Chapter 5-3 JPA 활용하기

application.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/demo_jpa_schema
    username: demo_jpa
    password:
  jpa:
    hibernate:
      ddl-auto: create
    show-sql: false
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect
```

`ddl-auto` : 테이블을 생성하고 제거하는 과정을 자동으로 해주는 옵션

`show-sql` : jpa가 작동하면서 실제 실행하는 SQL문을 보여줄지 말지 설정하는 옵션

`dialect` : 어떤 DB 사용할지 알려주는 부분 ?!

Entity 작성하기

`@Id` : primary key

`@GeneratedValue(strategy = GenerationType.IDENTITY)` ID 생성 규칙을 적용해주기 위한 ?!

`@OneToMany` 1:N 관계

`@ManyToOne` N:1 관계

속성

optional : false 설정 시 연관된 엔티티가 항상 있어야 함. (default. true)

- EAGER 설정의 조인 전략

@ManyToOne, @OneToOne

- (optional = false) : 내부 조인
- (optional = true) : 외부 조인

@OneToMany, @ManyToMany

- (optional = false) : 외부 조인
- (optional = true) : 외부 조인

fetch : global fetch 전략 설정

@ManyToOne, @OneToOne(fetch = FetchType.EAGER) *"즉시 로딩"*

- 엔티티를 조회할 때 연관된 엔티티도 함께 조회
- 연관된 엔티티를 즉시 조회, hibernate는 가능하면 SQL 조인을 사용해서 한 번에 조회
- 객체가 계속 연결되어있을 경우 무한루프로 stack overflow 가능성..

@OneToMany, ManyToMany(fetch = FetchType.LAZY) *"지연 로딩"*

- 연관된 엔티티를 프록시로 조회, 프록시를 실제 사용할 때 초기화하면서 데이터베이스 조회
- 프록시 객체 : 지연 로딩 기능을 위해 실제 엔티티 객체 대신 데이터베이스 조회를 지연할 수 있는 가짜 객체

cascade : 영속성 전이 기능 사용

@OneToMany(mappedBy = "parent", cascade = CascadeType.PERSIST)

- 부모를 영속화할 때 연관된 자식들도 함께 영속화
- 부모 엔티티를 저장할 때 자식 엔티티도 함께 저장

, cascade = CascadeType.REMOVE

- 부모 엔티티만 삭제하면 연관된 자식 엔티티도 함께 삭제
- CasecadeType 의 다양한 옵션

ALL, // 모두 적용

PERSIST, // 영속

MERGE, // 병합

REMOVE, // 삭제

REFRESH,

DETACH

orphanRemoval : 부모 엔티티의 컬렉션에서 자식 엔티티의 참조만 제거하면 자식 엔티티가 자동으로 삭제(true)

- 참조하는 곳이 하나일 때만 사용 --> @OneToOne, @OneToMany
- CascadeType.Remove 설정과 동일
- cascadeType.All + orphanRemoval = true 일 경우, 부모 엔티티를 통해서 자식의 생명주기를 관리할 수 있음.
- 자식을 저장하려면 부모에 등록(CASECADE)
- 자식을 삭제하려면 부모에서 제거(removeObject)

Fetch Type

: JPA 가 하나의 Entity 를 조회할 때, 연관관계에 있는 객체들을 어떻게 가져올 것이냐를 나타내는 설정값

- 연관 관계에 있는 Entity 들 모두 가져온다 → Eager 전략
- 연관 관계에 있는 Entity 가져오지 않고, getter 로 접근할 때 가져온다 → Lazy 전략

ManyToOne 컬럼 있을 때 (주인일 때)

Order Entity는 단일 Member Entity 를 가지는 ManyToOne 컬럼이 있음 (member 의 PK 가 Foreign Key 로 실제로 order DB컬럼에 매핑되어있으므로 Order 가 주인)

- ManyToOne 의 기본 FetchType 은 EAGER

OneToMany 컬럼 있을 때 (종일 때)

Member Entity 에는 Order Entity Collection (List 혹은 Set) 을 가지는 OneToMany 컬럼이 있음

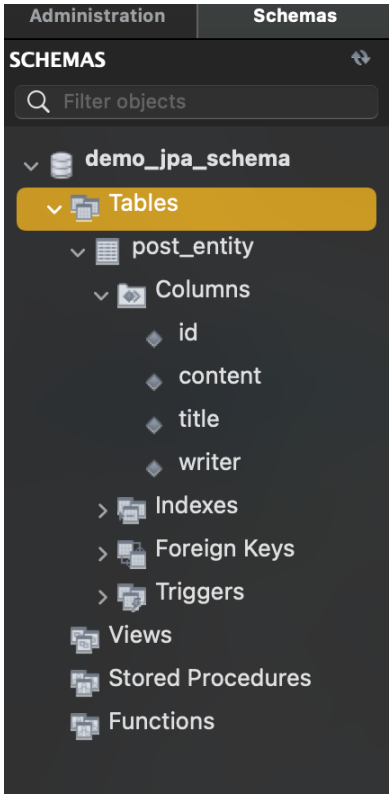
- OneToMany 의 기본 FetchType 은 LAZY

fetch = FetchType.EAGER

fetch = FetchType.LAZY

PostEntity

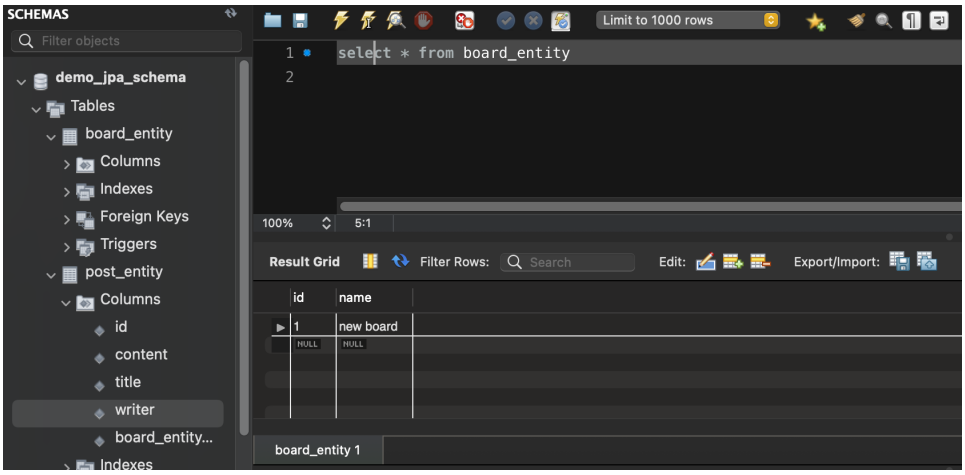
```
package dev.hsooovn.jpa.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class PostEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String content;
    private String writer;

