1장. JPA 소개

1.1 SQL을 직접 다룰 때 발생하는 문제점

자바로 작성한 애플리케이션은 JDBC API를 사용해서 SQL을 데이터베이스에 전달하는게 거의 일반적이다.

내가 개발을 했을 때도, 회원에 대한 CRUD를 진행할 때 Member라는 객체(DTO)를 만들고 Mybatis로 SQL 쿼리를 통해 데이터베이스에 접근하였다. 처음에는 헷갈렸지만, 몇번 하다보니 계속 비슷한 형태로 객체를 생성하고 SQL문을 날리게 된다.

어떤 Table이든 CRUD는 존재하기 때문?이다.

나는 감을 잡았고, 어떤 객체를 만들건 똑같겠네 하며 코딩을 했었는데, 여기서는

‘비슷한 일을 반복하여 데이터 접근 계층(DAO)을 개발하는 일은 지루함과 반복의 연속이다.’라고 표현을 했다. 초심자한테는 비슷한 코드로 DB에 접근하며 원하는 데이터를 얻는게 재밌었을지 몰라도 이게 계속 반복되다보면 개발을 통해 무언가 추가되고 DB 스키마가 변경되면 관련된 SQL문까지 다 수정해야한다는 단점이 있으며 이를 다시 하나하나 테스트 해야 한다는 가장 큰 문제가 있었다.

이 부분은 모두가 느꼈을 부분이라 많이 와닿았다. DB 스키마를 변경하고 잘 적용 되었나 확인 하면 null값이 들어있는.. 이런 문제점을 책에서 아주 잘 언급해주었으며 나는 생각을 못하고 SQL문을 같이 수정해주었는데. DAO 객체에 필드를 추가했을 때 SQL도 같이 수정해줘야 하는 SQL과 아주 강한 의존성을 가지고 있다는 점이었다.

정리를 하면,

SQL을 직접 다룰 때 우리는 Mybatis 같은 방식을 이용해 DB(신뢰할 수 있고 안전한 데이터)에 접근해서 원하는 쿼리를 날렸다. 이러한 부분을 데이터 접근 계층을 만들어 계층 분할을 했다라고 했지만 DAO를 열어보면 어떤 SQL이 실행되는지 알 수 있다.

즉, 진정한 의미의 계층 분할이 아니며, DAO 객체에 필드를 하나 추가하면 최소 CRUD 코드와 SQL문을 변경해야 한다. 이는 곧 SQL과 매우 강한 의존성이 있다는 문제점이 있다.

—-> 이런 문제를 JPA는 해결해 준다.

JPA는 개발자가 직접 SQL을 작성하는게 아니라 JPA가 제공하는 API를 사용하면 된다.

즉, 테이블에 매핑할 객체를 전달하여 관리할 수 있게 해준다고 한다.

이렇게 되면 객체에 매핑하기 때문에, DB 스키마가 변경되어도 DAO 객체의 필드를 수정해주어 매핑하면 그냥 그대로 저장된다…

이런 방식을 통해 이젠 객체 모델링 방식을 할 수 있게해준다라는 말을 어디선가 본 적이 있는것 같기도 하다.

1.2 패러다임의 불일치

우리는 애플리케이션을 개발할 때 DAO같은 객체를 만들어서 데이터베이스에 데이터를 저장하고 각종 쿼리를 통해 가공?했다.

그런데 객체는 객체지향에서 이야기하는 추상화, 상속, 다형성 같은 개념이 있으며

데이터베이스는 데이터 중심으로 구조화 되어있으며 객체지향의 개념이 없다.

이러한 패러다임의 불일치를 개발자가 중간에서 해결해야하는 문제점이 있으며,

JPA를 사용하면 이러한 불일치 문제를 어떻게 해결해주는지 설명해주며

객체지향적으로 개발을 할 수 있게 도와준다.

1. JPA와 상속

데이터베이스에서의 상속은 객체의 상속과는 다르다.

따라서, 상속을 한 객체를 DB에 저장하기 위해서는 부모와 자식 객체를 분리한 후 두개의 테이블로 저장을 해야하며 조회를 하기 위해서도 두 테이블을 JOIN하여야 한다. 이러한 부분이 패러다임 불일치에 대한 비용이라고 한다.

JPA는 자바 컬렉션에 객체를 저장하듯 상속을 한 객체를 persist 메소드로 저장하면 된다고 한다. 알아서 두개의 테이블에 나누어 저장 해주며, 조회또한 알아서 두 테이블을 조인해서 필요한 데이터를 조회해 준다.

