ORM 개념과 JPA 소개

ORM(Object-Relational Mapping)의 개념과 장단점

- ORM이란?

  : **Object-Relational Mapping**으로 객체와 관계형 데이터 베이스의 데이터를 자동으로 매핑해주는 것

- 객체 지향 프로그래밍과 관계형 데이터베이스

  - 객체 지향 프로그래밍에서는 클래스를 사용해 데이터를 표현

  - 관계형 데이터베이스는 데이터를 테이블 형태로 저장

이렇게 되면..!

객체 모델과 관계형 모델 사이에서 불일치가 존재하게 된다.

그래서 ORM을 통해 **객체 간의 관계를 바탕으로 SQL을 자동으로 생성**하여 불일치를 해결해준다.

- ORM의 장점
  - 객체 지향적인 코드로 더 직관적이고 비즈니스 로직에 더 집중할 수 있게 도와준다.

    ⇒ SQL 쿼리가 아닌 직관적인 코드로 데이터를 조작할 수 있다.
  - 재사용성 및 유지보수의 편리성이 증가합니다.

    ⇒ ORM은 독립적으로 작성되어있고, 해당 객체들을 재활용할 수 있습니다.
- ORM의 단점
  - 완벽한 ORM으로만 서비스를 구현하기가 어렵습니다.

    ⇒ 프로젝트의 복잡성이 커질 경우 난이도 또한 올라갈 수 있습니다.
  - 프로시저가 많은 시스템에서는 ORM의 객체 지향적인 장점을 활용하기 어렵습니다.

JPA의 등장 배경과 특징

- 등장 배경
  - 객체와 관계형 데이터베이스의 차이
  - 쿼리문 무한 반복과 지루한 코딩
  - 객체의 필드가 추가되면 모든 쿼리문 수정 필요
- 특징
  - Java Persistemce API로 자바 ORM 기술에 대한 API 표준이다.
  - 특정 데이터베이스에 종속되지 않습니다. 예를 들어 오라클에서 MariaDB로 변경한다면 쿼리문이 달라 전체 수정을 해야 한다. 하지만, JPA에서는 설정 파일에 어떤 데이터베이스를 사용하는지 알려주면 데이터베이스를 변경할 수 있다.
  - 새로운 컬럼이 추가되었을 때, JPA에서는 테이블과 매핑된 클래스에 필드만 추가하면 쉽게 관리가 가능하다.
  - 직접 SQL문을 작성하지 않고 객체를 사용하여 동작하기 때문에 유지보수 측면에서 좋고 재사용성도 증가한다.

아래는 JPA와 SQL문으로 id 값을 조회한 쿼리입니다.

```
// JPA 사용 예시
Optional<Entity> entity = entityRepository.findById(1);

// SQL을 직접 작성했을 때
SELECT * FROM entity_table WHERE id = 1;
```

