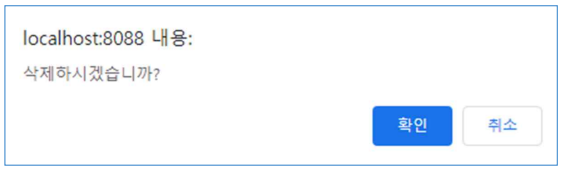
src/main/resources/templates/student/edit.html

```
<!DOCTYPE html>
<html lang="ko" xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="utf-8">
  <link rel="stylesheet" type="text/css" href="/common.css" />
  <style>
    form { width: 600px; margin: auto; padding: 5px 20px; box-shadow: 2px 2px 5px gray; }
    td { min-width: 100; padding: 5px; }
    td:nth-child(1) { text-align: right; }
    button { margin: 5px 0 20px 20px; }
  </style>
</head>
<body>
<div class="container">
 <form method="post" th:object="${student}">
    <h1 th:text="${student.id > 0 ? '학생 수정' : '학생 등록'}"></h1>
    <table>
      <tr>
        <td>학번:</td>
        <td><input type="text" th:field="*{studentNo}" /></td>
      </tr>
      <tr>
        <td>이름:</td>
        <td><input type="text" th:field="*{name}" /></td>
      </tr>
      <tr>
        <td>학과:</td>
        <td><select th:field="*{departmentId}">
              <option th:each="dt : ${ departments }"
                      th:value="${ dt.id }" th:text="${ dt.name }">
              </option>
            </select>
      </tr>
      <tr>
        <td>전화:</td>
        <td><input type="text" th:field="*{phone}" /></td>
      </tr>
      <tr>
        <td>성별:</td>
        <td><input type="text" th:field="*{sex}" /></td>
      </tr>
      <tr>
        <td>이메일:</td>
        <td><input type="text" th:field="*{email}" /></td>
      </tr>
    </table>
    <hr />
    <div>
      <button type="submit" class="btn">저장</button>
      <a th:if="${ student.id > 0 }" th:href="${ 'delete?id=' + student.id }"
         class="btn" onclick="return confirm('삭제하시겠습니까?')">삭제</a>
      <a href="list" class="btn">목록으로</a>
    </div>
  </form>
</div>
</body>
</html>
```

student는 학생 객체이다. (뷰에 전달된 model attribute 데이터)
연두색으로 칠한 부분은, student 학생 객체의 속성이다.

```
<h1 th:text="${student.id > 0 ? '학생 수정' : '학생 등록'}"></h1>
  student는 학생 객체의
  id 속성 값이 0 보다 크면, '학생 수정' 문자열이 <h1> 태그에 출력되고,
  그렇지 않으면 '학생 등록' 문자열이 출력된다.


<form method="post" th:object="${student}">
  이 입력 폼에서 편집할 데이터 객체는, student 이름의 객체이다. (뷰에 전달된 model attribute 데이터)


<input type="text" th:field="*{studentNo}" />
  위 확장 태그의 기능은 다음과 같다.
  - 입력 폼이 출력될 때, student 객체의 studentNo 속성값이 input 태그의 value에 채워진다.
  - 이 input 태그의 name 값은 studentNo 이다.
    따라서 저장(submit) 버튼이 클릭되고 폼에 입력된 데이터가 서버에 전송될 때,
    이 input 태그에 입력된 데이터가 request parameter로 서버에 전송된다.
    이 request parameter의 이름은 studentNo 이다.
 즉 위 소스코드에 의해서 출력되는 html 태그는 다음과 같다.
   <input type="text" name="studentNo" value="" />
   ${student.studentNo} 값이 null이기 때문에 value 값이 빈 문자열이다.

<select th:field="*{departmentId}">
  <option th:each="dt : ${ departments }"
          th:value="${ dt.id }" th:text="${ dt.name }">
  </option>
</select>
  이 확장 태그의 기능은 다음과 같다.
  - select 태그와 option 태그들이 출력된다.
  - departments 객체 목록에 들어있는 Department 객체 각각이 dt 변수에 대입되어,
    option 태그가 반복 출력된다.
  - 반복 출력되는 option 태그의 value는 Department 객체의 id 값이고,
                  option 태그의 텍스트를 Department 객체의 name 값이다.
  - student 객체의 departmentId 속성값과 value 값이 일치하는 option 태그가 selected 된다.
  - select 태그의 name 값은 departmentId 이다.
    따라서 저장(submit) 버튼이 클릭되고 폼에 입력된 데이터가 서버에 전송될 때,
    이 select 태그에서 선택된 option 태그의 value 값이 request parameter로 서버에 전송된다.
    이 request parameter의 이름은 departmentId 이다.
 즉 위 소스코드에 의해서 출력되는 html 태그는 다음과 같다.
     <select name="departmentId">
       <option value="1">소프트웨어공학과</option>
       <option value="2">컴퓨터공학과</option>
       <option value="3">정보통신공학과</option>
       <option value="4">글로컬IT공학과</option>
     </select>
  그런데 ${student.departmentId} 값이 0 이다. 이 값과 value가 일치하는 항목이 없기 때문에
  selected 가 붙은 option 태그가 없다.
  만약 ${student.departmentId} 값이 1 이라면, 이 값과 value가 일치하는 항목이 있으니
  다음과 같이 출력된다.
     <select name="departmentId">
       <option value="1" selected>소프트웨어공학과</option>
       <option value="2">컴퓨터공학과</option>
       <option value="3">정보통신공학과</option>
       <option value="4">글로컬IT공학과</option>
     </select>

<a th:if="${ student.id > 0 }" th:href="${ 'delete?id=' + student.id }"
   class="btn" onclick="return confirm('삭제하시겠습니까?')">삭제</a>

   th:if="${ student.id > 0 }"
         student.id > 0 조건식이 false 이면 이 <a> 태그가 출력되지 않는다.

   th:href="${ 'delete?id=' + student.id }"
         예를 들어 student 객체의 id 값이 3 이라면 다음과 같은 태그가 출력된다.
         <a href="delete?id=3" class="btn" onclick="return confirm('삭제하시겠습니까?')">삭제</a>
```