1. 연관관계

객체는 참조를 통해 다른 객체와 연관관계를 가지지만, 데이터베이스는 외래 키를 통해 연관관계를 가지며 이 패러다임의 불일치가 가장 극복하기 어렵다고 한다.

객체는 참조할 객체가 있으면 되고, 데이터베이스는 참조할 테이블의 외래 키만 있으면 된다.

따라서 객체에 참조할 객체를 보관하고, 참조할 객체의 필드에 외래 키가 있으므로 그것을 사용하여 테이블간 연관관계를 맺는것이 패러다임 불일치 비용이다.

JPA를 사용하면, 연간관계를 맺어 객체를 저장하고 그 객체를 JPA에 등록하여 저장하면, 참조한 객체를 외래 키로 변환하여 INSERT SQL을 DB에 전달해준다.

불일치에 대한 비용을 개발자가 아니라 JPA가 처리해 준다는 것이다.

1. 객체 그래프 탐색

객체의 연관관계를 그래프로 표현하고 참조를 사용해 연관된 객체를 찾는것을 객체

그래프 탐색이라고 한다. 객체는 객체 그래프의 어떤 객체든 탐색하여 찾을 수 있어야 한다.

하지만 데이터베이스에선 처음 실행하는 SQL에 따라 탐색할 수 있는 범위가 정해진다.(JOIN의 범위에 따라 탐색하여 참조할 수 있는 객체의 범위가 정해진다.)

우리가 무엇을 탐색하고 싶냐에따라 여러개의 메소드를 정의해놔서 골라서 써야 한다는 비용이 발생한다.

JPA를 사용하면 객체 그래프를 마음껏 탐색할 수 있게 한다.

어쨌든 우리는 연관된 객체를 사용하고 싶을때 SELECT하는 SQL문을 활용해야하는데, JPA는 실제 객체를 사용하는 시점(객체를 불러오고 객체의 메소드를 사용하는 시점 등)까지 데이터베이스 조회를 미루는 지연 로딩도 지원하고, 연관된 객체를 함께 조회해서 즉시 불러와 사용하는 두가지 방식을 지원한다.

1. 비교

객체에서의 비교는 동일성 비교와 동등성 비교가 있다.

동일성은 ==비교로 객체 인스턴스의 ID가 같은지를 보는 것이고,

동등성 비교는 equals 객체의 내용을 보는 것이다.

SQL 문을 통해 같은 테이블을 읽어와 생성한 객체는 동일성을 만족하지 못한다.

각 다른 객체로 데이터베이스에서 읽어온 데이터를 채워서 생성하기 때문이다.

개인적인 생각은 우리가 원한것은 원하는 레코드를 읽어왔을 때 객체 하나로 동일하게 받아야 관리하기 쉽고 메모리 낭비가 없을 것이다. 그런데 이렇게 설정하는 과정이 매우 복잡해 비용이 크다고 한다, 이러한 기능을 어노테이션을 통해 싱글톤으로 관리해주던 스프링의 매력을 느꼈었고, 동일 레코드에 대한 객체 동일성 마저 JPA에서 해준다는 것이 매우 편리하다라고 생각했다. 다른 것을 다 빼고 동일 객체에 대한 관리만으로도 JPA 사용의 이유가 있다고 생각한다.

1.3 JPA란 무엇인가?

책에서 JPA는 자바 ORM 기술에 대한 API 표준 명세다. 라고 표현했다.

인터페이스를 모아둔 것으로 JPA를 구현한 프레임워크를 선택해서 사용해야 하는데 이것이 하이버네이트 프레임워크 같은 것을 말하고 있었다.

한마디로 자바 ORM 기술을 일반적이고 편하게 사용하려면 하이버네이트 같이 JPA를 구현한 프레임워크를 사용하면 된다는 것이다.

1.3.2 왜 JPA를 사용해야 하는가?

위의 절에서 다뤘던 내용을 정리

생산성

* 자바 컬렉션에 객체를 저장하듯이 JPA에게 저장할 객체를 전달하면 되므로 지루한 CRUD SQL문을 개발자가 직접 작성하지 않아도된다.

유지보수

* SQL을 직접 다루면 엔티티에 필드를 하나만 추가해도 관련된 SQL문을 수정 및 결과 매핑이 잘 되었는지 테스트 해야하는데 JPA가 대신 처리해 준다.

