2장 JPA 시작

기간 : 21.12.01 ~ 21.12.08

64p~

2.2 H2 데이터베이스 설치 69p

- 자바가 설치되어 있어야 동작함
- http://www.h2database.com/html/main.html

- H2 설치 경로에서 h2.sh 실행
    - C:\Program Files (x86)\H2\bin 경로 → h2.bat 실행

2.3 라이브러리와 프로젝트 구조 71p

- hibernate-entitymanager
    - 하이버네이트가 JPA 구현체로 동작하도록 JPA 표준을 구현한 라이브러리
    - JPA 표준과 하이버네이트를 포함하는 라이브러리
    - hibernate-entitymanager를 라이브러리로 지정하면 아래 중요 라이브러리도 함께 내려받는다.
        - hibernate-core : 하이버네이트 라이브러리
        - hibernate-jpa-2.1-api : JPA 2.1 표준 API를 모아둔 라이브러리

2.3.1 메이븐과 사용 라이브러리 관리 72p

- 메이븐은 라이브러리를 관리해주는 도구

<pom.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>jpa-basic</groupId>
    <artifactId>ex1-hello-jpa</artifactId>
    <version>1.0.0</version>
    <dependencies>
        <!-- JPA 하이버네이트 -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>5.3.10.Final</version>
        </dependency>
        <!-- H2 데이터베이스 -->
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.199</version>
        </dependency>
    </dependencies>
</project>
```

2.4 객체 매핑 시작 74p

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

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

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

    // 매핑 정보가 없는 필드
    private Integer age;
```

@Entity

- 이 클래스를 테이블과 매핑한다고 JPA에게 알려준다.
- 이 어노테이션이 사용된 클래스를 엔티티 클래스라고 한다.

@Table

- 엔티티 클래스에 매핑할 테이블 정보를 알려준다.
- 위 예제에서는 name 속성을 사용해서 Member 엔티티를 MEMBER 테이블에 매핑
- 이 어노테이션을 생략하면 클래스 이름을 테이블 이름으로 매핑한다(엔티티 이름을 사용한다).

@Id

- 엔티티 클래스의 필드를 테이블의 기본 키에 매핑한다.
- 위 예제에서는 엔티티의 id 필드를 테이블의 ID 기본 키 컬럼에 매핑
- @Id 가 사용된 필드를 식별자 필드라고 한다.

@Column

- 필드를 컬럼에 매핑
- 위 예제에서는 name 속성을 사용해서 Member 엔티티의 username 필드를 MEMBER 테이블의 NAME 컬럼에 매핑했다.

매핑 정보가 없는 필드

- 매핑 어노테이션을 생략하면 필드명을 사용해서 컬럼명으로 매핑한다.
- 위 예제에서는 필드명이 age이므로 age 컬럼으로 매핑

2.5 persistence.xml 설정 77p

- JPA는 persistence.xml을 사용해서 필요한 설정 정보를 관리
- 이 설정 파일이 META-INF/persistence.xml 클래스 패스 경로에 있으면 별도의 설정 없이 JPA가 인식할 수 있다.

