**1.1 SQL을 직접 다룰 때 발생하는 문제점**

자바로 작성한 애플리케이션은 JDBC API를 이용해서 SQL을 데이터 베이스에 전달한다.

JDBC API와 SQL

데이터 베이스는 객체구조와는 다른 데이터 중심의 구조를 가지므로 개발자가 객체 지향 애플리케이션과 데이터 베이스 중간에서 SQL과 JDBC API를 이용해서 변환 작업을 직접 해주어야 한다.

문제는 객체를 데이터베이스에 CRUD하려면 너무 많은 SQL과 JDBC API 코드로 작성해야 한다는 점이다. 그리고 테이블 마다 이런 비슷한 일을 반복해야 한다.

애플리케이션에서 SQL을 직접 다룰 때 발생하는 문제점을 요약하면 다음과 같다.

• 진정한 의미의 계층 분할이 어렵다.

• 엔티티를 신뢰할 수 없다.

• SQL에 의존적인 개발을 피하기 어렵다..

**JPA와 문제 해결**

JPA를 이용하면 객체를 데이터베이스에 저장하고 관리할 때, 개발자가 직접 SQL을 작성하는 것이 아니라 JPA가 제공하는 API를 사용하면 된다.

그러면 JPA가 개발자 대신에 적절한 SQL을 생성해서 데이터베이스에 전달한다.

JPA가 제공하는 CRUD API는 간단히 아래와 같다.

**저장 기능**

jpa.persist(member) // 저장

JPA가 객체와 매핑정보를 보고 적절한 INSERT SQL을 생성해서 데이터베이스에 전달한다.

**조회 기능**

String memberId = "helloId";

Member member = jpa.find(Member.class, memberId); // 조회

JPA가 객체와 매핑정보를 보고 적절한 SELECT SQL을 생성해서 데이터 베이스에 전달하고 그 결과로 Member 객체를 생성해서 반환한다.

**수정 기능**

Member member = jpa.find(Member.class, memberId);

member.setName("이름변경"); // 수정

JPA는 별도의 수정 메소드를 제공하지 않는 대신 객체를 조회해서 값을 변경하면 트랜잭션을 커밋할 때 데이터베이스에 적절한 UPDATE SQL이 전달 된다.

**연관된 객체 조회**

Member member = jpa.find(Member.class, memberId);

Team team = member.getTeam(); // 연관된 객체 조회

JPA는연관된 객체를 사용하는 시점에 적절한 SELECT SQL을 실행한다.

**1.2 패러다임의 불일치**

애플리케이션은 자바라는 객체지향 언어로 개발하고 데이터는 관계형 데이터베이스에 저장해야 한다면, 패러다임의 불일치 문제를 개발자가 중간에서 해결해야 하면서 많은 시간과 코드를 소비한다.

자바 진영에서 이러한 패러다임의 불일치 문제를 해결하기 위해 많은 노력을 기울였고 그 결과물이 바로 JPA이다.

**1.3 JPA란**

**JPA란 무엇인가?**

JPA(Java Persistence API)는 자바 진영의 ORM 기술 표준이다.

아래 그림처럼 애플리케이션과 JDBC 사이에서 동작한다.

**ORM이란 무엇일까?**

ORM(Object-Relational Mapping)은 이름 그대로 객체와 관계형 데이터베이스를 매핑한다는 뜻이다.

ORM 프레임워크는 객체와 테이블을 매핑해서 패러다임의 불일치 문제를 개발자 대신 해결해준다.

ORM 프레임워크는 단순히 SQL을 개발자 대신 생성해서 데이터베이스에 전달 해주는 것뿐만 아니라 앞서 이야기한 다양한 패러다임의 불일치 문제들도 해결해준다.

따라서 객체 측면에서는 정교한 객체 모델링을 할 수 있고 관계형 데이터 베이스는 데이터베이스에 맞도록 모델링하면 된다. 그리고 그 둘을 어떻게 매핑해야 하는지 매핑 방법만 ORM 프레임워크에 알려주면 된다. 덕분에 개발자는 데이터 중심인 관계형 데이터베이스를 사용해도 객체지향 애플리케이션 개발에 집중 할 수 있다.

자바 진영에는 다양한 ORM 프레임워크들이 있는데 그중에 하이버네이트 프레임워크가 가장 많이 사용된다.

JPA 표준 인터페이스와 구현체

과거 자바 진영은 엔터프라이즈 자바 빈즈(EJB)라는 기술 표준을 만들었는데 그 안에는 엔티티 빈이라는 ORM 기술도 포함되어 있었다.

하지만 너무 복잡하고 기술 성숙도도 떨어졌으며 자바 엔터프라이즈(J2EE) 애플리케이션 서버에서만 동작했다.

