[실습 1] Spring JPA 로 SpringBootLab6 N:M 관계를 구현

Spring Boot JPA(또는 Hibernate JPA)?

JPA 학습 단계

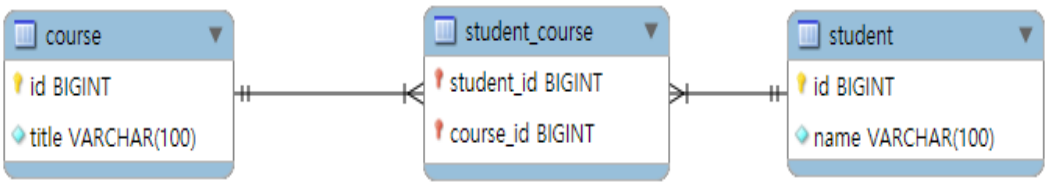
단계	내용
[1] 엔티티 선언	@Entity, @Id, @GeneratedValue 로 엔티티와 기본키 정의
[2] 관계 매핑	@ManyToOne, @OneToMany 등 관계 설정 (Fetch: LAZY, EAGER)
[3] 영속성 컨텍스트	1 차 캐시, Dirty Checking 으로 엔티티 상태 관리
[4] Repository	JpaRepository 상속 및 메서드명 기반 쿼리 자동 생성
[5] JPQL	@Query 로 복잡한 쿼리 작성
[6] 테스트	@DataJpaTest, @Transactional 로 Repository 단위 테스트

연관 관계 매핑

단방향/양방향 관계

@	설명	샘플 예제 (간단하게)
@ManyToOne	다대일 관계 (ex. 여러 사원이 한 부서에 소속)	@ManyToOne private Dept dept;
@OneToMany	일대다 관계 (ex. 한 부서에 여러 사원)	@OneToMany(mappedBy="dept") private List<Emp> emps;
@OneToOne	일대일 관계 (ex. 사원과 사원정보가 1:1)	@OneToOne private EmpInfo empInfo;
@ManyToMany	다대다 관계 (ex. 학생-강의)	@ManyToMany private List<Course> courses;

[실습] SpringLab06 (N:M) JPA



1) 디렉토리 구조

```
SpringBootLab6 /
└─── src/
    └─── main/
        ├─── java/
        |    └─── com/sec01/
        |         ├─── Sec01Application.java
        |         ├─── controller/
        |         |    └─── StudentCourseController.java
        |         ├─── service/
        |         |    └─── StudentCourseService.java
        |         ├─── repository/
        |         |    ├─── StudentRepository.java
        |         |    └─── CourseRepository.java
        |         └─── entity/
        |              ├─── Student.java
        |              └─── Course.java
        └─── resources/
             ├─── application.yml
             ├─── templates/
             |    ├─── students.html
             |    └─── courses.html
```

2) Application.yml

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/spring_lab06
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update # 또는 create, create-drop, none
    show-sql: true # JPA 가 생성하는 SQL 을 콘솔에 출력
    properties:
      hibernate:
        format_sql: true # SQL 포맷팅
logging:
  level:
    org:
      hibernate:
        SQL: DEBUG # 실행되는 SQL 쿼리 로깅
        type:
          descriptor:
            sql: TRACE # SQL 파라미터 로깅
```

Entity

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Student {

    @Id
    @GeneratedValue
    private Long id;

    private String name;

    @ManyToMany
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private List<Course> courses;
}
```

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Course {

    @Id
    @GeneratedValue
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private List<Student> students;
}
```

Repository

```
@Repository
public interface CourseRepository extends JpaRepository<Course, Long> {}
```

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {}
```

StudentCourseService

메서드명 : 반환타입	기능 설명	사용 Repository
getAllStudents() : List<Student>	모든 학생 목록을 조회한다.	StudentRepository
getAllCourses() : List<Course>	모든 과목 목록을 조회한다.	CourseRepository
addStudent(String, String) : void	학생과 과목을 동시에 추가하고, 중간테이블까지 함께 INSERT 한다.	StudentRepository (save) (JPA 가 중간테이블 insert 까지 처리)
addCourse(String) : void	과목만 단독으로 추가한다.	CourseRepository
deleteStudent(Long) : void	학생을 삭제한다.	StudentRepository
deleteCourse(Long) : void	과목을 삭제한다.	CourseRepository

StudentCourseController

메서드명 : 반환타입	HTTP 메서드	경로 (URL)	기능 설명
showStudents(Model) : String	GET	/students	모든 학생 목록과 과목 목록을 함께 조회한다.
showCourses(Model) : String	GET	/courses	모든 과목 목록을 조회한다.
addStudent(String, String) : String	POST	/students	학생과 과목을 동시에 추가한다.
addCourse(String) : String	POST	/courses	과목을 단독으로 추가한다.
deleteStudent(Long) : String	GET	/students/delete/{id}	특정 학생을 삭제한다.
deleteCourse(Long) : String	GET	/courses/delete/{id}	특정 과목을 삭제한다.

Template 구현 요구사항

1 메인 화면 (/students)

- 학생 목록을 테이블로 출력한다.
  ➜ 학생의 **이름, 번호, 수강 중인 과목명**을 출력할 것.
- 학생 추가 폼을 구현한다.
  ➜ 학생 이름, 과목명을 입력받아 POST /students 로 전송.

2. **목록**

- 전체 과목 목록을 아래쪽에 테이블로 출력한다.
  ➜ 과목의 **번호, 과목명**을 출력할 것.
- 과목 추가 폼을 구현한다.
  ➜ 과목명만 입력받아 POST /courses 로 전송.