<persistence.xml>

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.2">

    <persistence-unit name="hello">
        <properties>
            <!-- 필수 속성 -->
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
            <property name="javax.persistence.jdbc.user" value="study_test"/>
            <property name="javax.persistence.jdbc.password" value=""/>
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:tcp://localhost/~/study_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.jdbc.batch_size" value="10"/>
            <!--<property name="hibernate.hbm2ddl.auto" value="create"/>-->
        </properties>
    </persistence-unit>
</persistence>
```

`<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" version="2.2">`

- XML 네임스페이스와 사용할 버전 지정
- JPA 2.1을 사용하려면 `xmlns`와 `version`을 명시하면 된다.

`<persistence-unit name="hello">`

- JPA 설정은 영속성 유닛(persistence-unit)이라는 것부터 시작
- 일반적으로 연결할 데이터베이스당 하나의 영속성 유닛을 등록
- 영속성 유닛에는 고유한 이름(name)을 부여해야한다.

<properties>

- **JPA 표준 속성**

  이름이 `javax.persistence` 로 시작하는 속성은 JPA 표준 속성으로 특정 구현체에 종속되지 않는다.

  - `javax.persistence.jdbc.driver` : JDBC 드라이버
  - `javax.persistence.jdbc.user` : 데이터베이스 접속 아이디
  - `javax.persistence.jdbc.password` : 데이터베이스 접속 비밀번호
  - `javax.persistence.jdbc.url` : 데이터베이스 접속 URL

- **하이버네이트 속성**

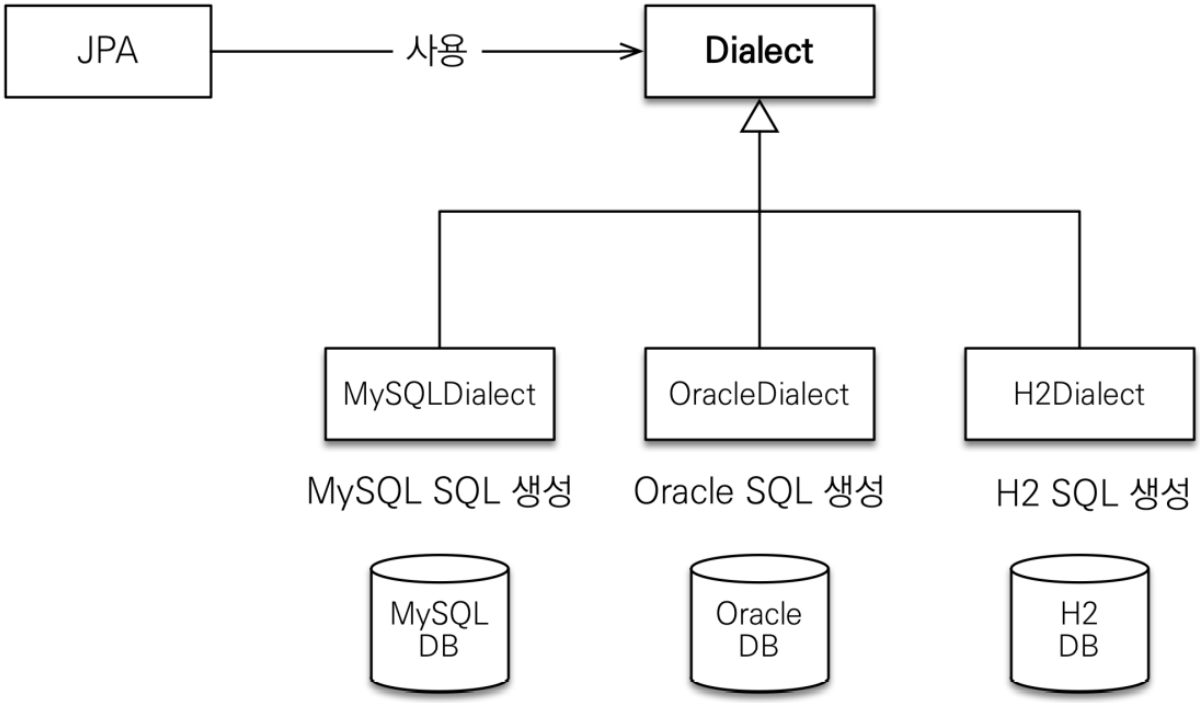
  `hibernate` 로 시작하는 속성은 하이버네이트 전용 속성으로 하이버네이트만 사용할 수 있다.

  - `hibernate.dialect` : 데이터베이스 방언(Dialect) 설정

2.5.1 데이터베이스 방언 79p

- JPA는 특정 데이터베이스에 종속적이지 않은 기술이다.
- 각 데이터베이스가 제공하는 SQL 문법과 함수가 조금씩 다른데 아래와 같은 차이점이 있다.