이때 하이버네이트([hibernate.org](http://hibernate.org/))라는 오픈소스 ORM 프레임워크가 등장했는데 EJB의 ORM기술과 비교해서 가볍고 실용적인 데다 기술 성숙도도 높았다.

또한 자바 엔터프라이즈 애플리케이션 서버 없이도 동작해서 많은 개발자가 사용하기 시작했다.

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

**JPA는 자바 ORM 기술에 대한 API 표준 명세이다. (인터페이스를 모아둔 것-!)**

따라서 JPA를 사용하려면 JPA를 구현한 ORM 프레임워크를 선택해야 한다.

**왜 JPA를 사용해야 하는가?**

**생산성**

JPA를 사용하면 자바컬렉션에 객체를 저장하듯이 JPA에게 저장할 객체를 전달하면된다.

INSERT SQL을 작성하고 JDBC API를 사용하는 지루하고 반복적인 코드와 CRUD용 SQL을 개발자가 직접 작성하지 않아도 된다.

**유지보수**

SQL을 직접 다루면 엩티티에 필드를 하나만 추가해도 관련된 SQL과 결과를 매핑하기 위한 JDBC API 코드를 모두 변경해야 했다.

반면에 JPA를 사용하면 이런 과정을 JPA가 대신 처리해주므로 필드를 추가하거나 삭제해도 수정해야 할 코드가 줄어든다.

**패러다임 불일치 해결**

JPA는 상속, 연관관계, 객체 그래프 탐색, 비교하기와 같은 패러다임의 불일치 문제를 해결해 준다.

**성능**

JPA는 애플리케이션과 데이터베이스 사이에서 다양한 성능 최적화 기회를 제공 한다.

String memberId = "helloId";

Member merber1 = jpa.find(memberId);

Member member2 = jpa.find(memberId);

이것은 같은 트랜잭션 안에서 같은 회원을 두 번 조회하는 코드의 일부분이다.

JDBC API를 사용해서 해당 코드를 직접 작성했다면 회원을 조회 할 때마다 SELECT SQL을 사용해서 데이터베이스와 두 번 통신했을 것이다.

JPA를 사용하면 회원을 조회하는 SELECT SQL을 한번만 데이터베이스에 전달하고 두 번째는 조회한 객체를 재사용한다.

**데이터 접근 추상화와 벤더 독립성**

관계형 데이터베이스는 같은 기능도 벤더마다 사용법이 다른 경우가 많다.

단적인 예로 페이징 처리는 데이터베이스 마다 달라서 사용법을 각각 배워야 한다.

결국 애플리케이션은 처음 선택한 데이터베이스 기술에 종속되고 다른 데이터베이스로 변경하기는 매우 어렵다.

JPA는 아래 그림처럼 애플리케이션과 데이터베이스 사이에 추상화된 데이터 접근 계층을 제공해서 애플리케이션이 특정 데이터베이스 기술에 종속되지 않도록 한다.

만약 데이터 베이스를 변경하면 JPA에게 다른 데이터베이스를 사용한다고 알려주기만 하면 된다.

벤더 독립성

**표준**

JPA는 자바 진영의 ORM 기술 표준이다.

표준을 사용하면 다른 구현 기술로 손쉽게 변경할 수 있다.

**cf. ORM에 대한 궁금증과 오해**

**Q. 성능이 느리진 않나요?**

JPA는 다양한 성능 최적화 기능을 제공해서 잘 이해하고 사용하면 SQL을 직접 사용할 때 보다 더 좋은 성능을 낼 수도있습니다.

또한 JPA의 네이티브 SQL 기능을 사용해서 SQL을 직접 호출하는 것도 가능합니다.

**Q. 통계 쿼리처럼 매우 복잡한 SQL은 어떻게 하나요?**

JPA는 통계 쿼리 같이 복잡한 쿼리보다는 실시간 처리용 쿼리에 더 최적화 되어 있습니다.

JPA가 제공하는 네이티브 SQL을 사용하거나 마이바티스나 스프링의 JdbcTemplate 같은 SQL 매퍼 형태의 프레임워크를 혼용하는 것도 좋은 방법입니다.

**Q. 마이바티스와 어떤 차이가 있나요?**

마이바티스나 스프링 JdbcTemplate을 보통 SQL 매퍼라 합니다.

이것은 객체와 SQL을 매핑합니다. 따라서 SQL과 매핑할 객체만 지정하면 반복되는 JDBC API 사용과 응답 결과를 객체로 매핑하는 일은 SQL 매퍼가 대신 처리해 줍니다.

이런 SQL 매퍼가 편리하긴 하지만 결국 개발자가 SQL을 직접 작성해야 하므로 SQL에 의존하는 개발을 피할 수 없습니다.

반면에 ORM은 객체와 테이블을 매핑만하면 ORM 프레임워크가 SQL을 만들어서 데이터베이스와 관련된 처리를 해주므로 SQL에 의존하는 개발을 피할 수 있습니다.