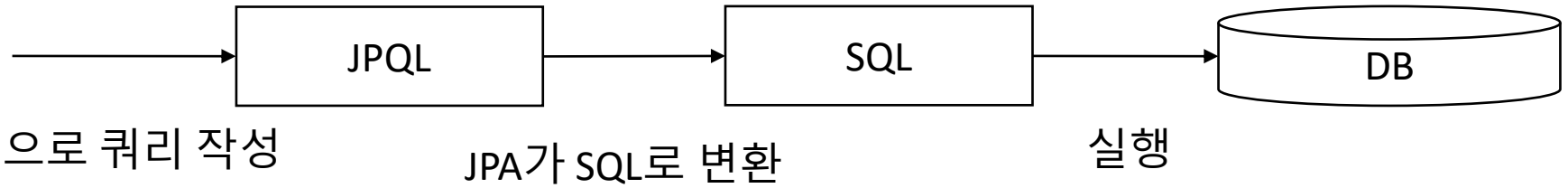
JPQL

JPQL – How to Define Queries in JPA and Hibernate

https://thorben-janssen.com/jpql/

소개

- **JPQL(Java Persistence Query Language)**
  - 객체지향 쿼리
  - JPQL은 테이블을 대상으로 쿼리를 생성하는 것이 아닌 엔티티 객체를 대상으로 쿼리를 생성
  - JPQL로 실행한 쿼리도 결국 SQL로 변환되어 실행
  - 따라서 실행되는 SQL 로깅하여 확인하는 작업이 필요
  - JPA에서 관리되는 엔티티에 대한 SQL은 트랜잭션 커밋 시점이지만 JPQL은 즉시 실행 됨



Selection

- **Selection – The FROM clause**
  - FROM 절은 select 대상이 되는 엔티티를 지정하기 위해 사용

```
SELECT a FROM Author (AS) a
```

  - AS는 생략 가능
  - 위 예시에서 Author는 DB 테이블을 의미하는 것이 아닌 엔티티를 의미
  - 반드시 별칭을 명시해야 함. 위 예시에서는 a에 해당
  - persist가 INSERT문을 대신함

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Types of JPA Queries

- **Query의 종류**
  - TypedQuery
    - 쿼리 실행의 결과로 반환될 타입을 알고 있을 때
    - 추가적인 형변환이 없고 유연하고 쉬운 테스트를 제공

```
public UserEntity getUserByIdWithTypedQuery(Long id) {
   TypedQuery<UserEntity> typedQuery
     = getEntityManager().createQuery("SELECT u FROM UserEntity u WHERE u.id=:id", UserEntity.class);
   typedQuery.setParameter("id", id);
   return typedQuery.getSingleResult();
}
```

  - NamedQuery: 엔티티에 실행될 JPQL과 그 이름을 명시하여 직관적으로 이해

```
@Table(name = "users")
@Entity
@NamedQuery(name = "UserEntity.findByUserId", query = "SELECT u FROM UserEntity u WHERE u.id=:userId")
public class UserEntity {
...
}
```

https://www.baeldung.com/jpa-queries

Types of JPA Queries

- **Query의 종류**
  - NativeQuery
    - SQL쿼리
    - JPQL의 구문에 제약 없이 대상 DB에 곧 바로 SQL실행
    - JPQL이나 JPA의 관리를 벗어나므로 이식성이 떨어질 수 있음
    - NativeQuery는 최후의 수단으로 생각

```
public UserEntity getUserByIdWithNativeQuery(Long id) {
    Query nativeQuery
      = getEntityManager().createNativeQuery("SELECT * FROM users WHERE id=:userId", UserEntity.class);
    nativeQuery.setParameter("userId", id);
    return (UserEntity) nativeQuery.getSingleResult();
}
```

https://www.baeldung.com/jpa-queries

TypedQuery, Query

- 작성한 JPQL을 실행하려면 쿼리 객체를 만들어야 함
  - 타입을 명시할 수 있을 때

SELECT NAME FROM PERSON

```
TypedQuery<String> query = em.createQuery("SELECT p.name from Person p", String.class);
List<String> resultList = query.getResultList();
```

