Section 5~7

JPA

목차

엔티티 매핑 기초

객체의 참조와 테이블의 외래 키 매핑

객체는 참조 시 객체의 참조값을 저장하지만 테이블의 경우에는 Id로 저장하는 차이가 발생
JPA에서는 이를 해결 가능하다.
단방향 매핑에서는 객체를 참조하고 다대다(@ManyToMany), 일대다(@OneToMany)등 표시해주고
@JoinColumn으로 조인 해야하는 컬럼을 명시해주면 된다(이 부분은 생략 가능)

양방향에서는 두 객체 모두 객체를 참조해야 하는데 일대다의 경우에는 리스트를 넣어준다.
그리고 나서 @OneToMany 속성에 mappedBy = "참조 할 객체의 변수명"를 명시해 준다
이렇게 되면 2개의 객체가 각자 관리하게 되는데 실제 DB에서는 외래 키 하나로만 관리하기 때문에 차이점이 생긴다
그렇기 때문에 연관관계 주인이라는 개념이 생기게 된다, 여기서 주인은 mappedBy사용이 안된다

주인은 외래 키가 있는 곳을 주인으로 지정하면 된다.(필수는 아님)

단방향을 주로 개발 하고 필요시에만 양방향으로 설계

고급 매핑

상속관계 매핑

RDB에는 상속이 없음
슈퍼타입, 서브타입이 객체 상속과 유사함
JPA 기본 전략은 단일 테이블 전략
속성으로 변경이 가능하다.
@Inheritance(strategy = InheritanceType.전략)
전략 :
JOINED-조인
@DiscriminatorColumn ➔ 어느 타입인지 알려줌(DTYPE)
DTYPE은 자식 테이블에 @DiscriminatorValue("NAME")으로 지정 가능
장점 : 테이블 정규화, 외래 키 참조 무결성, 저장공간 효율
단점 : 조회 시 조인 많이 사용(성능 저하), 저장 시 insert sql 2번 호출, 쿼리가 복잡함

SINGLE_TABLE-단일 테이블 전략
@DiscriminatorColumn 생략 가능(자동 생성)
장점 : 조인이 필요 없어 조회 성능 빠름, 조회 쿼리 단순함
단점 : 자식 엔티티가 매핑한 컬럼 null사용 강제, 테이블이 너무 커지면 성능 저하(한곳에만 저장하므로)

TABLE_PER_CLASS-구현 클래스마다 테이블 전략
장점 : not null 사용 가능, 서브타입을 명확하게 구분해서 처리할 때 효과적
단점 : 통합 쿼리 어려움, 자식테이블들끼리 조회하면 성능 저하(UNION)

고급 매핑

@MappedSuperclass

공통 매핑 정보가 필요할 때 사용
매핑 정보만 지정(반복되는 것 들) : @MappedSuperclass
간단하게 말해서 속성을 같이 쓰고 싶을 때
엔티티가 아님 → 조회, 검색 불가능
직접 생성해서 사용할 일이 없으므로 추상 클래스 권장