**04. 엔티티 매핑**

\* 엔티티 매핑 소개

객체와 테이블 매핑 : @Entity, @Table

필드와 컬럼 매핑 : @Column

기본 키 매핑 : @Id

연관관계 매핑 : @ManyToOne, @JoinColumn

*- 객체와 테이블 매핑*

**@Entity**

1. @Entity가 붙은 클래스는 JPA가 관리, 엔티티
2. JPA를 사용해서 테이블과 매핑할 클래스는 @Entity 필수
3. 기본 생성자(public, protected) 필수
4. Final 클래스, enum, interface, inner 클래스 사용 x
5. 저장할 필드에 final 사용 x

- 속성

1. JPA에서 사용할 엔티티 이름을 지정

2. 기본값 : 클래스 이름을 그대로 사용

3. 같은 클래스 이름이 없으면 가급적 기본값 사용

**@Table**

1. 엔티티와 매핑할 테이블 지정

- *데이터 베이스 스키마 자동 생성*

1. DDL을 애플리케이션 실행 시점에 자동 생성

2. 테이블 중심 -> 객체 중심

3. 데이터베이스 방언을 사용해서 데이터베이스에 맞는 적절한 DDL 생성

4. 개발 장비에서만 사용

5. 생성된 DDL은 운영서버에서는 사용하지 않거나, 적절히 다듬은 후 사용

; 운영 장비에는 절대 create, create-drop, update 사용하면 안됨

*- 필드와 컬럼 매핑*

1. 회원은 일반 회원과 관리자로 구분해야한다.

2. 회원 가입일과 수정일이 있어야 한다.

3. 회원을 설명할 수 있는 필드가 있어야 한다. 길이 제한이 없음.

*- 기본 키 매핑*

•직접할당: @Id만사용

•자동생성(@GeneratedValue)

•IDENTITY: 데이터 베이스에 위임, MYSQL

•SEQUENCE: 데이터 베이스 시퀀스 오브젝트 사용, ORACLE

•@SequenceGenerator 필요

•TABLE: 키생성용 테이블 사용, 모든 DB에서 사용

•@TableGenerator 필요

•AUTO: 방언에 따라 자동 지정, 기본값

IDENTITY 전략 – 특징

• 기본 키 생성을 데이터 베이스에 위임

• 주로 MySQL, PostgreSQL, SQL Server, DB2에서사용 (예: MySQL의 AUTO\_ INCREMENT)

• JPA는 보통 트랜잭션 커밋 시점에 INSERT SQL 실행

•AUTO\_ INCREMENT는 데이터 베이스에 INSERT SQL을실행한이후에 ID 값을 알 수 있음

• IDENTITY 전략은 em.persist() 시점에 즉시 INSERT SQL 실행하고 DB에서 식별자를 조회

SEQUENCE 전략 – 특징

• 데이터 베이스 시퀀스는 유일한 값을 순서대로 생성하는 특별한 데이터 베이스 오브젝트 (예: 오라클시퀀스)

• 오라클, PostgreSQL, DB2, H2 데이터 베이스에서 사용

@SequenceGenerator

Table 전략

• 키 생성 전용 테이블을 하나 만들어서 데이터 베이스 시퀀스를 흉내내는 전략

• 장점: 모든 데이터 베이스에 적용 가능

•단점: 성능

권장하는 식별자 전략

• 기본 키 제약 조건: null 아님, 유일, 변하면 안된다.

• 미래까지 이 조건을 만족하는 자연 키는 찾기 어렵다. -> 대리키 (대체키)

• 권장 : Long형 + 대체키 + 키 생성 전략 사용

**05. 연관 관계 매핑 기초**

- 단방향 연관 관계

객체 지향 모델링

; 객체 연관관계 사용, 객체의 참조와 테이블의 외래 키를 매핑, ORM 매핑,

연관 관계 저장, 참조로 연관 관계 조회 – 객체 그래프 탐색, 연관 관계 수정

- 양방향 연관 관계

양방향 매핑

; Member 엔티티는 단방향과 동일, Team 엔티티는 컬렉션 추가, 반대 방향으로 객체 그래프 탐색

mappedBy = JPA의 멘탈 붕괴 난이도

객체의 양방향 관계

- 서로 다른 단방향 관계 2개

- 단방향 연관 관계를 2개 만들어야함

테이블의 양방향 연관관계

- 테이블은 외래 키 하나로 두 테이블의 연관 관계를 관리

- MEMBER, TEAM\_ID 외래 키 하나로 양방향 연관 관계 가짐 (양쪽으로 조인)

- 둘 중 하나로 외래 키를 관리

양방향 매핑 규칙

- 객체의 두 관계중 하나를 연관 관계의 주인으로 지정

- 연관관계의 주인만이 외래 키를 관리(등록, 수정)

