JPA쿼리(네이티브 쿼리, Native Query)

- JPA에서 엔티티를 기반으로 JPQL을 직접 작성하거나 또는 SQL 쿼리를 타입 세이프한 방식인 메소드기반 형태로 쿼리를 만들면 JPQL(Java Persistence Query Language)로 변환되고 이것이 JPA 구현체 하이버네이트 같은 것에 의해 SQL문으로 변환되어 DB에서 실행된다.

- JPA는 JAVA 영속성 관리를 위한 SQL 작성 표준 API로 JPA에서 지원하는 않고 해당 DB에서만 사용 가능한 쿼리라면 Native SQL을 사용해서 DB에서 사용하는 SQL구문 형식 그대로 쿼리를 작성하면 된다.

- Native SQL을 사용하는 방법은 간단하다. EntityManager의 **createNativeQuery**() 메소드를 이용하거나 Spring Data JPA를 사용한다면 @Query에 nativeQuery=true 라고 하면 DB에서 사용하는 SQL 구문을 직접 사용할 수 있다.

```
// Emp 테이블에서 모든 사원의 이름, 급여 출력
Query query = em.createNativeQuery("SELECT e.ename, e.sal FROM Emp e");
List<Object[]> emps = query.getResultList();  //여러건을 리턴받으므로 getResultList()를 사용한다.

for (Object[] e : emps) {
    System.out.println("Ename ="+ e[0] + ", Salary = " + e[1]);
}

////////////////////////////////////// 파라미터 바인딩
// ?를 이용한 파라미터 바인딩, Query 인터페이스의 setParameter() 메소드로 바인딩을 한다.
// Emp 테이블에서 job이 "CLERK"의 사원들의 이름, 급여 출력
Query query = em.createNativeQuery("SELECT e.ename, e.sal FROM Emp e WHERE e.job = ?");
Query.setParameter(1, "CLERK");
List<Object[]>emps = (Object[]) query.getResultList();

for (Object[] e : emps) {
    System.out.println("Ename ="+ e[0] + ", Salary = " + e[1]);
}

// ":"을 사용한 네임드 파라미터 바인딩, 첫글자는 ":"이고 이후는 임의로 부여 가능
// 기본JPA에서는 지원하지 않으며 하이버네이트에서는 Named Parameter Biding을 지원한다.
// Spring Data JPA에서도 가능한 방식
Query query = em.createNativeQuery("SELECT e.ename, e.sal FROM Emp e WHERE e.job = :job");
```

```
Query.setParameter(1, "CLERK");
List<Object[]>emps = query.getResultList();

for (Object[] e : emps) {
      System.out.println("Ename ="+ e[0] + ", Salary = " + e[1]);
}

////////////////////////////////// 쿼리결과 매핑
// 위 예문처럼 일부 칼럼을 추출하는 경우 Object[] 의 List 타입으로 받는 방식
List<Object[]> emps = em.createNativeQuery("SELECT e.ename, e.sal FROM Emp e").getResultList();

// createNativeQuery() 메소드의 인자로 결과타입을 넘기는 방식을 사용할 수 있다.
Query query = em.createNativeQuery("SELECT * FROM Emp WHERE job = ?", Emp.class);
query.setParameter(1, "SALESMAN");
List<Emp> emps = query.getResultList();

for (Emp e : emps) {
      System.out.println("Ename ="+ e.ename + ", Salary = " + e.sal);
}

// 결과 매핑 사용, 매핑이 복잡한 경우 @SqlResultSetMaping 어노테이션을 이용하여
// 별도의 엔티티에서 결과 매핑을 정의하는 것이 좋다.

Query query = em.createNativeQuery(
          "SELECT e.empno AS empno, " +
          "          e.ename AS ename, " +
          "          e.sal       AS sal,      " +
          "          sal * 12   AS annual_income " +
          "FROM Emp e " +
          "WHERE sal > 1000",
          "empResults");



@SqlResultSetMapping(name="empResults",
      entities={
            @EntityResult(entityClass=Emp.class,
              fields={
                    @FieldResult(name="no",     column="empno"),
                    @FieldResult(name="name", column="ename"),
```

```
                    @FieldResult(name="salary",    column="sal")})},
                columns={
            @ColumnResult(name="annual_income")}
@Entity
class Emp {
            private Long no;
            private String name;
            private Long salary;

            //getter, setter
}

// 위 쿼리의 결과는 아래처럼 읽을 수 있다.

for (Object[] row : query.getResultList()) {
    Emp emp = (Emp)row[0];                    //@EntityResult의 결과
    Long annual_income = (Long)row[1];     //@ColumnResult의 결과
}
```