# JPA_Chapter2

| Column | @2021년 12월 5일 오후 2:32 |
| --- | --- |
| Tags | |



기초적인 설정은 넘김

## 객체 매핑

```
@Entity
@Table(name="MEMBER")
public class Member {

    @Id
    @Column(name = "ID")
    private String id;

    @Column(name = "NAME")
    private String username;

    private Integer age;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
```

```
        this.username = username;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

## JPA Mapping Annotation

현재 MySuni는 `DomainEntity`, `DramaEntity`로 하기의 어노테이션 대부분을 숨기기 때문에 별도로 정리한다.

- `@Entity`

  > 이 클래스를 테이블과 매핑하낟고 JPA에게 알려준다.
  >
  > 이 어노테이션이 사용된 클래스를 엔티티 클래스라고 한다.

- `@Table`

  > 엔티티 클래스에 매핑할 테이블 정보를 알려준다.
  >
  > name속성으로 테이블 명을 설정하거나, 이 어노테이션을 생략해 클래스 명으로 테이블 명을 설정할 수 있다.

- `@Id`

  > 엔티티 틀래스의 필드를 테이블의 Primary key에 매핑
  >
  > 이 어노테이션이 사용된 필드를 식별자 필드라 한다.

- `@Column`

  > 필드를 컬럼에 매핑

> `@Table`과 명명 규칙은 같다.
>
> 만약 대소문자를 구분하는 데이터베이스를 사용한다면 헷갈리지 않도록 반드시 명시해줄 것

# Persistence.xml 설정

> Gradle설정과 비교할 것

▼ 설정 코드

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.1">

    <persistence-unit name="jpabook">

        <properties>

            <!-- 필수 속성 -->
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.user" value="sa"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/test"/>
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />

            <!-- 옵션 -->
            <property name="hibernate.show_sql" value="true" />
            <property name="hibernate.format_sql" value="true" />
            <property name="hibernate.use_sql_comments" value="true" />
            <property name="hibernate.id.new_generator_mappings" value="true" />

            <!--<property name="hibernate.hbm2ddl.auto" value="create" />-->
        </properties>
    </persistence-unit>

</persistence>
```

## 데이터베이스 방언

> JPA는 특정 데이터베이스에 종속적이지 않고 다른 데이터 베이스로 손쉽게 교체할 수 있다.

> 그런데 각 데이터베이스마다 제공하는 문법, 함수가 조금씩 다른 문제점이 있느데, 이를 해결하기 위해 방언(Dialect)를 사용한다.
>
> ⇒ 데이터베이스가 변경되어도 애플리케이션 코드를 변경할 필요 없이 데이터베이스 방언만 교체해주면 된다.  H2dialect 참고 - 정리필요 → MySuni에서는 H2용으로 변경해주려고 설정했던 적이 있던 것 같다.

# 애플리케이션 개발

```
public class JpaMain {

    public static void main(String[] args) {

        //엔티티 매니저 팩토리 생성
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpabook");
        EntityManager em = emf.createEntityManager(); //엔티티 매니저 생성

        EntityTransaction tx = em.getTransaction(); //트랜잭션 기능 획득

        try {


            tx.begin(); //트랜잭션 시작
            logic(em);  //비즈니스 로직
            tx.commit();//트랜잭션 커밋

        } catch (Exception e) {
            e.printStackTrace();
            tx.rollback(); //트랜잭션 롤백
        } finally {
            em.close(); //엔티티 매니저 종료
        }

        emf.close(); //엔티티 매니저 팩토리 종료
    }

    public static void logic(EntityManager em) {

        String id = "id1";
        Member member = new Member();
        member.setId(id);
        member.setUsername("지한");
        member.setAge(2);

        //등록
        em.persist(member);

        //수정
        member.setAge(20);
```

```
        //한 건 조회
        Member findMember = em.find(Member.class, id);
        System.out.println("findMember=" + findMember.getUsername() + ", age=" + findMember.getAge());

        //목록 조회
        List<Member> members = em.createQuery("select m from Member m", Member.class).getResultList();
        System.out.println("members.size=" + members.size());

        //삭제
        em.remove(member);

    }
}
```