}
```



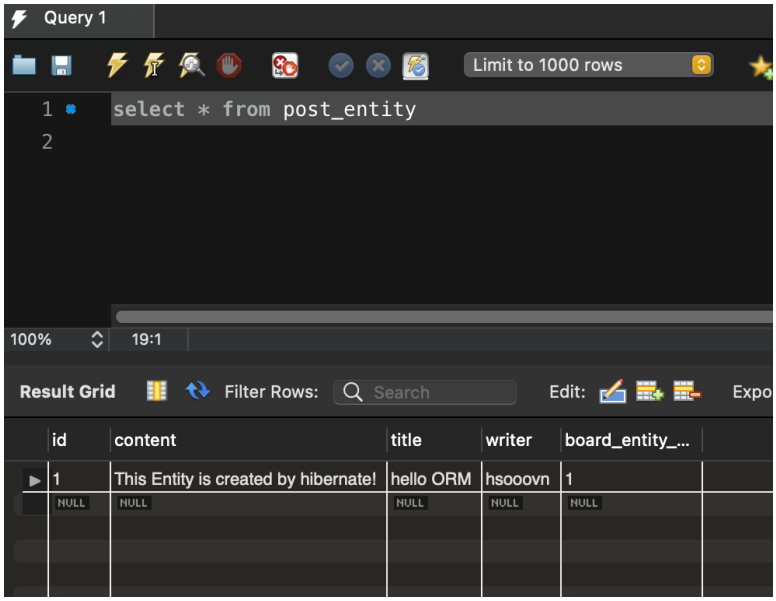
TestComponent

```
package dev.hsooovn.jpa;

import dev.hsooovn.jpa.entity.BoardEntity;
import dev.hsooovn.jpa.repository.BoardRepository;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Component;

