004. Kotlin에서 JPA(Hibernate) ID 매핑 팁과 PostgreSQL ID 컬럼 자료형에 따른 매핑 방법

Hibernate, Kotlin without nullable ID Property

Hibernate Entity에서 ID는 객체 생성 시점에 값을 알 수 없기 때문에
Kotlin Nullable 타입을 지정해줘야 했다.

```
@Table(name = "GRP_GROUP")
@Entity(name = "Group")
class Group(
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ_GROUP")
    @SequenceGenerator(name = "SEQ_GROUP", sequenceName = "SEQ_GROUP", allocationSize = 1)
    @Column(name = "GROUP_KEY", unique = true, nullable = false, updatable = false)
    val key: Long? = null,

    @Column(name = "GROUP_CODE", columnDefinition = "uuid", updatable = false)
    val code: UUID
) {
    val rowId: String
        get() = "SG-${key}"
}
```

그래서 Entity 조회 후 key 프로퍼티는 항상 아래와 같이 써줘야 했다.
이 부분이 매우 마음에 안들었다.

```
group.key!!
```

(1) -1로 값을 주면 어떻게 되나?
이렇게 주면 어떻게 돌아갈까 테스트 해봤다.

```
//given
val group = Group(-1L, code = UUID.randomUUID())

//when
val saved = groupRepository.save(group)

//then
assertThat(saved.key).isNotNull
```

삽입은 잘 된다.
하지만 아래와 같이 select를 한 번 치고 sequence를 조회해서 insert 한다.

> select group0_.group_key as group_ke1_0_0_, group0_.group_code as group_co2_0_0_ from GRP_GROUP group0_ where group0_.group_key=?

> select nextval ('seq_group')

> insert into GRP_GROUP (group_code, group_key) values (?, ?)

ID가 있기에 새로운 엔티티라고 생각 안한 듯하다.
-1로 조회 된 엔티티가 없으니 새로 삽입 절차를 거친 듯 했다.

그럼 Hibernate에서 entity가 언제 새로운 Entity인지 판단하는지 궁금했다.
ID가 null이거나 null이 아니고 DB에 존재하지 않거나 하는 경우 외에 뭐가 있을 지...

SimpleJpaRepository의 save는 아래와 같이 생겼다.
entityInformation 객체를 통해 isNew인지 확인한다.

```
@Transactional
@Override
public <S extends T> S save(S entity) {

    Assert.notNull(entity, "Entity must not be null.");

    if (entityInformation.isNew(entity)) {
        em.persist(entity);
        return entity;
    } else {
        return em.merge(entity);
    }
}
```

entityInformation객체는 AbstractEntityInformation 클래스의 인스턴스이다.
isNew 메서드면 null인지 비교하고 값이 0인지 비교한다.
-1 대신 0을 새로운 엔티티로 판단한다.

```
public boolean isNew(T entity) {
    ID id = getId(entity);
    Class<ID> idType = getIdType();

    if (!idType.isPrimitive()) {
        return id == null;
    }

    if (id instanceof Number) {
        return ((Number) id).longValue() == 0L;
    }

    throw new IllegalArgumentException(String.format("Unsupported primitive id type %s!", idType));
}
```

Entity 설정을 아래와 같이 바꾼다.

```
@Table(name = "GRP_GROUP")
@Entity(name = "Group")
class Group(
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ_GROUP")
    @SequenceGenerator(name = "SEQ_GROUP", sequenceName = "SEQ_GROUP", allocationSize = 1)
    @Column(name = "GROUP_KEY", unique = true, nullable = false, updatable = false)
    val key: Long = 0L,

    @Column(name = "GROUP_CODE", columnDefinition = "uuid", updatable = false)
    val code: UUID
) {
    val rowId: String
        get() = "SG-${key}"
}
```

아래와 같이 새로운 엔티티로 인식되어 flush 될 때 nextval + insert 쿼리만 나간다.

> select nextval ('seq_group') insert

> into GRP_GROUP (group_code, group_key) values (?, ?)

JPA, Postgresql ID Column 매핑

Postgresql에서 ID 컬럼을 설정하는 방법이 여러개가 있다.

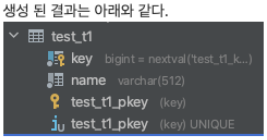
(1) Sequence를 생성하여 Default 속성으로

아래는 시퀀스 오브젝트를 생성한 후 테이블의 default 값으로 NEXTVAL을 호출하게 해 주는 것이다.

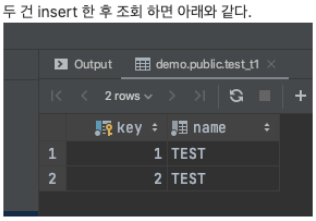
```
DROP TABLE IF EXISTS TEST_T1;               -- NEXTVAL로 시퀀스가 묶인 경우 테이블을 먼저 삭제 해 줘야 한다. 혹은 CASCADE 옵션으로.
DROP SEQUENCE IF EXISTS TEST_T1_KEY_SEQ;
CREATE SEQUENCE TEST_T1_KEY_SEQ;
CREATE TABLE TEST_T1
(
    KEY  BIGINT DEFAULT NEXTVAL('TEST_T1_KEY_SEQ'),
    NAME VARCHAR(512) NOT NULL,

    PRIMARY KEY (KEY)
);
INSERT INTO TEST_T1(NAME) VALUES ('TEST');
INSERT INTO TEST_T1(NAME) VALUES ('TEST');
```

생성 된 결과는 아래와 같다.



두 건 insert 한 후 조회 하면 아래와 같다.