패러다임의 불일치 해결

* JPA는 상속, 연관관계, 객체 그래프 탐색, 비교와 같이 객체와 데이터베이스의 패러다임 불일치를 해결해준다.

성능

* JPA는 애플리케이션과 데이터베이스 사이에서 성능 최적화 기회를 제공한다.

같은 회원을 연속해서 조회한다면 JDBC API를 이용하면 SELECT SQL이 두 번 날라가 데이터베이스를 두 번 다녀올 것이다. 그런데 JPA는 한번 통신만으로 같은 객체를 반환해준다. -> 이를 JPA의 캐싱기능이라고도 한다.

그냥 JPA를 사용하는 것만으로도 성능이 좋아지며 상황에 따라 커스텀하게 최적화 시킬 수 있는 방법이 있는 것 같다.

데이터 접근 추상화와 벤더 독립성

* JDBC API를 통해 개발을 하면 어떤 벤더를 사용하느냐에 따라 사용법이 종속된다.

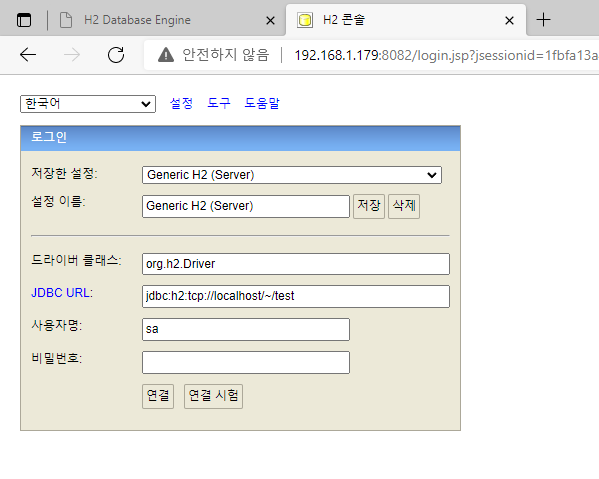
하지만 JPA를 쓰게되면 로컬에서 개발할 땐 H2 데이터베이스를 써도 나중에 개발이나 상용 환경은 MySQL을 사용하겠다고 알려주기만 하면 변경이 된다.

2장. JPA 시작

교재에서 주어진 GitHub 주소를 Clone하여 작업하였다.

<https://github.com/holyeye/jpabook>

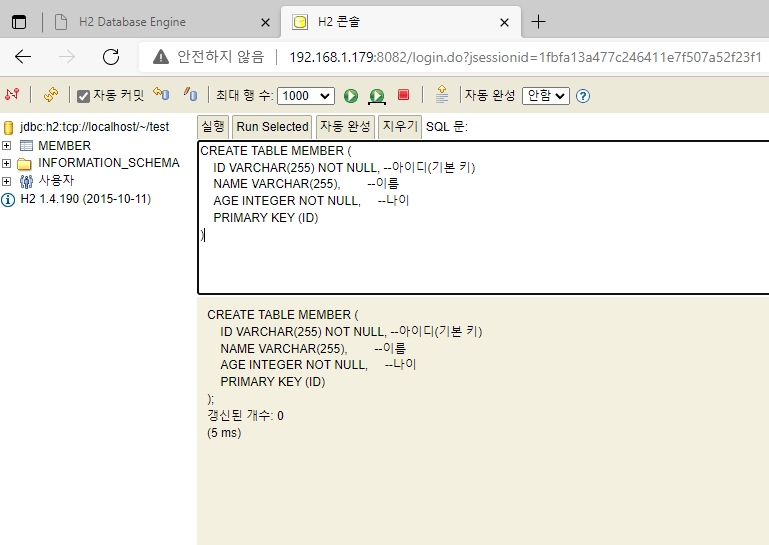
2.2 H2 데이터베이스 설치



H2 데이터베이스를 1.4.190으로 설치하고 H2/bin/h2 배치파일을 실행하면 데이터베이스를 연결할 수 있다. ( 책에서는 1.4.187을 이용하지만 현재 다운로드 불가..)

깃허브에서 Window 버전으로 DownLoad하는 방법에 대한 가이드를 소개해주었다.