3. 삭제 능

- 각 학생/과목 옆에 **삭제 버튼**을 구현하고,
  ➜ 클릭 시 각각의 /students/delete/{id}, /courses/delete/{id} 경로로 이동하여 삭제되도록 처리할 것.

[추가 실습 01] Controller 테스트를 구현한다

Controller

```
@Test
void testShowAllPages() throws Exception {
    // 테스트용 학생/강의 저장
    Student student = new Student();
    student.setName("홍길동");
    studentRepository.save(student);

    Course course = new Course();
    course.setTitle("자바 프로그래밍");
    courseRepository.save(course);

    // 학생 목록 페이지 테스트
    mockMvc.perform(get("/students"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("홍길동")));

    // 강의 목록 페이지 테스트
    mockMvc.perform(get("/courses"))
            .andExpect(status().isOk())
            .andExpect(content().string(containsString("자바 프로그래밍")));
}
```

[실습 02] Querydsl 를 사용해보자

Querydsl 라이브러리 의존성

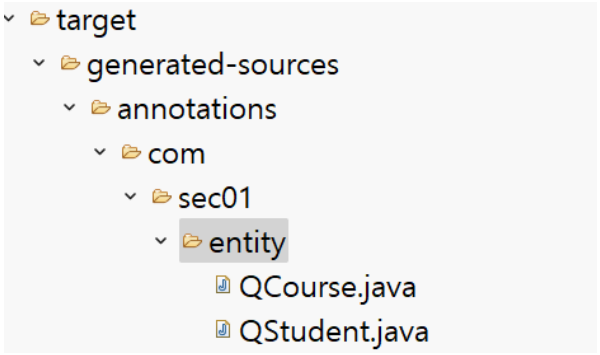
```
  <dependency>
                  <groupId>com.querydsl</groupId>
                  <artifactId>querydsl-jpa</artifactId>
                  <version>5.1.0</version>
                  <classifier>jakarta</classifier>
</dependency>

<dependency>
                  <groupId>com.querydsl</groupId>
                  <artifactId>querydsl-apt</artifactId>
                  <version>5.1.0</version>
                  <classifier>jakarta</classifier>
                  <scope>provided</scope>
</dependency>
```

Querydsl Q 타입 자동 생성 설정 <plugin> </plugin>추가

```
 <configuration>
     <source>${java.version}</source>
     <target>${java.version}</target>
     <annotationProcessorPaths>
         <path>
             <groupId>org.projectlombok</groupId>
             <artifactId>lombok</artifactId>
             <version>1.18.32</version>
         </path>
         <path>
             <groupId>com.querydsl</groupId>
             <artifactId>querydsl-apt</artifactId>
             <version>5.1.0</version>
             <classifier>jakarta</classifier>
         </path>
         <path>
             <groupId>jakarta.persistence</groupId>
             <artifactId>jakarta.persistence-api</artifactId>
             <version>3.1.0</version>
         </path>
     </annotationProcessorPaths>
 </configuration>
```

Q 타입 클래스 생성하기

mvn clean compile 후 생성된 클라스 확인



디렉토리 구조

```
├─── repository/
        |    ├─── StudentRepository.java          // 기존 Repository
        |    ├─── StudentRepositoryCustom.java    // 사용자 정의 인터페이스
        |    └─── StudentRepositoryImpl.java      // 사용자 정의 구현 클래스
        |
        └─── config/
             └─── QuerydslConfig.java            // Querydsl 설정 클래스
```

연결대상

연결 대상	설명
StudentRepository ↔ StudentRepositoryCustom	사용자 정의 기능 명세
StudentRepositoryImpl	실제 Querydsl 쿼리 실행
QuerydslConfig	JPAQueryFactory 를 Bean 으로 등록해서 Impl 에 주입
Q 타입 (QStudent, QCourse)	mvn compile 시 자동 생성, join()에서 사용

com.sec01.config

```
@Configuration
public class QuerydslConfig {

    @PersistenceContext
    private EntityManager em;

    @Bean
    public JPAQueryFactory queryFactory() {
        return new JPAQueryFactory(em);
    }
}
```

사용자 정의 Repository 인터페이스 com.sec01.repository

```
public interface StudentRepositoryCustom {
    List<Student> findByCourseTitle(String courseTitle);
}
```

사용자 정의 Repository 구현 클래스 com.sec01.repository

```
@Repository
public class StudentRepositoryImpl extends QuerydslRepositorySupport implements StudentRepositoryCustom {

    private final JPAQueryFactory queryFactory;

    public StudentRepositoryImpl(JPAQueryFactory queryFactory) {
        super(Student.class);
        this.queryFactory = queryFactory;
    }

    @Override
    public List<Student> findByCourseTitle(String courseTitle) {
        QStudent student = QStudent.student;
        QCourse course = QCourse.course;

        return queryFactory
                .selectFrom(student)
                .join(student.courses, course)
                .where(course.title.eq(courseTitle))
                .fetch();
    }
}
```

메인 Repository 에 등록 com.sec01.repository. StudentRepository.java

```
public interface StudentRepository extends JpaRepository<Student, Long>,
StudentRepositoryCustom {
}
```

서비스 확인 com.sec01.service

```
// [7] 특정 과목명을 수강하는 학생 목록 조회
        public List<Student> getStudentsByCourseTitle(String courseTitle) {
            return studentRepository.findByCourseTitle(courseTitle);
        }
```

컨트롤러 확인

```
//[7] 특정 과목명을 수강하는 학생 목록을 조회하는 기능
    @GetMapping("/students/by-course")
    public String showStudentsByCourse(@RequestParam("title") String courseTitle,
Model model) {
        List<Student> students = service.getStudentsByCourseTitle(courseTitle);
        model.addAttribute("students", students);
        model.addAttribute("courseTitle", courseTitle);
        return "students-by-course";      }
```