# 섹션 2 JPA 시작하기

## 1. Hello JPA - 프로젝트 생성

### 1.1 환경 구성

- H2 설치하기: http://www.h2database.com/
- 메이븐 사용: https://maven.apache.org/
- 자바 8 버전: 자바 8 이상(8 권장)

- groupId: jpa-basic
- artifactId: ex1-hello-jpa
- version: 1.0.0

라이브러리 추가 - 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>

        <dependency>
            <groupId>javax.xml.bind</groupId>
            <artifactId>jaxb-api</artifactId>
            <version>2.3.0</version>
        </dependency>

    </dependencies>
</project>
```

JPA 설정하기 - **persistence.xml**

- JPA 설정 파일
- /META-INF/persistence.xml 위치
- persistence-unit name으로 이름 지정
- javax.persistence로 시작: JPA 표준 속성
- hibernate로 시작: 하이버네이트 전용 속성

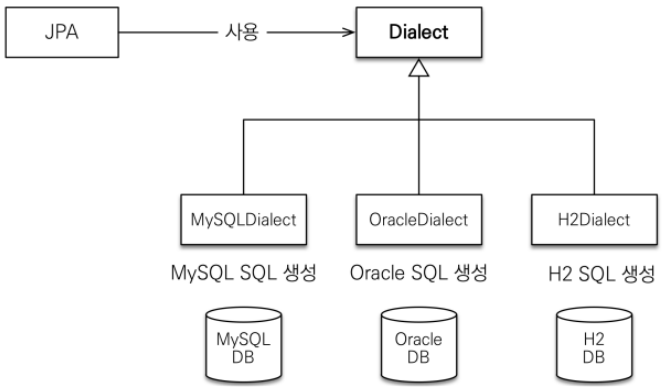
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.2"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="hello">
        <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.jdbc.batch_size" value="10"/>
            <property name="hibernate.hbm2ddl.auto" value="create" />

        </properties>
    </persistence-unit>
</persistence>
```

javax: 다른 JPA 구현체를 사용하더라도 그대로 사용이 가능하다. (표준)
hibernate: 하이버네이트에 종속적이다. (hibernate 전용 옵션)

## 1.2 데이터베이스 방언

- JPA는 특정 데이터베이스에 종속되지 않는다.
- 각각의 데이터베이스가 제공하는 SQL 문법과 함수는 조금씩 다르다.
    - 가변 문자: MySQL은 VARCHAR, Oracle은 VARCHAR2
    - 문자열을 자르는 함수: SQL 표준은 SUBSTRING(), Oracle은 SUBSTR()
    - 페이징: MySQL은 LIMIT , Oracle은 ROWNUM

방언이란, SQL 표준을 지키지 않는 특정 데이터베이스만의 고유한 기능을 의미한다. 대표적으로 MySQL은 MySQLDialect, Oracle은 OracleDialect, H2는 H2Dialect 가 있다.

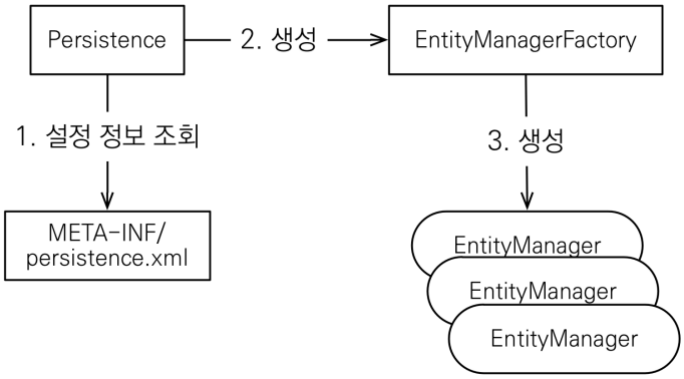



persistence.xml 파일의 hibernate.dialect 속성에서 데이터베이스 방언을 지정할 수 있다.

- H2 : org.hibernate.dialect.H2Dialect
- Oracle 10g : org.hibernate.dialect.Oracle10gDialect
- MySQL : org.hibernate.dialect.MySQL5InnoDBDialect
- 하이버네이트는 40가지 이상의 데이터베이스 방언 지원

# 2.Hello JPA - 애플리케이션 개발

## 2.1 JPA 구동 방식




> 1. persistence 클래스에서 시작
> 2. xml 설정 정보를 읽음
> 3. 읽은 정보를 기반으로 entityManagerFactory 클래스를 생성
> 4. entityManagerFactory 클래스가 필요할때마다 EntityManager를 만들어서 작동

```
// java -> hellojpa -> JpaMain클래스 생성
public class JpaMain {
    public static void main(String[] args) {
        // Persistence 클래스가 persistence.xml(name을 hello로 설정)을 읽고,
EntityManagerFactory 생성
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("hello");

        // EntityManagerFactory가 EntityManager를 생성
        EntityManager em = emf.createEntityManager();
        //code작성위치, EntityManager를 통해 쿼리를 날리는 등 jpa의 작업 진행
        em.close();
        emf.close();
    }
}
```