<https://medium.com/@oopchoi/jpa-%ED%94%84%EB%A1%9C%EA%B7%B8%EB%9E%98%EB%B0%8D-fc443b647ec8>

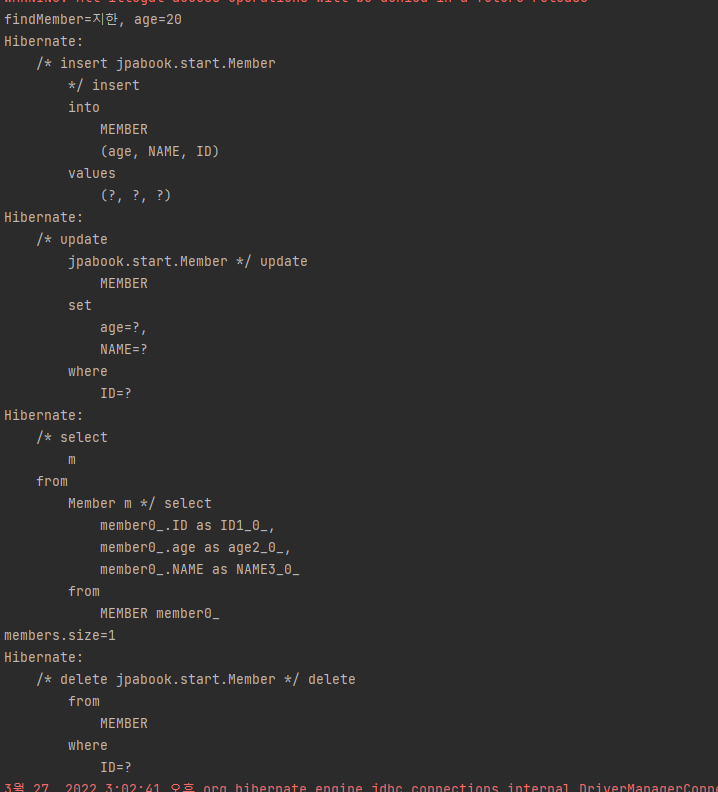


git에서 clone한 ch02-jpa-start1에서 member.sql을 실행해 MEMBER 테이블을 생성하였다.

2.3 라이브러리와 프로젝트 구조

pom.xml에 JPA 프레임워크로 하이버네이트 의존성을, 사용할 DB인 H2 의존성이 추가 되어있었다.



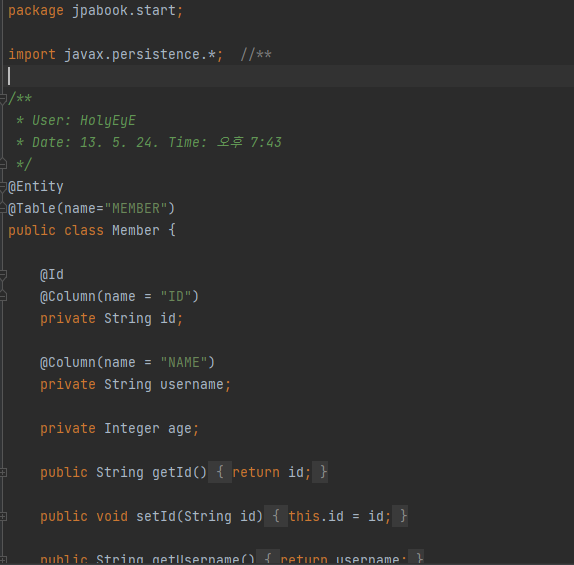


JpaMain.java를 실행해보았는데 일단은 잘 실행되는 것 같다.

2.4 객체 매핑 시작

H2 데이터베이스의 MEMBER테이블과 Member 객체를 매핑하는데 이를 위해

매핑 어노테이션을 활용한다.(매핑 어노테이션 패키지 : javax.persistence)



@Entity

이 클래스를 테이블과 매핑한다고 JPA에게 알려준다. 이렇게 Entity가 사용된 클래스를 엔티티 클래스라고 한다.

@Table

엔티티 클래스에 매핑할 테이블 정보를 알려준다. name 속성을 사용해서 Member 엔티티를 MEMBER 테이블에 매핑하였고, 이 어노테이션을 생략하면 클래스 이름을 테이블 이름으로 매핑한다.

@Id

엔티티 클래스의 필드를 테이블의 기본 키에 매핑한다. @Id가 사용된 필드를 식별자 필드라고 한다.

@Column

필드를 컬럼에 매핑한다.name 속성을 사용해서 Member 엔티티의 username필드를 MEMBER 테이블의 NAME에 매핑.