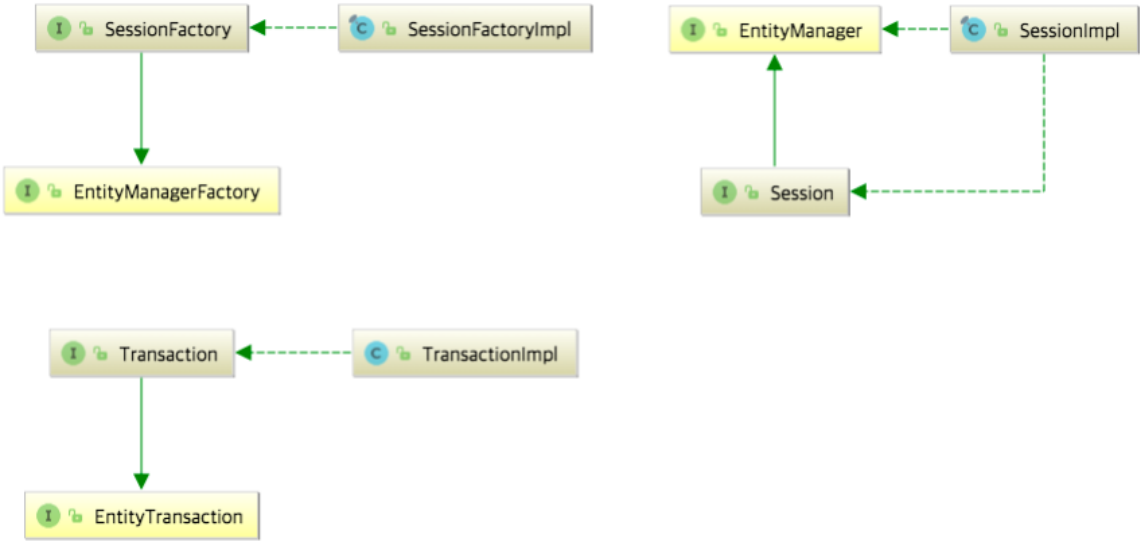
JPA vs Hibernate vs Spring Data JPA

- **JPA**
  - 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스이다.
  - 특정 기능을 하는 라이브러리가 아닌 인터페이스이다.
  - 관계형 데이터베이스를 어떻게 사용해야 하는 지를 정의한 방법이다.

- **Hibernate**
  - JPA라는 명세의 구현체
  - `javax.persistence.EntityManager` 와 같은 인터페이스를 직접 구현한 라이브러리
  - Java의 인터페이스와 해당 인터페이스를 구현한 클래스와 같은 관계이다.
  - 아래 사진은 JPA와 Hibernate의 상속 및 구현 관계를 나타낸 것이다.

    JPA의 핵심인 `EntityManagerFactory`, `EntityManager`, `EntityTransaction` 을

    Hibernate에서는 `SessionFactory`, `Session`, `Transaction` 으로 상속받고 각각 Impl로 구현



** 반드시 이걸 사용할 필요 없고, DataNucleus, EclipseLink 등 다른 JPA 구현체를 사용해도 된다.

- **Spring Data JPA**
  - Spring에서 제공하는 모듈 중 하나
  - JPA를 한 단계 추상화 시킨 `Repository` 라는 인터페이스를 제공
  - `Repository` 인터페이스에 정해진 규칙대로 메소드 입력하면 Spring이 알아서 해당 메소드 이름에 적합한 쿼리를 날리는 구현체를 만들어 Bean으로 등록
  - 아래는 `Repository` 인터페이스의 기본 구현체인 `SimpleJpaRepository` 의 코드이다. `EntityManager` 을 사용하고 있다.

```
public class SimpleJpaRepository<T, ID> implements JpaRe
positoryImplementation<T, ID> {

    private final EntityManager em;

    public Optional<T> findById(ID id) {

        Assert.notNull(id, ID_MUST_NOT_BE_NULL);

        Class<T> domainType = getDomainClass();

        if (metadata == null) {
            return Optional.ofNullable(em.find(domainTyp
e, id));
        }

        LockModeType type = metadata.getLockModeType();

        Map<String, Object> hints = getQueryHints().with
FetchGraphs(em).asMap();

        return Optional.ofNullable(type == null ? em.fin
d(domainType, id, hints) : em.find(domainType, id, type,
hints));
    }
```

JPA 설정 방법 (persistence.xml, application.properties/yml)

- JPA를 사용하기 위한 설정 방법
  - persistence.xml

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persist
    <persistence-unit name="myPersistenceUnit">
        <properties>
```

```
            <!-- JPA 설정 -->
            <property name="hibernate.dialect" value="o
            <property name="hibernate.show_sql" value="
            <property name="hibernate.format_sql" value
            <property name="hibernate.hbm2ddl.auto" val
        </properties>
    </persistence-unit>
</persistence>
```

- application.properties

```
spring.jpa.database-platform=org.hibernate.dialect.MySQ
spring.jpa.show-sql=true                # 실행되는 SQL 쿼리
spring.jpa.hibernate.ddl-auto=update    # 테이블 생성 및 업
```

- application.xml

```
spring:
    jpa:
        database-platform: org.hibernate.dialect.MySQLD
        show-sql: true                  # 실행되는 SQL 출력
        hibernate:
        ddl-auto: update              # 테이블 생성 및 업데이트
```