## 2.2 EntityManagerFactory

- `EntityManagerFactory`는 Java Persistence API (JPA)에서 사용되는 중요한 인터페이스
- `EntityManagerFactory`는 JPA의 핵심 요소 중 하나로서, 애플리케이션에서 `EntityManager`를 생성하는 역할
- `EntityManager`는 JPA에서 영속성 컨텍스트와 관련된 작업을 수행하며, 데이터베이스와의 상호 작용을 관리
- JPA 는 `EntityManagerFactory` 를 생성한 후에 사용
- application loading 시점에 DB 당 딱 하나만 생성되어야 함

간단히 말하면, `EntityManagerFactory`는 JPA를 사용하여 데이터베이스에 액세스하는 데 필요한 `EntityManager` 인스턴스를 생성합니다. `EntityManager`는 데이터베이스 트랜잭션을 관리하고 엔터티의 영속성을 제어합니다.

```
EntityManagerFactory entityManagerFactory =
            Persistence.createEntityManagerFactory("hello");
```

`persistence.xml` 의 `persistence-unit` 태그와 동일한 이름으로 인자값을 설정해야 한다.
그래야 설정 파일의 정보들을 읽어와 해당 객체를 만들 수 있다.

```
entityManagerFactory.close();
```

WAS가 종료되는 시점에 `EntityManagerFactory` 를 닫는다. 그래야 내부적으로 Connection pooling 에 대한 Resource 가 Release 된다.

## 2.3 EntityManager

- `EntityManager`는 엔터티의 영속성을 관리하고 데이터베이스와의 상호 작용을 제어하는 인터페이스
- 실제 Transaction 단위를 수행할 때마다 생성한다.

- 즉, 고객의 요청이 올 때마다 사용했다가 닫는다. thread 간에 공유하면 안된다. (사용하고 버려야 한다.)

`EntityManager` 인스턴스는 스레드 간에 공유되지 않아야 합니다. 일반적으로 각각의 트랜잭션마다 새로운 `EntityManager` 인스턴스를 생성하고 사용하는 것이 권장됩니다. 따라서 `EntityManager` 인스턴스는 하나의 트랜잭션 범위에 속하며, 트랜잭션이 끝나면 해당 `EntityManager`를 닫아야 합니다.

```
entityManager.close();
```

Transaction 수행 후에는 반드시 EntityManager 를 닫는다. 그래야 내부적으로 DB Connection 을 반환한다.

## 2.4 EntityTransaction

- `EntityTransaction`은 JPA(Java Persistence API)에서 트랜잭션을 관리하는 인터페이스
- JPA에서 데이터베이스와의 상호 작용은 트랜잭션 내에서 이루어짐
- `EntityTransaction`은 이러한 트랜잭션을 시작, 커밋, 롤백하는 등의 작업을 수행

`EntityTransaction`이 제공하는 주요 메서드는 다음과 같습니다:

1. `begin()` **메서드**
    - 새로운 트랜잭션을 시작
    - 트랜잭션 시작 이후에는 엔터티 매니저를 통해 엔터티의 상태를 변경하고 데이터베이스와 상호 작용이 가능

    ```
    EntityTransaction transaction = entityManager.getTransaction();
    transaction.begin();
    ```

2. `commit()` **메서드**
    - 현재 진행 중인 트랜잭션을 커밋
    - 트랜잭션 내의 모든 변경 사항이 데이터베이스에 반영

    ```
    transaction.commit();
    ```

3. `rollback()` **메서드**
    - 현재 진행 중인 트랜잭션을 롤백
    - 트랜잭션 내의 모든 변경 사항이 취소되고, 데이터베이스는 트랜잭션 이전의 상태로 복구

    ```
    transaction.rollback();
    ```

4. `setRollbackOnly()` **메서드**
    - 현재 트랜잭션을 롤백 상태로 표시
    - 이 메서드를 호출하면 트랜잭션이 커밋되지 않고 롤백

    ```
    transaction.setRollbackOnly();
    ```

`EntityTransaction`을 사용하여 트랜잭션을 관리하면 데이터베이스 작업을 안전하게 수행할 수 있습니다. 예외가 발생하면 롤백을 수행하여 데이터의 일관성을 유지하고, 모든 작업이 정상적으로 완료된 경우에만 커밋을 수행하여 데이터베이스에 변경 사항을 반영합니다.

아래는 `EntityTransaction`을 사용한 간단한 예제 코드입니다:

```
import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

public class TransactionExample {
    public static void main(String[] args) {
        // EntityManager 생성
        EntityManager entityManager = Persistence.
                createEntityManagerFactory("persistence-unit-
name").createEntityManager();

        // EntityTransaction 얻기
        EntityTransaction transaction = entityManager.getTransaction();

        try {
            // 트랜잭션 시작
            transaction.begin();

            // 엔터티 작업 수행
            // ...

            // 트랜잭션 커밋
            transaction.commit();
        } catch (Exception e) {
            // 예외 발생 시 롤백
            if (transaction.isActive()) {
                transaction.rollback();
            }
            e.printStackTrace();
        } finally {
            // EntityManager 닫기
            entityManager.close();
        }
    }
}
```

이 코드에서 `transaction.begin()`으로 트랜잭션을 시작하고, `transaction.commit()`으로 트랜잭션을 커밋합니다.
예외가 발생하면 `transaction.rollback()`으로 트랜잭션을 롤백합니다.