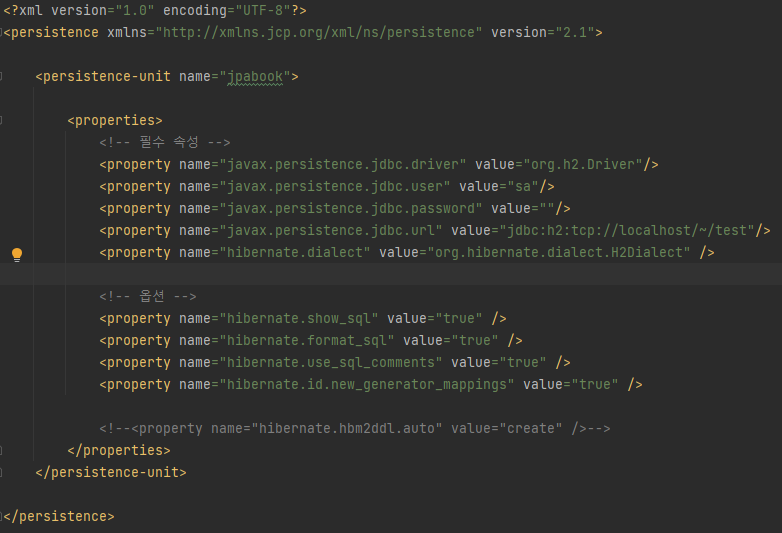
매핑 정보가 없는 필드

age 필드에는 매핑 어노테이션이 없는데, 이렇게 되면 필드명을 사용해서 컬럼명으로 매핑한다.

여기서는 대소문자를 구분하지 않는다고 가정했지만, 대소문자를 구분하는 데이터베이스를 사용하면 어노테이션을 활용해 명시적으로 매핑해야 한다. ( 필드명으로 대소문자를 구분해서 쓰는 것은 일관된 표기법에 어긋날 것 같아 어노테이션을 활용하는게 좋을 것 같다.)

2.5 persistence.xml 설정

JPA 설정 정보를 관리하는 xml이다. 이 설정 파일이 META-INF/persistence.xml 경로에 있으면 별도의 설정없이 JPA가 인식할 수 있다.

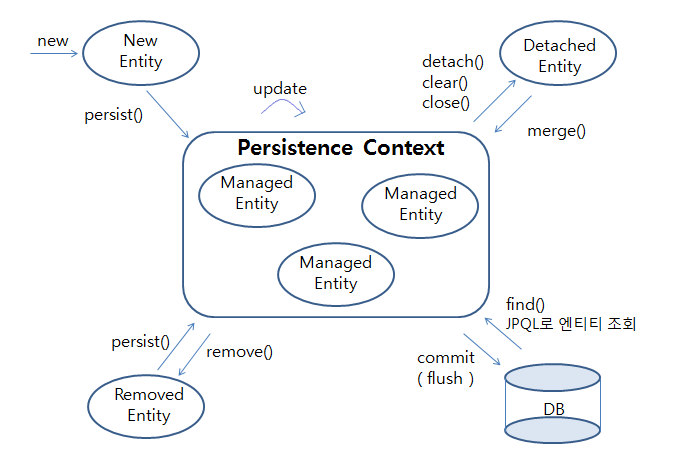


javax.persistence로 시작하는 속성은 JPA 표준 속성으로 특정 구현체에 종속되지 않고, hibernate로 시작하는 속성은 하이버네이트 전용 속성이므로 하이버네이트에서만 사용할 수 있다고 한다.

자주 등장하는 persistence는 영속성이라는 뜻으로, 다음 장에서 배우는 것 같다.

쉽게 이해하려고 정리해보면 persistence.xml에 보면 persistence-unit의 name을 저장해서 JPA에서 관리하는 DB를 연결한 어떤 Context의 이름을 말한다고 생각할 수 있다.

영속성의 이름으로 유닛을 구분하며 그 안의 Context(가장 쉽게 Entity들)을 수정, 관리할 때 영구히 저장해준다. 라는 의미로 생각한다.



2.5.1 데이터베이스 방언

JPA는 특정 데이터베이스에 종속적이지 않아 데이터베이스를 언제는 바꿀 수 있다.

그런데, 데이터베이스 마다 SQL 문법과 함수가 조금씩 다른데 이를 데이터베이스 방언이라고 하며 JPA 구현체인 하이버네이트에서 데이터베이스 방언 클래스를 제공해주며 이를 설정하는 문장인 것 같다.