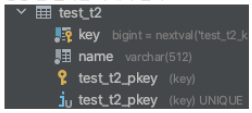
(2) SERIAL 자료형 사용

(1)의 방법은 시퀀스를 직접 생성해 줘야 한다. 또한 테이블 drop 시 시퀀스도 별도로 drop 해주어야 한다.
이러한 불편함을 해소하기 위해 SERIAL 자료형을 사용할 수 있다.
SERIAL은 INT(4) 자료형이므로 실무에선 BIGSERIAL INT(8) 자료형을 사용하는 것이 좋겠다.

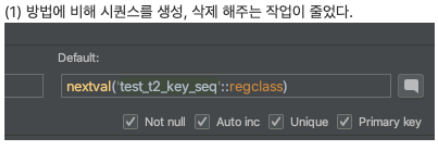
```
DROP TABLE IF EXISTS TEST_T2;
CREATE TABLE TEST_T2
(
    KEY  BIGSERIAL    NOT NULL,
    NAME VARCHAR(512) NOT NULL,

    PRIMARY KEY (KEY)
);
INSERT INTO TEST_T2(NAME) VALUES ('TEST');
INSERT INTO TEST_T2(NAME) VALUES ('TEST');
```

생성 된 결과는 아래와 같다.



SERIAL 자료형인 경우 기본적으로 'TABLE_NAME'_'KEY_NAME'_SEQ 이름으로 시퀀스를 만들고 default 값으로 지정 한다.
(1) 방법에 비해 시퀀스를 생성, 삭제 해주는 작업이 줄었다.



이렇게 SERIAL을 이용 하는 경우 JPA 매핑은 아래와 같다.
columnDefinination에 serial 혹은 bigserial을 기재하여 JPA(Hibernate)에게 알려준다.

```
@Table(name = "TEST_T2")
@Entity(name = "TestT2")
@Audited
@EntityListeners(value = [AuditingEntityListener::class])
class TestT2(
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "TEST_T2_KEY_SEQ")
    @SequenceGenerator(name = "TEST_T2_KEY_SEQ", sequenceName = "TEST_T2_KEY_SEQ", allocationSize = 1)
    @Column(name = "KEY", unique = true, nullable = false, updatable = false, columnDefinition = "bigserial")
    val key: Long = 0L,

    @Column(name = "NAME")
    val name: String,
)
```

(3) IDENTITY 속성 사용

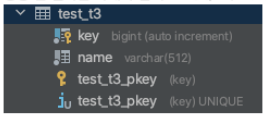
Serial 자료형은 표준이 아니다.
그에 따라 Postgresql 10 부터 Identity 속성을 지원한다.

> GENERATED (ALWAYS | BY DEFAULT) AS IDENTITY

```
DROP TABLE IF EXISTS TEST_T3;
CREATE TABLE TEST_T3
(
    KEY  BIGINT GENERATED BY DEFAULT AS IDENTITY,
    NAME VARCHAR(512) NOT NULL,

    PRIMARY KEY (KEY)
);
```

생성 된 결과는 아래와 같이 MYSQL의 Auto increment와 동일하다.



GENERATED BY DEFAULT는 IDENTITY 컬럼에 값을 넣거나 변경하는 것을 허용한다.

```
INSERT INTO TEST_T3(NAME) VALUES ('TEST');
INSERT INTO TEST_T3(NAME) VALUES ('TEST');
INSERT INTO TEST_T3(KEY, NAME) VALUES (3, 'TEST'); -- OK
INSERT INTO TEST_T3(KEY, NAME) VALUES (4, 'TEST'); -- OK
INSERT INTO TEST_T3(NAME) VALUES ('TEST');         -- ERROR (SEQ 조절 필요)

UPDATE TEST_T3 SET KEY = 6 WHERE KEY = 2; -- OK
```

반면 ALWAYS는 IDENTITY 컬럼에 값을 넣거나 변경하는 것을 불허한다.

```
CREATE TABLE TEST_T4
(
    KEY  BIGINT GENERATED ALWAYS AS IDENTITY,
    NAME VARCHAR(512) NOT NULL,

    PRIMARY KEY (KEY)
);
INSERT INTO TEST_T4(NAME) VALUES ('TEST');
INSERT INTO TEST_T4(NAME) VALUES ('TEST');
INSERT INTO TEST_T4(KEY, NAME) VALUES (3, 'TEST');
```

```
[428C9] ERROR: cannot insert into column "key"
Detail: Column "key" is an identity column defined as GENERATED ALWAYS.
Hint: Use OVERRIDING SYSTEM VALUE to override.
```

```
UPDATE TEST_T4 SET KEY = 5 WHERE KEY = 2;
```

```
[428C9] ERROR: column "key" can only be updated to DEFAULT
Detail: Column "key" is an identity column defined as GENERATED ALWAYS.
```

Identity 설정인 경우 JPA 매핑은 아래와 같다.

Identity 설정인 경우 JPA 매핑은 아래와 같다.

```
@Table(name = "TEST_T4")
@Entity(name = "TestT4")
@Audited
@EntityListeners(value = [AuditingEntityListener::class])
class TestT4(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "KEY", unique = true, nullable = false, updatable = false)
    val key: Long = 0L,

    @Column(name = "NAME")
    val name: String,
)
```

Postgresql wiki에서 새로운 Application 이라면 Serial을 사용하지 말라고 권장하니
새로운 서비스라면 Identity 속성을 사용하는 것이 좋겠다.

참조

- https://www.postgresqltutorial.com/postgresql-identity-column/
- https://wiki.postgresql.org/wiki/Don%27t_Do_This#Don.27t_use_serial
- https://postgresql.kr/blog/sequence_pg.html

기타 POSTGRESQL 주의사항

https://wiki.postgresql.org/wiki/Don%27t_Do_This

원문

https://gitlab.crsdev.io/integration/sitn-api/-/blob/master/README.md