- Data 를 "변경"하는 모든 작업은 반드시 Transaction 안에서 이루어져야 한다.

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

단순한 조회의 경우는 상관없음.

**핵심 포인트**

1. 엔티티 매니저 팩토리는 하나만 생성해서 애플리케이션 전체에서 공유한다.
2. 엔티티 매니저는 쓰레드 간에 공유 X (사용하고 버려야 한다.), 데이터 베이스 커넥션을 빨리 쓰고 돌려줘야 하기 때문이다.
3. JPA의 모든 데이터 변경은 트랜잭션 안에서 실행한다.

## 2.5 JPQL 소개

- 가장 단순한 조회 방법
    1. `EntityManager.find()`
    2. 객체 그래프 탐색(`a.getB().getC()`)

단순한 조회 방법이 아닌 조건을 부여해서 **나이가 18살 이상인 회원을 모두 검색하고 싶다면?** 또는 **모든 회원을 검색하고 싶다면?**

이럴때 `JPQL`을 써야 한다. (현업에서 개발의 고민은, 테이블이 굉장히 많고 필요하면 `JOIN`도 해야하고, 내가 원하는 데이터를 최적화 해서 가져와야 하고, 필요하면 통계성 쿼리도 날려야 하는데,, 이걸 어떻게 할거냐? -> `JPA`에서 `JPQL`를 이용하여 도와준다.)

**JPQL로 전체회원 조회**

```
List<Member> result = em.createQuery("select m from Member as m",
Member.class).getResultList();
```

JPA입장에서, 테이블을 대상으로 코드를 절대 짜지 않는다.
**즉 위 코드의 createQuery 안의 쿼리에서 Member는 `멤버객체` 를 의미한다. (테이블이 아니라는 것이 `포인트`다.)**

```
List<Member> result = em.createQuery("select m from Member as m", Member.class)
                    .getResultList();

            for (Member member : result) {
                System.out.println("member.name = " + member.getName());
            }
```

```
Hibernate:
    /* select
        m
    from
        Member as m */ select
            member0_.id as id1_0_,
            member0_.name as name2_0_
        from
            Member member0_
member.name = HelloJPA
member.name = HelloB
```

실제로 쿼리는 id, name 필드를 select 하지만, JPQL에서는 m, 즉 **멤버 엔티티**를 선택한 것이라고 보면 된다.
이 `JPQL`이 어떤 메리트가 있을까?

`페이징`할때 엄청난 메리트가 있다.

```
List<Member> result = em.createQuery("select m from Member as m", Member.class)
                    .setFirstResult(5)
                    .setMaxResults(8)
                    .getResultList();
```

```
Hibernate:
    /* select
        m
    from
        Member as m */ select
            member0_.id as id1_0_,
            member0_.name as name2_0_
        from
            Member member0_ limit ? offset ?
```

이렇게 페이지 가져올때 굉장히 편하다. 즉, `JPQL`은 객체를 대상으로 하는 `객체지향 쿼리`라고 보면 된다.
위와 같이 JPQL을 짜면 각 DB의 dialect에 맞게 번역해 준다.

---

## JPQL 정리

- `JPA`를 사용하면 `엔티티 객체`를 중심으로 개발
- 문제는 검색 쿼리에서 발생한다. (데이터를 단건만 가져오는게 아니라 여러개를 가져와야 하는 상황 -> 데이터베이스에서 데이터를 필터링 해서 가져와야 하는 상황)
- 검색을 할 때도 테이블이 아닌 `엔티티 객체`를 대상으로 검색 (테이블에서 검색하면 `JPA`의 사상이 깨짐)
- 그렇다고 모든 `DB` 데이터를 객체로 변환해서 검색하는 것은 불가능함
- 애플리케이션이 필요한 데이터만 DB에서 불러오려면 결국 **검색 조건이 포함된 SQL**이 필요하다.
- `JPA`는 `SQL`을 추상화한 `JPQL`이라는 `객체 지향 쿼리 언어` 제공
- `SQL`과 문법 유사, `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING`, `JOIN` 지원
- `JPQL`은 엔티티 객체를 대상으로 쿼리
- `SQL`은 데이터베이스 테이블을 대상으로 쿼리
- 테이블이 아닌 객체를 대상으로 검색하는 객체 지향 쿼리
- `SQL`을 추상화해서 특정 데이터베이스 `SQL`에 의존X
- `JPQL`을 한마디로 정의하면 `객체 지향 SQL`

결국 테이블을 대상으로 쿼리를 짜면 **RDB에 종속적인 설계**가 되버린다. 그래서 엔티티 객체를 대상으로 쿼리를 할 수 있는 `JPQL`이 제공 되는것이다.