예를 들면, 가변 문자에 대해 MySQL은 VARCHAR, 오라클은 VARCHAR2를 사용하는데 이러한 차이를 방언 클래스를 제공해 각 데이터베이스마다 알아서 다른 함수를 사용해준다는 것이다.

2.6 애플리케이션 개발

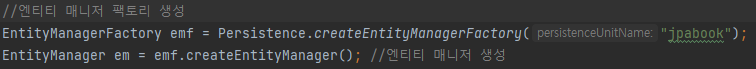
책에서 주어진 코드를 라인별로 살펴볼 것이다.

전체 코드는 다음을 참고. <https://github.com/holyeye/jpabook/blob/master/ch02-jpa-start1/src/main/java/jpabook/start/JpaMain.java>

전체 코드는 크게 3부위로 나누어지는데 하나하나 살펴볼 것이다.

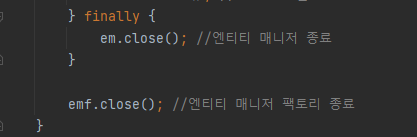
* 엔티티 매니저 설정

JPA를 시작하려면 우선 persistence.xml에서 설정했던 이름이 jpabook이었던 영속성 유닛을 엔티티 매니저 팩토리를 생성한 후, 엔티티 매니저 클래스를 생성한다.



엔티티 매니저 팩토리는 애플리케이션 전체에서 딱 한 번만 생성하고 공유해서 사용해야 한다. 그 이유는 엔티티 매니저 팩토리는 데이터베이스 간의 커넥션을 생성하면서, 데이터베이스에 따라 객체 풀을 생성하기도 해 비용이 매우 크기 때문이다.

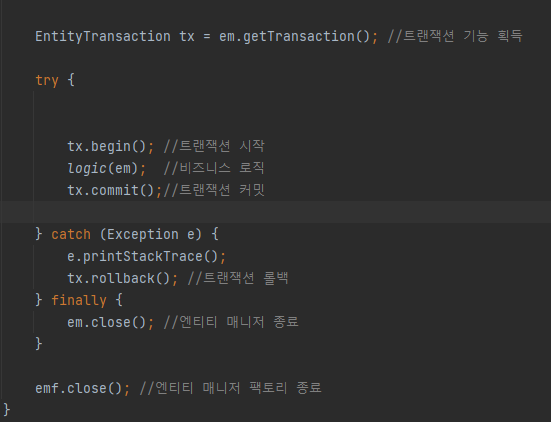
엔티티 매니저는 엔티티를 데이터베이스에 CRUD 등을 할 수 있고, 데이터베이스 커넥션과 밀접한 관계가 있는 객체로 스레드 간에 공유하거나 재사용하면 안 된다.



마지막으로 사용이 끝나면 엔티티 매니저와 매니저 팩토리 모두 close 해주어야 한다.

* 트랜잭션 관리

JPA를 사용하면 항상 트랜잭션 안에서 데이터를 변경해야 한다. 트랜잭션 없이 데이터를 변경하면 예외가 발생한다.



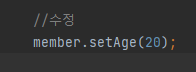
다음과 같이 트랜잭션을 엔티티 매니저에서 받아오며 logic같은 엔티티에 대한 조작을 할 때 시작시켜야 한다.

* 비즈니스 로직

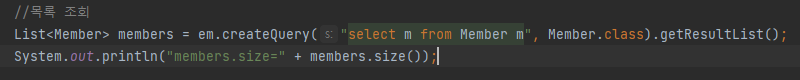


Member 객체를 생성하고, 엔티티 매니저에 등록을 하면 데이터베이스에 올라가는 형태이며 Member 객체의 매핑 어노테이션을 통해 매핑해서 저장한다고 보면 된다.

여기서 특이한 점은 수정을 하는 부분인데,

엔티티 매니저에 등록한 객체의 필드 값을 변경만 해주어도 UPDATE SQL을 생성해서 데이터베이스에 값을 변경한다고 한다.

2.6.4 JPQL

그렇다면 여러 회원을 조회하는 쿼리에 대해서는 어떻게 처리를 하는가?

Mybatis에서는 Result Map을 통해 Map으로 매핑된 객체를 리스트에 저장을 하는 형식이었는데 JPA를 사용하면 getResultList() 메소드를 호출하면 List로 받아준다.

그냥 받으면 된다. 즉, JPA에서 객체 중심 개발을 할 수 있도록 알아서 엔티티 객체로 반환해준다. 그냥 객체로 다 처리를 해준다.