이 삭제 버튼을 클릭하면 onclick="return confirm('삭제하시겠습니까?')" 부분이 실행된다.



확인 버튼을 클릭하면, a 태그의 href URL이 서버에 요청된다.
취소 버튼을 클릭하면, a 태그 클릭이 취소되어 아무일도 일어나지 않는다.

삭제 기능과 GET 요청

삭제 버튼을 위와 같이 a 태그로 구현하면,
a 태그를 클릭하면 GET 요청이 서버에 전송된다.
이것은 바람직하지 않다.

GET 요청은 자료 조회 요청에서만 사용해야 한다.
삭제 기능은 GET이 아닌, POST 요청이나 DELETE 요청으로 구현해야 한다.
이렇게 바람직 하게 구현하는 기법은, 나중에 pagination 예제에서 살펴보자.

조회가 아닌 수정/삭제 요청을 GET 요청으로 구현하는 것이 바람직하지 않은 이유
  GET 요청을 자료 조회 요청이기 때문에,
  웹브라우저 캐시나 웹 서버 근처 캐시 서버가 처리하여
  캐시된 응답 화면만 전달할뿐 그 요청이 실제 서버까지 전달되지 않을 수 있다.

12) 실행

http://localhost:8088/student/list

id	학번	이름	학과	전화	성별	이메일
3	201532008	나철진	정보통신공학과	010-8703-1239	남	na088@skhu.ac.kr
4	201532009	이익수	글로컬IT공학과	010-7875-1251	남	lee097@skhu.ac.kr
7	201532012	주한요	정보통신공학과	010-4624-1467	남	joo124@skhu.ac.kr
9	201532014	홍영수	글로컬IT공학과	010-2897-1525	남	hong142@skhu.ac.kr
12	201532016	고희정	정보통신공학과	010-5691-1943	여	ko165@skhu.ac.kr
13	201432016	신철대	글로컬IT공학과	010-3221-1956	남	sin163@skhu.ac.kr
14	201532017	서윤정	정보통신공학과	010-4310-1965	여	seo174@skhu.ac.kr
15	201432017	오화순	정보통신공학과	010-8527-2048	여	oh178@skhu.ac.kr
17	201532019	최현복	컴퓨터공학과	010-7312-2171	남	choi197@skhu.ac.kr
18	201432019	임봉영	컴퓨터공학과	010-7475-2192	남	lim197@skhu.ac.kr
19	201532020	장영만	컴퓨터공학과	010-1527-2270	남	jang201@skhu.ac.kr
20	201432020	박주용	글로컬IT공학과	010-1640-2396	남	park201@skhu.ac.kr
21	201532021	황하지	컴퓨터공학과	010-6682-2463	남	hwang216@skhu.ac.kr
22	201532022	우환치	글로컬IT공학과	010-3951-2464	남	yu223@skhu.ac.kr

위 화면에서 나철진 학생 클릭

위 화면에서 삭제 버튼을 클릭하면, 참조 무결성 제약조건 위반 에러가 발생한다.

4. 연습 문제

professor 테이블에 대해서
목록, 등록, 수정, 삭제 기능을 구현하라.