**CHATER1**

* 1. **SQL을 직접 다룰 때 발생하는 문제점**

**1.1.1 반복, 반복 그리고 반복**

-데이터베이스는 객체 구조와는 다른 데이터 중심의 구조를 가지므로 객체를 데이터베이스에 직접 저장허간 조회할 수 없다.

-중간에서 SQL과 JDBC API를 사용해서 변환 작업

🡺너무 많은 SQL과 JDBC API를 코드로 작성해야 된다.

**1.1.2 SQL에 의존적인 개발**

-필드를 추가하고, SQL 쿼리도 수정해야 된다.

-만약 자바 컬렉션이라면

ex)

list.add(member) // 등록

Member member = list.get(xxx); // 조회

Member.setTel(“xxx”); // 수정

-Member나 Team처럼 비즈니스 요구사항을 모델링한 객체를 엔티티 라고 한다.

-애플리케이션에서 SQL을 직접 다룰 때 문제점

1) 진정한 의미의 계층 분할이 어렵다.

2) 엔티티를 신뢰할 수 없다. (계속 변경)

3) SQL에 의존적인 개발을 피하기 어렵다.

**1.1.3 JPA와 문제 해결**

-JPA를 사용하면 SQL을 직접 작성하는 것이 아니라 JPA가 제공하는 API를 사용하면 된다.

그러면 JPA가 개발자 대신에 적절한 SQL을 생성해서 데이터베이스에 전달한다.

**1.2 패러다임의 불일치**

-객체지향 프로그래밍은 추상화, 캡슐화, 정보은닉, 상속, 다형성 등 복잡성을 제어할 수 있는 다양한 장치들을 제공한다.

-도메인 모델도 객체로 모델링하면 객체지향 언어가 가진 장점들을 활용할 수 있다.

-하지만 부모 객체를 상속받았거나, 다른 객체를 참조하고 있다면 객체의 상태를 저장하기 쉽지 않다.

**🡺그러면 관계형은?** 데이터 중심으로 구조화되어 있기 때문에 객체지향에서 이야기하는 추상화, 상속, 다형성 같은 개념이 없다.

**1.2.1 상속**

-테이블은 상속이라는 기능이 없다. 그나마 슈퍼타입 서브타입 관계를 사용하면 유사한 형태로 테이블을 설계할 수 있다.

**JPA와 상속**

-JPA는 상속과 관련된 패러다임의 불일치 문제를 개발자 대신 해결해준다. 개발자는 컬렉션에 객체를 저장하듯이JPA에게 객체를 저장하면 된다.

ex)

**저 장**

jpa.persist(album);

🡺INSERT INTO ITEM …

INSERT INTO ALBUM …

**조 회**

Album album = jpa.find(Album.class, albumId);

🡺 SELECT I.\*, A.\*

FROM ITEM i

JOIN ALBUM A ON I.ITEM\_ID = A.ITEM\_ID

**1.2.2 연관관계**

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

-반면에 테이블은 외래 키를 사용해서 다른 테이블과 연관관계를 가지고 조인을 사용해서 연관된 테이블을 조회한다.

-객체는 참조가 있는 방향으로만 조회할 수 있는 반면, 테이블은 외래 키 하나로 반대로도 조회 가능하다.

**잘못된 방법**

class Member {  
 String id; // MEMBER\_ID 컬럼 사용  
 Long teamId; // TEAM\_ID FK 컬럼 사용  
 String username; // USERNAME 컬럼 사용  
}  
  
class Team {  
 Long id; // TEAM\_ID PK 사용  
 String name; // NAME 컬럼 사용  
}

🡺Member 객체와 연관된 Team 객체를 참조를 통해서 조회할 수 없다.

**정상적인 방법**

class Member {  
 String id; // MEMBER\_ID 컬럼 사용  
 Team team; // 참조로 연관관계를 맺는다.  
 String username; // USERNAME 컬럼 사용  
}  
  
class Team {  
 Long id; // TEAM\_ID PK 사용  
 String name; // NAME 컬럼 사용  
}

🡺객체를 테이블에 저장하거나 조회하기 쉽지 않다. (테이블은 외래키가 있어야 된다.)

