4장 엔티티 매핑

4장 엔티티 매핑

4.1 @Entity

- @Entity 가 붙은 클래스는 JPA 가 관리하게 된다.
- 주의 사항
  - 기본 생성자는 필수 이다.(public or protected)
  - final class, enum, interface, inner 클래스 에는 사용 할 수 없다.
  - 저장 할 필드에 final 사용해선 안된다.
  - 자바의 경우 생성자를 하나 이상 만들 경우 기본 생성자를 만들지 않으므로 직접 만들어 줘야 한다.

```
//직접 만든 기본 생성자
public Member() {}
//임의의 생성자
public Member(String name) {
  this.name = name
}
```

4.2 @Table

- 엔티티와 매핑할 테이블을 지정 한다.
- 속성

- name : 매핑할 테이블 이름 (default : 엔티티 이름)
- catalog : catalog 기능이 있는 DB 에서 catalog 를 매핑
- schema : 스키마 기능이 있는 DB 에서 스키마를 매핑 , DDL 생성 시 유니크 제약 조건을 만든다.

4.3 다양한 매핑 사용

- @Enumerated , @temporal , @lob 등의 다양한 매핑을 entity 클래스 내부에 선언 할 수있다.
- 자세한 내용은 아래에 정리

4.4 데이터베이스 스키마 자동 생성

- persistence.xml 속성
  - <property name="hibernate.hbm2ddl.auto" value="create" />
    - 애플리케이션 실행 시점에 데이터베이스 테이블을 자동으로 생성
  - <property name="hibernate.show_sql" value="true" />
    - 콘솔에 실행되는 테이블 생성 DDL 을 출력 해 줌

  - hibernate.hbm2ddl.auto 속성
    - create : 기존 테이블을 삭제하고 새로 생성 (DROP + CREATE)
    - create-drop : 애플리케이션 종료 시 create 한 DDL 제거 (DROP+CREATE+DROP)
    - update : 데이터베이스 테이블과 엔티티 매핑정보를 비교 하여 변경 사항만 수정 한다.
    - validate : 데이터베이스 테이블과 엔티티 매핑정보를 비교하여 차이가 있을 시 경고를 남기고 애플리케이션을 실행 하지 않는다. (유효성검사, DDL 수정하지 않음)
    - none : 자동 생성 기능 사용 안할 경우 (none 대신 유효하지 않는 값을 넣어도 사용 안함, none 자체가 유효하지 않은 값임 (의미상 편하게 알아보려고 none 을 썼음))

- 참고
  - hibernate.ejb.naming_strategy 속성을 이용 시 자바의 카멜표기법을 언더스코어(_) 표기법으로 매핑 해 준다.
    - <property name="hibernate.ejb.naming_strategy" value = "org.hibernate.cfg.ImproveNamingStrategy" />

4.5 DDL 생성 기능

- @Column 매핑 정보의 nullable 속성을 false 지정 시 DDL 에 not null 제약조건 추가 가능
  - length 속성 이용 시 문자 크기 지정 할 수 있음

```
@Column(name = "NAME", nullable = false, length = 10)

//결과
create Table MEMBER {
  ...
  NAME varchar(10) not null,
  ...
}
```

- **이런 기능들은 단지 DDL 을 자동 생성 할때만 사용되고 JPA 실행 로직에는 영향을 주지 않는다!**
- 이 기능을 이용 시 개발자가 엔티티만 보고도 손쉽게 다양한 제약조건을 파악할 수 있는 장점이 있다

4.6 기본 키 매핑

- JPA가 제공 하는 데이터베이스 기본 키 생성 전략
  - 직접 할당 : 기본 키를 애플리케이션에서 직접 데이터베이스에 할당 한다.
  - 자동생성 : 대리키 사용 방식
    - IDENTITY : 기본 키 생성을 데이터베이스에 위임 한다. (ex. mysql)
    - SEQUENCE : DB 시퀀스를 사용 해 기본 키를 할당 한다. (ex. oracle)
    - TABLE : 키 생성 테이블을 사용

            - 키 생성용 테이블을 하나 만들어 두고 시퀀스 처럼 사용 하는 방법, 이 방법은 테이블을 활용하므로 oracle,mysql 등을 가리지 않고 모든 데이터베이스에서 사용 할 수 있다.
- 키 생성 전략을 사용하려면 persistence.xml 에 hibernate.id.new_generator_mappings 속성을 반드시 추가 해야 함. JPA는 과거 버전과의 호환성을 위해 기본값을 false 로 설정 해두었음
    - <property name="hibernate.id.new_generator_mappings" value="true" />