  - 데이터 타입 : 가변 문자 타입으로 MySQL은 VARCHAR, 오라클은 VARCHAR2를 사용
  - 다른 함수명 : 문자열을 가르는 함수로 SQL 표준은 SUBSTRING()을 사용하지만 오라클은 SUBSTR()을 사용
  - 페이징 처리 : MySQL은 LIMIT를 사용하지만 오라클은 ROWNUM을 사용

- SQL 표준을 지키지 않거나 특정 데이터베이스만의 고유한 기능을 JPA에서는 **방언(Dialect)**라고 한다.

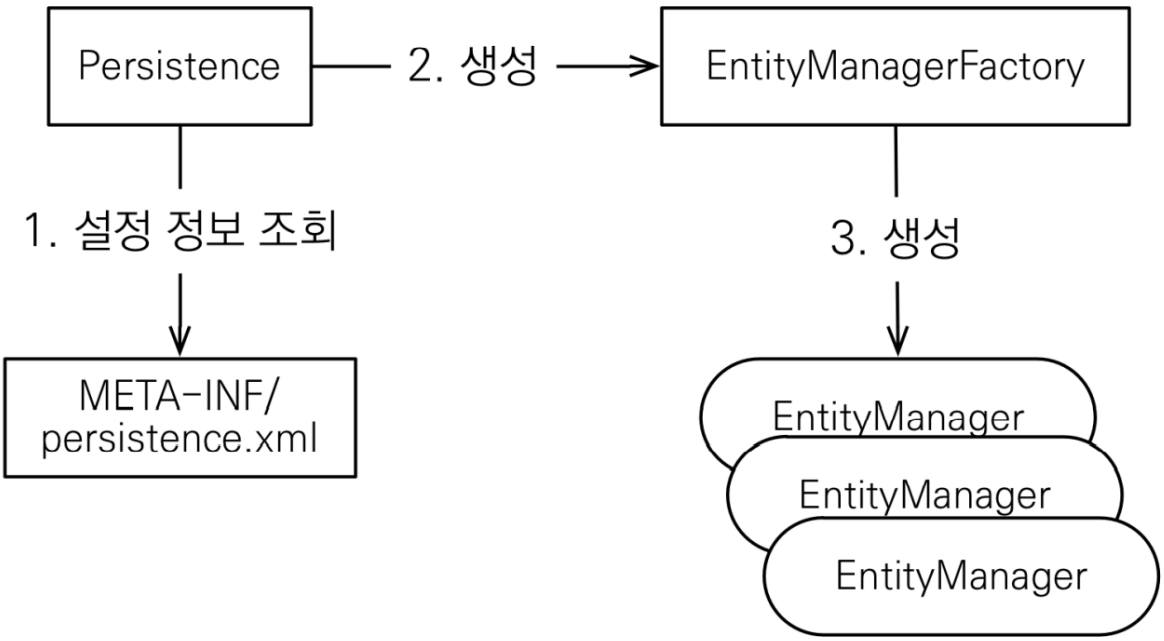


- 개발자는 JPA 가 제공하는 표준 문법에 맞추어 JPA를 사용하면 되고, 특정 데이터베이스에 의존적인 SQL 은 데이터베이스 방언이 처리해준다.
- 하이버네이트는 다양한 데이터베이스 방언을 제공한다. ⇒ hibernate.dialect 속성에 지정
  - **H2** : org.hibernate.dialect.H2Dialect
  - **Oracle 10g** : org.hibernate.dialect.Oracle10gDialect
  - **MySQL** : org.hibernate.dialect.MySQL5InnoDBDialect

2.6 애플리케이션 개발 81p

2.6.1 엔티티 매니저 설정

엔티티 매니저의 생성 과정



- 엔티티 매니저 팩토리(EntityManagerFactory) 생성
  - JPA를 시작하려면 우선 persistence.xml의 설정 정보를 사용해서 엔티티 매니저 팩토리를 생성해야 한다.
  - 이때 Persistence 클래스를 사용하는데 이 클래스는 엔티티 매니저 팩토리를 생성해서 JPA를 사용할 수 있게 준비한다.

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

  - META-INF/persistence.xml 에서 이름이 jpabook인 영속성 유닛을 찾아서 엔티티 매니저 팩토리를 생성
  - **엔티티 매니저 팩토리를 생성하는 비용이 아주 크기 때문에 애플리케이션 전체에서 딱 한 번만 생성하고 공유해서 사용해야 한다.**

- 엔티티 매니저 생성

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

  - 엔티티 매니저 팩토리에서 엔티티 매니저를 생성
  - 엔티티 매니저를 사용해서 엔티티를 데이터베이스에 등록/수정/삭제/조회할 수 있다.
  - 엔티티 매니저를 가상의 데이터베이스로 생각할 수 있다.
  - **엔티티 매니저는 데이터베이스 커넥션과 밀접한 관계가 있으므로 스레드간에 공유하거나 재사용하면 안 된다.**
- 종료
  - 사용이 끝난 엔티티 매니저는 반드시 종료해야 한다.

- 애플리케이션을 종료할 때 엔티티 매니저 팩토리도 종료해야 한다.

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

2.6.2 트랜잭션 관리 84p

- JPA를 사용하면 항상 트랜잭션 안에서 데이터를 변경해야 한다.
- 트랜잭션을 시작하려면 엔티티 매니저em 에서 트랜잭션 API를 받아와야 한다.