**JPA와 연관관계**

-JPA는 연관관계와 관련된 패러다임의 불일치 문제를 해결해준다.

🡺member.setTeam(team); // 회원과 팀 연관관계 설정

jpa.persist(member); // 회원과 연관관계 함께 저장

**1.2.3 객체 그래프 탐색**

-참조를 사용해서 연관된 객체를 찾는 것

-SQL을 직접 다루면 처음 실행하는 SQL에 따라 객체 그래프를 어디까지 탐색할 수 있는지 정해진다.

-객체 그래프 탐색은 코드만 보고 어디까지 할 수 있는지 예측할 수 없다. DAO를 열어서 SQL을 직접 확인해야 한다. (엔티티가 SQL 종속)

🡺연관된 모든 객체 그래프를 조회하면? 메모리 부담

**JPA와 객체 그래프 탐색**

-JPA를 사용하면 **지연 로딩**을 사용해서 신뢰된 객체를 마음껏 조회 가능

Order order = member.getOrder();  
order.getOrderDate(); // Order를 사용하는 시점에 SELECT ORDER SQL

🡺실제 객체를 불러 올 때 자주 사용되나 여부에 따라 PreLoading, LazyLoading 전략을 취한다.

**1.2.4 비교**

-동일성 비교 (==) : 인스턴스의 주소 값 비교

-동등성 비교 (equals) : 객체 내부의 값 비교

-SQL 조회

String memberId = "100";  
Member member1 = memberDAO.getMember(memberId);  
Member member2 = memberDAO.getMember(memberId);  
  
member1 == member2 // 다르다

🡺컬렉션이라면? 같았을텐데… 패러다임의 불일치

-JPA 조회

String memberId = "100";  
Member member1 = jpa.find(Member.class, memberId);  
Member member2 = jpa.find(Member.class, memberId);  
  
member1 == member2 // 같다

🡺동일하다.

**1.3 JPA란 무엇인가?**

-JPA는 자바 진영의 ORM 기술 표준



ORM은?

-객체와 관계형 데이터베이스 매핑

-ORM은 단순히 SQL을 개발자 대신 생성해서 데이터베이스에 전달할 뿐만 아니라 다양한 패러다임의 불일치까지 해결해준다.

**1.3.1 JPA 소개**

-EJB3.0에서 하이버네이트를 기반으로 새로운 자바 ORM 기술 표준이 만들어졌는데 이것이 JPA다.

-간략 버전 특징

1) JPA 1.0 (2006년) : 초기 버전. 복합 키와 연관관계 기능 부족

2) JPA 2.0 (2009년) : 대부분의 ORM 기능 포함. JPA Criteria가 추가됨

3) JPA 3.0 (2013년) : SP, 컨버터, 엔티티 그래프 기능등이 추가

**1.3.2 왜 JPA를 사용해야 하는가?**

생산성

-자바 컬렉션에 객체를 저장하듯이 JPA에게 저장할 객체를 전달하면 된다.

-지루하고 반복적인 코드와 CRUD용 SQL을 개발자가 직접 작성하지 않아도 된다.

유지보수

-번거로운 부분을 JPA가 대신 처리해주므로 수정해야 할 코드가 줄어든다.

-페러다임의 불일치도 해결해주므로 유지보수하기 좋은 도메인 모델을 편리하게 설계가능

패러다임의 불일치 해결

-JPA는 상속, 연관관계, 객체 그래프 탐색, 비교하기와 같은 패러다임의 불일치 문제를 해결해준다.

성능

String memberId = "100";  
Member member1 = jpa.find(Member.class, memberId);  
Member member2 = jpa.find(Member.class, memberId);

-트랜잭션 안에서 두번째 조회 시, 첫번째 조회결과를 재사용한다.

데이터 접근 추상화와 벤더 독립성

-관계형 데이터 베이스는 벤더마다 사용법이 다른 경우가 많다.

🡺기술에 종속됨

-JPA는 추상화 계층을 제공해서 데이터베이스 기술에 종속되지 않도록 한다.

🡺연결 정보만 바꾸면 된다.

**CHAPTER2**