- 직접 할당
    - @Id 로 매핑
        - @Id 적용 가능 한 자바 타입 목록
            - 자바 기본형
            - 자바 wrapper 형
            - String
            - java.util.Date
            - java.sql.Date
            - java.math.BigDecimal
            - java.math.BigInteger
        - 사용

```
Board board = new Board();
board.setId("id1") // 기본키 직접 할당
em.persist();
```

- IDENTITY 전략
    - 기본 키 생성을 데이터베이스에 위임 하는 전략 (Mysql, postgreSQL, SQLServer, DB2 에서 주로 사용)
    - JPA는 기본 키 값을 얻어오기 위해 데이터베이스를 추가로 조회 하게 된다.

```
@Id
@GeneratedValue(strategy = Generation.Type.IDENTITY)
```

- 하이버네이트의 경우 JDBC3 에 추가된 Statement.getGeneratedKeys() 를 사용 해 데이터 저장과 동시에 생성된 기본 키 값을 얻어 올 수 있다. 즉, 하이버네이트는 이 메소드를 사용 해 데이터베이스와 한번만 통신한다.
- IDENTITY 식별자 생성 전략은 엔티티를 데이터베이스에 저장 해야 식별자를 얻을 수 있으므로 em.persist() 호출 즉시 INSERT SQL이 데이터베이스에 전달 된다.
  → 쓰기지연 동작 하지 않음

- SEQUENCE 전략
  - 데이터베이스 시퀀스를 사용 해 기본 키를 생성 한다. (주로 오라클 ,PostgreSQL , DB2 , H2 에 사용 할 수 있다)

```
 @SequenceGenerator(
    name = "SOON_SEQ_GENERATOR",
    sequenceName = "SOON_SEQ", //DB 시퀀스 이름
    initialValue = 1, allocationSize = 1)

  ...
 @Id
 @GeneratedValue(strategy = GenerationType.SEQUENCE,
                 generator = "SOON_SEQ_GENERATOR")
  private Long id;

 ...

// id 식별자 값은 SOON_SEQ_GENERATOR 시퀀스 생성기가 할당 한다.
```

  - 내부 동작 방식
    - em.persist() 호출 시 먼저 데이터베이스 시퀀스를 사용 해 식별자 조회
    - 조회한 식별자를 엔티티에 할당 후 영속성 컨텍스트에 저장
    - 이후 트랜잭션 커밋하여 플러시가 일어날 때 엔티티를 데이터베이스에 저장
    - IDENTITY 전략은 먼저 엔티티를 데이터베이스에 저장 한 뒤 식별자 조회 후 엔티티의 식별자에게 할당 하는데 이와는 정 반대의 동작 방식이다.
  - @SequenceGenerator
    - name : 식별자 생성기 이름

    - squenceName : 데이터베이스에 등록 되어 있는 시퀀스 이름
    - initialValue : DDL 생성 시에만 사용되며 시퀀스 DDL을 생성 시 처음 시작 하는 수 지정 (default:1)
    - allocationSize : 시퀀스를 호출 시 마다 증가 하는 수 (default : 50)
    - catalog, schema : 데이터베이스 카탈로그,스키마 이름

- 최적화 관련
    - 시퀀스 전략은 결국 데이터베이스와 2번 통신 하게 된다.
        - 식별자 구하기 위해 DB 조회, 조회 한 시퀀스를 기본 키값으로 사용해 DB에 저장
        - 매번 식별자를 구하기 위해 DB 조회하는걸 줄이고자 allocationSize 에 저장된 값만큼 시퀀스 값을 늘려놓고 그만큼의 값을 메모리에 할당 시켜놓고 allocationSize 에 저장된 값을 초과 하기 전까지 메모리에서 식별자를 할당 해주는 최적화 방법이 있다.
    - allocationSize 가 기본 50인 이유는 한번에 시퀀스를 50 증가시킨 뒤 1~50까지는 메모리에서 식별자를 할당하게 되고, 시퀀스가 51이 되면 시퀀스를 100까지 증가 시킨 뒤 51~100까지 메모리에서 식별자를 할당 한다.
    - 이러한 최적화 방법은 여러 JVM 이 동시에 테이블에 접근 하여도 기본 키 값 충돌을 피할 수 있는 장점이 있다.
        - 여러 JVM 이 동시에 insert 요청을 할 때 식별자를 구하기 위해 DB 조회 시 같은 시퀀스 값을 받게 될 경우 데이터 등록 시 기본키 값 충돌이 일어날 수 있으나, 한번에 50 이상씩 설정 해놓게 될 경우엔 충돌 문제가 해결된다.
        - 예를 들어 JVM#1이 식별자 조회 시 값이 1일 경우 1~50까지 시퀀스를 늘려 놓는다.
        - 바로 다음에 JVM#2 이 식별자 조회 시 값은 51이 되므로 51~100까지 시퀀스를 늘려 놓는다.
        - 결국 요청 할때마다 애플리케이션이 관리(메모리) 할 시퀀스 범위만큼 늘려놓기 때문에 기본 키 값 충돌은 일어나지 않게 된다.
    - 하지만 데이터베이스에 한번 접근 시 시퀀스를 한번에 많이 증가시켜야 한다는 점을 염두해 둬야 한다!

