Impedance Mismatch (패러다임 불일치)

자바 ORM 표준 JPA 프로그래밍

JPA 지연로딩 김영한 패러다임 불일치

hyeonZIP

2 days ago · 10 min read

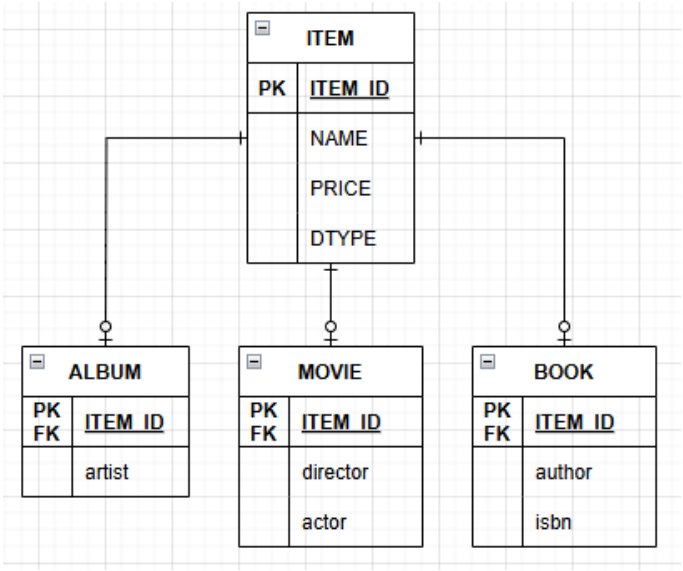
> *"Impedance mismatch is a term used in computer science to describe the problem that arises when two systems or components that are supposed to work together have different data models, structures, or interfaces that make communication difficult or inefficient.*

객체(Object)와 관계형 데이터베이스(RDB)는 지향하는 목적이 다르다.

객체 지향 프로그래밍 모델은 데이터를 **속성**과 **메서드**가 있는 **객체**로 표현하지만, **관계형 모델**은 데이터를 **열**과 **행**이 있는 표로 표현한다.

상속의 부재

먼저 객체는 **상속**이라는 기능을 지원하지만 테이블은 상속이라는 기능이 없다.



위와 같은 데이터베이스 모델링을 객체로 모델링 하면 다음과 같다.

```
abstract class Item {
    Long id;
    String name;
    int price;
}

class Album extends Item {
    String name;
}

class Movie extends Item {
    String director;
    String actor;
}

class Book extends Item {
    String author;
    String isbn;
}
```

이제 Album 객체를 저장하려면 데이터베이스 입장에서는

ITEM 테이블과 ALBUM 테이블 각각에 2개의 쿼리를 작성해야 한다.

```
INSERT INTO ITEM...

INSERT INTO ALBUM...
```

반대로 조회할 때는 ITEM과 ALBUM 테이블을 조인한 결과를 AIbum 객체로 반환해야 한다.

만약 Java Collection에 보관한다면 상속에 대한 고민 없이 그냥 사용하면 된다.

```
list.add(album);
list.add(movie);

Album album = list.get(albumId);
```

연관관계 불일치

객체는 참조를 사용해서 다른 객체와 연관관계를 가지고 참조에 접근해서 연관된 객체를 조회한다.

하지만 테이블은 외래 키(Foreign Key)와 조인을 사용해서 연관된 테이블을 조회한다.

```
class Member{
    Team team;
    ...
    //가능
    Team getTeam(){
        return team;
    }
}

class Team{
    ...
    //불가능
```

```
    //Member getMember(){
    //    return member;
    //}
}
```

Member가 속한 Team을 알 수 있지만, Team에서 Member를 찾을 수 없다.

객체지향에서 **참조**란 Member에서 Team을 참조하듯이 Member는 Team을 알고 있지만, Team은 Member를 모르고 있어야 하는 것이다.

하지만 객체를 테이블에 맞춰서 모델링을 해도 문제가 발생한다.

```
class Member{
    String id;
    Long teamId;//Team 테이블의 PK값을 FK로 사용
    String username;
}

class Team{
    Long id;
    String name;
}
```

이렇게 저장하면 기존에는 Member에서 Team이라는 객체를 참조해서 Member가 속한 Team이 무엇인지 알 수 있었지만 이제는 할 수 없다. 이런 방식은 결국 객체지향의 특징을 지워버리는 것 이다.

```
class Member{
    String id;
    Team team;
    String username;

    Team getTeam(){
```

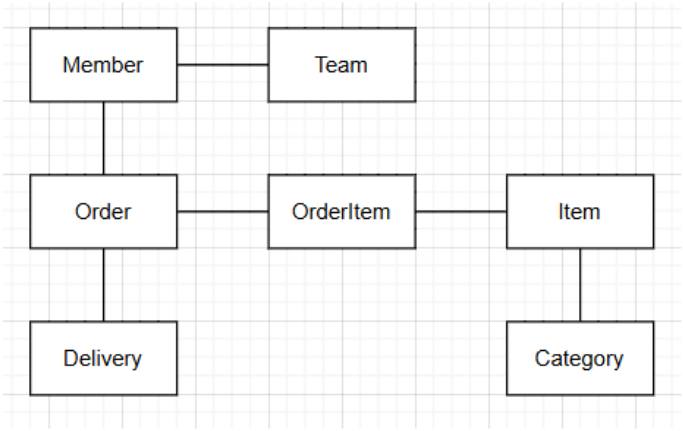
```
        return team;
    }
}

class Team{
    Long id;
    String name;
}
```

따라서 Team객체를 사용해야 한다.