- 주인이 아닌 쪽은 읽기만 가능

- 주인이 아닌 쪽은 읽기만 가능

- 주인은 mappedBy 속성 사용 x

- 주인이 아니면 mappedBy 속성으로 주인 지정 ; 주인 = 외래 키가 있는 곳

**06. 다양한 연관관계 매핑**

\* 연관관계 매핑시 고려사항 : 다중성, 방향, 연관관계의 주인

다중성

- 다대일 @ManyToOne

가장 많이 사용하는 연관 관계

외래 키가 있는 족이 연관관계의 주인

양쪽으로 서로 참조하도록 개발

- 일대다 @OneToMany

일이 연관관계의 주인

다(N)쪽에 외래 키 존재

객체와 테이블의 차이 때문에 반대쪽 테이블의 외래 키를 관리하는 특이한 구조

@JoinColumn을 꼭 사용해야함, 그렇지 않으면 조인 테이블 방식 사용

엔티티가 관리하는 외래 키가 다른 테이블에 있음

연관관계 관리를 위해 추가로 UPDATE SQL 실행

다대일 양방향 매핑을 더 사용

- 일대일 @OneToOne

주 테이블이나 대상 테이블 중에 외래 키 선택 가능

외래 키에 데이터 베이스 유니크(UNI) 제약 조건 추가

다대일 단방향 매핑과 유사

외래 키가 있는 곳이 연관관계, 반대편은 mappedBy

- 다대다 @ManyToMany

관계형 데이터 베이스는 정규화된 테이블 2개로 다대다 표현 붕가능

연결 테이블 추가 : @JoinTable

객체는 컬렉션을 사용, 객체 2개로 다대다 관계 가능

\*한계 : 실무에 사용x

단방향, 양방향

- 테이블

1. 외래 키 하나로 양쪽 조인 가능

2. 방향이라는 개념 없음

- 객체

1. 참조용 필드가 있는 쪽으로만 참조 가능

2. 한 쪽만 참조하면 단방향

3. 양쪽이 서로 참조하면 양방향

- 연관관계의 주인

1. 테이블은 외래 키 하나로 두 테이블이 연관관계를 맺음

2. 객체 양방향 관계는 참조가 2군데 -> 둘 중 테이블의 외래 키를 관리하는 곳 지정

3. 연관 관계의 주인 : 외래 키를 관리하는 참조

4. 주인의 반대편 : 외래 키에 영향을 주지 않음, 단순 조회만 가능

**07. 고급 매핑**

상속 관계 매핑

- 관계형 데이터 베이스는 상속 관계 x

- 슈퍼 타입 서브 타입 관계라는 모델링 기법이 객체 상속과 유사

- 상속 관계 매핑 : 객체의 상속과 구조와 DB의 슈퍼타입 서브타입 관계를 매핑

- 슈퍼 타입 서브 타입 논리 모델을 실제 물리 모델로 구현

각각 테이블로 변환 -> 조인 전략

통합 테이블로 변환 -> 단일 테이블 전략

서브 타입 테이블로 변환 -> 구현 클래스마다 테이블 전략

\*주요 어노테이션

@Inheritance(strategy=InheritanceType.XXX)

JOINED : 조인 전략

SINGLE\_TABLE : 단일 테이블 전략

TABLE\_PER\_CLASS : 구현 클래스마다 테이블 전략

@DiscriminatorColumn(name=“DTYPE”)

@DiscriminatorValue(“XXX”)

조인 전략

- 장점 : 테이블 정규화, 외래 키 참조 무결성 제약조건 활용 가능, 저장공간 효율화

- 단점 : 조회시 조인을 많이 사용, 성능 저하, 조회 쿼리가 복잡함, 데이터 저장시 INSERT SQL 2번 호출

단일 테이블 전략

- 장점 : 조인이 필요없으므로 일반적으로 조회 성능이 빠름, 조회 쿼리가 단순함

- 단점 : 자식 엔티티가 매핑한 칼럼은 모두 null 허용, 단일 테이블에 모든 것을 저장하므로 테이블이 커질 수 있는 상황에 따라서 조회 성능이 느려질 가능성 있음

구현 클래스마다 테이블 전략

- 이 전략은 데이터 베이스 설계자와 ORM 전문가 둘자 추천 x

- 장점 : 서브 타입을 명확하게 구분해서 처리할 때 효과적, not null 제약 조건 사용 가능

- 단점 : 여러 자식 테이블을 함께 조회할 때 성능이 느림, 자식 테이블을 통합해서 쿼리하기 어려움

- @MappedSuperClass

공통 매핑 정보가 필요할 때 사용(id, name)

상속관계 매핑, 엔티티, 테이블과 매핑 x

부모 클래스를 상속받는 자식 클래스에 매핑 정보만 제공

조회, 검색(em.find(BaseEntity) 불가

직접 생성해서 사용할 일이 없으므로 추상 클래스 권장

테이블과 관계 없고, 단순히 엔티티가 공통으로 사용하는 매핑 정보를 모으는 역할