- Table 전략
  - 키 생성 전용 테이블을 하나 만들고, 여기에 이름과 값으로 사용할 컬럼을 만들어 데이터베이스 시퀀스를 흉내내는 전략 (모든 데이터베이스에 적용 가능)

```
create table SOON_SEQUENCES {
  sequence_name varchar(255) not null , // 시퀀스 이름
  next_level bigint,  // 넥.스트 레블~ (ㅈㅅ), 시퀀스 값
  primary key (sequence_name)
}
```

```
@Entity
@TableGenerator(
  name = "SOON_SEQ_GENERATOR",
  table = "SOON_SEQUENCES",
  pkColumnValue = "SOON_SEQ", allocationSize = 1)

...

@Id
@GeneratedValue(strategy = GenerationType.TABLE,
                  generator = "SOON_SEQ_GENERATOR")
private Long id;
...

//SOON_SEQ 테이블 조회 결과
sequence_name | next_level
SOON_SEQ      | 2

//SOON_SEQ 테이블에 값이 없으면 JPA 가 알아서 값을 insert 하면서 초기화 한다.
```

  - @TableGenerator
    - name : 식별자 생성기 이름
    - table : 키 생성 테이블 명 (default : hibernate_sequences)
    - pkColumnName : 시퀀스 컬럼명 (default : sequence_name)
    - valueColumnName : 시퀀스 값 컬럼명 (default : next_val)
    - pkColumnValue : 키로 사용할 값 이름
    - initialValue : 초기 값 (default 0)
    - allocationSize : 시퀀스 호출 시 마다 증가 하는 수 (default : 50)
    - catalog,schema : DB 카탈로그, 스키마 명
    - uniqueConstraints(DDL) : 유니크 제약 조건

    - AUTO 전략
        - AUTO 설정 시 선택하는 데이터베이스 방언에 따라 IDENTITY, SEQUENCE, TABLE 중 하나를 자동으로 선택 함
        - 데이터베이스를 변경 해도 코드 수정 할 필요가 없는 장점이 있다.
- 참고 사항
    - 자연 키
        - 비즈니스에 의미가 있는 키 (주민번호, 이메일, 전화번호 등..)
    - 대리 키
        - 비즈니스와 관련 없는 임의로 만들어진 키, 대체 키로도 불림 (오라클 시퀀스, auto_increment, 키생성 테이블 등..)
    - 자연 키 보다는 대리키 사용을 권장 함.

4.7 필드와 컬럼 매핑 : 레퍼런스

필드와 컬럼 매핑 분류

Aa 분류	매핑 어노테이션	설명
필드와 컬럼 매핑	@Column	컬럼을 매핑 한다
제목 없음	@Enumerated	자바의 enum 타입을 매핑 한다
제목 없음	@Temporal	날짜 타입을 매핑 한다
제목 없음	@Lob	BLOB,CLOB 타입을 매핑 한다
제목 없음	@Transient	특정 필드를 DB 매핑 하지 않는다
기타	@Access	JPA가 엔티티에 접근하는 방식을 지정한다

4.8 정리

- 객체와 테이블 매핑 (@Entity, @Table 등 ...)
- 기본 키 매핑
    - 직접 할당 (@Id)
    - 시퀀스 (DB 시퀀스에서 식별자 획득 후 영속컨텍스트 저장)
    - 아이덴티티 (DB 에 엔티티 저장 후 식별자 획득 후 영속성 컨텍스트에 저장)→ DB 에 데이터를 먼저 저장한다.

    - 테이블 전략 (DB 시퀀스 생성용 테이블에서 식별자 값 획득 후 영속 컨텍스트에 저장)
- 필드와 컬럼 매핑
    - 레퍼런스..

실전 예제 | 1. 요구사항 분석과 기본 매핑

- 객체는 참조를 사용하여 연관된 객체를 찾고, 테이블은 외래 키를 사용 해 연관된 테이블을 찾으므로 둘 사이에는 큰 차이가 있다.
- JPA는 객체의 참조와 테이블의 외래 키를 매핑해서 객체에서는 참조를 사용하고, 테이블에서는 외래 키를 사용하도록 한다. → 5장에서 계속