하지만 결국 Member 테이블은 team_id 값 즉, FK 값을, Member 객체는 Team을 참조하기 위한 객체가 필요하기 때문에 이러한 차이는 중간에서 변환 역할을 해주어야 한다.

객체 그래프 탐색의 불확실성



해당 ERD만 보고 코드를 작성한다고 해보자.

```
class MemberService{
    ...
    public void process(){
        Member member = memberDAO.find(memberId);
        member.getTeam();
        member.getOrder().getDelivery();
```

```
    }
}
```

`memberId` 를 가지고 Member를 가지고 왔지만, Member에서 Team을 조회할 수 있나?

그러면 Member에서 Category까지 조회할 수 있을까? 이는 예측할 수가 없다.

정말 Category까지 조회할 수 있는지 확인하려면 결국 SQL을 직접 확인해야 한다.

이는 결국 **SQL에 의존적인 개발**이고, Entity가 SQL에 논리적으로 종속되어서 발생하는 문제이다.

동일성의 차이

- 동일성 비교는 `==` 로 객체 인스턴스의 주소 값을 비교한다.
- 동등성 비교는 `equals()` 메서드를 사용해서 객체 내부의 값을 비교한다.

```
class MemberDAO {
    public Member getMember(String memberId){
        String sql = "SELECT * FROM member WHERE member_id = ?";
        ...
        return new Member(...);
    }
}
```

```
String memberId = "100";
Member member1 = memberDAO.getMember(memberId);
Member member2 = memberDAO.getMember(memberId);
//member1과 member2는 서로 다른 객체이다. member1==member2 >> false
```

테이블에서 값을 조회하면 조회할 때마다 새로운 인스턴스를 생성해서 반환하기 때문에 false가 나온다.

```
Member member1 = list.get(0);
Member member2 = list.get(0);
//member1과 member2는 서로 같은 객체이다. member1==member2 >> true
```

JPA로 해결하기

상속

JPA가 대신해서 SQL문을 작성해주기 때문에 개발자는 Java Collection에 객체를 저장하듯이 JPA에 객체를 저장하면 된다.

```
//저장
jpa.persist(album);

//조회
String albumId = "id100";
Album album = jpa.find(Album.class, albumId);
```

저장 로직을 실행하면 JPA가 2개의 쿼리를 만들어주고 조회 로직을 실행하면 2개의 테이블을 조인한 결과를 Album객체로 반환해준다.

연관관계

```
member.setTeam(team); //회원과 팀 연관관계 설정
jpa.persist(member);  //회원과 연관관계 함께 저장
```

JPA는 team의 참조를 FK로 변환하여 INSERT 쿼리를 생성해주고 반대로 조회할 때도 FK를 참조하여 변환해주기도 한다.

객체 그래프 탐색

이전에 객체 그래프 탐색의 문제점은 SQL 쿼리에 의존적인 개발이 문제였다.

그도그럴것이 연관된 데이터를 모두 불러오는 것은 너무 비효율적이고 그렇다고 일부만 불러오니 비즈니스 로직을 수정하기 위해서는 SQL 쿼리를 봐야 한다.

따라서 이러한 비효율적인 문제를 개선하기 위해 JPA의 **지연 로딩**이 등장한다.

지연 로딩은 실제 객체를 사용하는 시점까지 데이터베이스 조회를 미루는 것이다. 따라서 테이블 간의 연관관계가 설정되어 있다면 어디든지 조회할 수 있게 된다.

```
//처음 조회 시점에 SELECT 쿼리
Member member = jpa.find(Member.class, memberId);

Order order = member.getOrder();
order.getOrderDate(); //Order를 사용하는 시점에 SELECT 쿼리
```

여기서 주의할 점은 **자기자신을 조회할 때는 즉시 조회**이지만 **참조를 조회하는 것은 지연 로딩 또는 즉시 로딩을 지정할 수 있다.**

비교

JPA는 같은 트랜잭션일 때 같은 객체가 조회되는 것을 보장하기 때문에 기존 객체 비교와 같이 동일성 비교가 동작한다.

정리

이처럼 JPA는 객체 모델과 관계형 데이터베이스 모델 간의 패러다임 불일치를 중간에서 해결해주고 동시에 Java의 객체 모델링을 유지하게 해준다.

이외에도 JPA를 사용함으로써 얻은 이점은 많다. 소개한 내용은 패러다임 불일치에 대한 JPA의 이점을 설명한 것이고 다음 포스트에서 JPA의 내부 구조와 함께 얻는 이점을 추가로 알아본다.

hyeonZIP
0 팔로워