# 엔티티 매니저 설정

## 엔티티 매니저 팩토리 생성

> JPA를 사용할 수 있게 준비하는 첫 단계

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("jpabook");
```

위 코드로 `persistence.xml`에서 이름이 jpabook인 `persistence-unit`을 찾아 엔티티 매니저 팩토리를 생성한다. 이 때 `persistence.xml`를 찾아 JPA기반 객체 생성, 데이터베이스 커넥션 풀을 만드는 등 생성비용이 매우 크기 때문에 **엔티티 매니저 팩토리는 애플리케이션 전체에서 딱 한 번만 생성하고 공유해서 사용할 것**

## 엔티티 매니저 생성

```
EntityManager em = emf.createEntityManager();
```

- JPA의 기능 대부분은 엔티티 매니저가 제공한다.

  ⇒ 엔티티 매니저를 사용해서 엔티티를 데이터베이스에 등록/수정/삭제/조회 할 수 있다.
- 엔티티 매니저는 내부에 데이터베이스 커넥션을 유지하면서 데이터베이스와 통신한다.

  ⇒ 애플리케이션 개발자는 엔티티 매니저를 가상의 데이터베이스로 생각 할 수 있다.

💡 엔티티 매니저는 데이터베이스 커넥션과 밀접한 관계가 있으므로 스레드간에 공유, 재사용을 해선 안된다.

### 종료

```
em.close(); //엔티티 매니저 종료
emf.close(); //엔티티 매니저 팩토리 종료
```

---

## 트랜잭션 관리

> 비즈니스 로직이 정상동작하면 commit, 예외가 발생하면 rollback

```
EntityTransaction tx = em.getTransaction(); //트랜잭션 기능 획득

try {
    tx.begin(); //트랜잭션 시작
    logic(em);  //비즈니스 로직
    tx.commit();//트랜잭션 커밋

} catch (Exception e) {
    e.printStackTrace();
    tx.rollback(); //트랜잭션 롤백
}
```

---

## 로직

```
public static void logic(EntityManager em) {

    String id = "id1";
    Member member = new Member();
    member.setId(id);
    member.setUsername("지한");
    member.setAge(2);

    //등록
    em.persist(member);

    //수정
    member.setAge(20);

    //한 건 조회
    Member findMember = em.find(Member.class, id);
    System.out.println("findMember=" + findMember.getUsername() + ", age=" + findMember.getAge());

    //목록 조회
    List<Member> members = em.createQuery("select m from Member m", Member.class).getResultList();
```

```
    System.out.println("members.size=" + members.size());

    //삭제
    em.remove(member);

}
```

▼ CRUD 코드

- 등록

  → `persist` 메소드를 사용한다.

  ```
  em.persist(member);
  ```

- 수정

  → `setter`를 호출하듯이 사용한다.

  ```
  member.setAge(20);
  ```

- 삭제

  → `remove`메소드를 사용한다.

  ```
  em.remove(member);
  ```

- 한 건 조회

  → `find`사용

  ```
  Member findMember = em.find(Member.class, id);
  ```

# JPQL

```
//목록 조회
List<Member> members = em.createQuery("select m from Member m", Member.class).getResultList();
System.out.println("members.size=" + members.size());
```

JPA를 사용하면 개발자는 엔티티 객체를 중심으로 개발하고 데이테 베이스에 대한 처리는 JPA에 맡겨야 한다.

앞에서 한 CRUD에서는 SQL을 전혀 사용하지 않은 것이 그 예이다.

문제는 검색 쿼리인데, JPA는 엔티티 객체 중심 개발이므로 *검색도 테이블이 아닌 엔티티 객체를 대상으로 검색해야 한다.*

하지만 엔티티 객체를 대상으로 검색을 하려면 DB의 모든 데이터를 불러와 엔티티 객체로 변경한 다음 검색해야 하는데, 이는 불가능 하다. ⇒ 이를 위해서 검색 조건이 포함된 SQL을 사용해야 하는데 JPA는 이를 `JPQL(Java Persistence Query Language)`로 해결한다.

> JPQL은 거의 SQL과 유사한데 큰 차이점은 아래와 같다.
>
> - JPQL은 **엔티티 객체**를 대상으로 쿼리한다. → 클래스와 필드대상 쿼리
> - SQL은 DB테이블 대상 쿼리
> - JPQL은 대소문자를 구분하지만 SQL은 구분하지 않는다.

💡 JPQL은 데이터베이스 테이블을 알지 못한다.

JPA는 JPQL을 분석해서 적절한 SQL을 만들어 데이터베이스에서 데이터를 조회한다.