@Component
public class TestComponent {
    public TestComponent(
            @Autowired BoardRepository boardRepository
    ){
        BoardEntity boardEntity = new BoardEntity();
        boardEntity.setName("new board");
        BoardEntity newBoardEntity = boardRepository.save(boardEntity);
        System.out.println(newBoardEntity.getName());
    }
}
```




```
PostEntity postEntity = new PostEntity();
        postEntity.setTitle("hello ORM");
        postEntity.setContent("This Entity is created by hibernate!");
        postEntity.setWriter("hsooovn");
        postEntity.setBoardEntity(newBoardEntity);
        PostEntity newPostEntity = postRepository.save(postEntity);
```



JPA Relationships

객체 연관관계 vs 테이블 연관관계

- 객체는 참조(주소)로 연관관계

ㄴ 연관 데이터 조회 시 .get() (참조) 사용 => *단방향*

ㄴ 객체 그래프 탐색

- 테이블은 외래 키로 연관관계

ㄴ 연관 데이터 조회 시 JOIN 사용 => *양방향*

- 객체를 양방향으로 참조하려면 단방향 연관관계를 2개 만들어야 함.

Table, Column, JoinColumn

`@Table` : 엔티티와 매핑할 테이블을 지정

: 생략 시 매핑한 엔티티 이름을 테이블 이름으로 사용

속성

Name : 매핑할 테이블 이름 (default. 엔티티 이름 사용)

Catalog : catalog 기능이 있는 DB에서 catalog 를 매핑 (default. DB 명)

Schema : schema 기능이 있는 DB에서 schema를 매핑

uniqueConstraints : DDL 생성 시 유니크 제약조건을 만듦

스키마 자동 생성 기능을 사용해서 DDL을 만들 때만 사용

`@Column` : 객체 필드를 테이블 컬럼에 매핑

속성 중 name, nullable이 주로 사용되고 나머지는 잘 사용되지 않음

name : 필드와 매핑할 테이블 컬럼 이름 (default. 객체의 필드 이름)

nullable (DDL) : null 값의 허용 여부 설정, false 설정 시 not null (default. true)

@Column 사용 시 nullable = false 로 설정하는 것이 안전

unique (DDL) : @Table 의 uniqueConstraints와 같지만 한 컬럼에 간단히 유니크 제약조건을 적용

columnDefinition (DDL) : 데이터베이스 컬럼 정보를 직접 줄 수 있음, default 값 설정

(default. 필드의 자바 타입과 방언 정보를 사용해 적절한 컬럼 타입을 생성)

length (DDL) : 문자 길이 제약조건, String 타입에만 사용 (default. 255)

percision, scale (DDL) : BigDecimal, BigInteger 타입에서 사용. 아주 큰 숫자나 정밀한 소수를 다룰 때만 사용 (default. precision = 19, scale = 2)

`@JoinColumn` : 외래키 매핑할 때 사용

name : 매핑할 외래 키 이름

default. [필드명]_[참조하는 테이블의 기본키 컬럼명]

referencedColumnName : 외래 키가 참조하는 대상 테이블의 컬럼명

default. 참조하는 테이블의 기본 키 컬럼명

foreignKey (DDL) : 외래 키 제약조건을 직접 지정 (테이블 생성시에만 사용)

unique : @Column 속성과 동일

nullable

insertable

updatable

columnDefinition

table

- @JoinColumn 생략 시 외래 키를 찾을 때 기본 전략을 사용

[필드명]_[참조하는 테이블의 컬럼명]