```
EntityTransaction tx = em.getTransaction();

try {

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

catch (Exception e) {
    tx.rollback(); // 예외 발생 시 트랜잭션 롤백
}
```

2.6.3 비즈니스 로직 84p

```
public static void logic(EntityManager em) {
    String id = "id1";
    Member member = new Member();
    member.setId(id0;
    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);
}
```

- 등록, 수정, 삭제, 조회 작업이 엔티티 매니저를 통해서 수행된다.
- 등록
  - 엔티티를 저장하려면 엔티티 매니저의 persist() 메서드에 저장할 엔티티를 넘겨주면 된다.
  - JPA는 회원 엔티티의 매핑 정보(어노테이션)를 분석해서 아래와 같은 SQL을 만들어 데이터베이스에 전달한다.

```
INSERT INTO MEMBER(ID, NAME, AGE) VALUES ('id1', '지한', 2)
```

- 수정
  - JPA는 어떤 엔티티가 변경되었는지 추적하는 기능을 갖추고 있다.
  - 따라서 member.setAge(20)처럼 엔티티의 값만 변경하면 아래와 같은 SQL을 생성해서 데이터 베이스에 값을 변경한다.

```
UPDATE MEMBER
   SET AGE = 20, NAME = '지한'
 WHERE ID = 'id1'
```

- 삭제
  - 엔티티를 삭제하려면 엔티티 매니저의 remove() 메소드에 삭제하려는 엔티티를 넘겨준다.

```
DELETE FROM MEMBER WHERE ID = 'id1'
```

- 한 건 조회
  - find() 메소드는 조회할 엔티티 타입과 @Id 로 데이터베이스 테이블의 기본 키와 매핑한 식별자 값으로 엔티티 하나를 조회한다.

```
SELECT * FROM MEMBER WHERE ID = 'id1'
```

2.6.4 JPQL 87p

- JPA는 엔티티 객체를 중심으로 개발하므로

  검색을 할 때도 테이블이 아닌 엔티티 객체를 대상으로 검색해야 한다.
- 모든 DB 데이터를 객체로 변환해서 검색하는 것은 불가능하다.

- 애플리케이션이 필요한 데이터만 데이터베이스에서 불러오려면 결국 검색 조건이 포함된 SQL을 사용해야 한다.
- JPA는 JPQL(Java Persistence Query Language)이라는 쿼리 언어로 이런 문제를 해결한다.
- JPA는 SQL을 추상화한 JPQL 이라는 객체지향 쿼리 언어를 제공한다.
- JPQL은 SQL과 문법이 거의 유사해서 SELECT, FROM, WHERE, GROUP BY, HAVING, JOIN 등을 사용할 수 있다.

> 💡 JPQL과 SQL 의 차이점
>
> JPQL
>
> - JPQL 은 **엔티티 객체**를 대상으로 쿼리한다.
> - 즉 클래스와 필드를 대상으로 쿼리한다.
>
> SQL
>
> - SQL은 **데이터베이스 테이블**을 대상을 쿼리한다.

- 위의 예제에서 `select m from Member m` 이 바로 JPQL이다.
- 여기서 from Member는 회원 엔티티 객체를 말하는 것이지 MEMBER 테이블이 아니다.
- JPQL은 데이터베이스 테이블을 전혀 알지 못한다.

- JPQL을 사용하려면
  1. `em.createQuery(JPQL, 반환타입)` 메소드를 실행
  2. 쿼리 객체를 생성한 후 쿼리 객체의 getResultList() 메소드를 호출

- JPA는 JPQL을 분석해서 아래와 같은 SQL을 만들어서 데이터베이스에서 데이터를 조회한다.

```
SELECT M.ID, M.NAME, M.AGE FROM MEMBER M
```

(JPQL은 뒤에서 자세히 배운다)