  - 타입을 명시할 수 없을 때

```
Query query = em.createQuery("SELECT p.name, p.age from Person p");
List resultList = query.getResultList();
for(Object o: resultList){
        Object[] result = (Object[]) o; //결과가 둘 이상이면 Object[] 반환, 하나면 Object 반환
        System.out.println("name = " + result[0]);
        System.out.println("age = " + result[1]);
}
```

getResult

- query.getResultList()
  - 결과가 하나 이상인 경우, 리스트를 반환
  - 결과가 없으면 null반환
- query.getSingleResult()
  - 일치하는 하나의 엔티티를 반환
  - 결과가 없거나 한 개를 초과하면 예외 발생
    - 스프링 데이터 JPA에서는 한 개를 초과할 때 null 반환

SQL이 실행되는 시점 확인해보기

Polymorphism and Restriction

- **Polymorphism**
  - book과 blogpost를 포함하는 publication 조회 가능(book도 개별적으로 조회 가능)

```
SELECT p FROM Publication p

SELECT b FROM BlogPost b
```

- **Restriction**
  - where에서 조건 걸기(검색 대상 제한)

```
SELECT a FROM Author a WHERE a.firstName like '%and%' and a.id >= 20 and size(author.books) >= 5
```

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Projection – The SELECT clause

- **Projection**
  - 테이블의 모든 컬럼이 아닌 특정 값(컬럼) 만 지정해서 조회
  - Projection은 다양한 대상에 적용 가능
  - Entities

    SELECT a FROM Author a
  - Scalar values

    SELECT a.firstName, a.lastName FROM Author a
  - Constructor references(DTO에 매핑)

    SELECT new org.thoughts.on.java.model.AuthorValue(a.id, a.firstName, a.lastName) FROM Author a

- **Distinct query results**

  SELECT DISTINCT a.lastName FROM Author a

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Projection – The SELECT clause

- **Function**
  - upper(String s)
  - lower(String s)
  - current_date()
  - current_time()
  - current_timestamp()
  - …
- **Ordering – The ORDER BY clause**
  - 정렬 방식 지정: ascending (ASC) or a descending (DESC) order

```
SELECT a FROM Author a ORDER BY a.lastName ASC, a.firstName DESC
```

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Joins

- **Inner Joins**

```
SELECT a, b FROM Author a JOIN a.publications b
```

  - FROM 절은 select 대상이 되는 엔티티를 지정하기 위해 사용
  - Author 엔티티가 Hibernate에 publications 엔티티와 어떻게 연관관계를 맺고 있는지 알려주므로 추가적인 ON절이 필요 없음
  - 연관관계가 설정되지 않은 엔티티간 조인은 JPA에서 제공하지 않음. 대신 세타 조인와 WHERE절을 이용하여 조인을 수행

```
SELECT b, p FROM Book b, Publisher p WHERE b.fk_publisher = p.id
```

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Joins

- **Left Outer Joins**
  - 연관관계 여부와 상관 없이 대상 엔티티는 모두 조회할 때 사용
    - 이전 예제의 INNER조인은 publication 이력이 있는 Author만 조회된다면 Left Outer 조인은 publication 여부와 상관없이 모두 조회

```
SELECT a, b FROM Author a LEFT JOIN a.publications b
```

- **Additional Join Conditions**
  - JPA 2.1부터 INNER 조인의 ON절 지원

```
SELECT a, p FROM Author a JOIN a.publications p ON p.publishingDate > ?1
```

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/

Join Fetch

- **Join Fetch**
  - 대상 엔티티를 조회할 때 연관 엔티티도 함께 조회
  - FetchType을 Lazy로 두는 것을 권장하지만 성능 문제가 발생할 수 있음
  - 이때, Join Fetch사용
  - 아래는 타입을 알고 있으므로 Query생성과 실행을 체이닝 형태로 수행

```
List<Team> result = em.createQuery("SELECT p FROM Person
p join fetch p.addresses", Person.class)
.getResultList();
```

실행되는 SQL 확인해보기

https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/