Primary 키 매핑

- @Id : 개발자가 직접 지정
- @GeneratedValue
  - IDENTITY: 데이터베이스에 위임, MYSQL
  - SEQUENCE: 데이터베이스 시퀀스 오브젝트 사용, ORACLE
    - @SequenceGenerator 필요
  - TABLE: 키 생성용 테이블 사용, 모든 DB에서 사용
    - @TableGenerator 필요
  - AUTO: 방언에 따라 자동 지정, 기본값

@Id

```
@Entity
public class MiniUser {

    @Id
    private Long id;

    ...
}
```

```
MiniUser user = new MiniUser();
user.setId(1L); // 직접 입력해야 함.
em.persist(user);
```

자동 생성 전략1 - @GeneratedValue(strategy = GenerationType.IDENTITY)

```
@Entity
public class MiniUser {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    ...
}
```

- 기본 키 생성을 데이터베이스에 위임
- 주로 MySQL, PostgreSQL, SQL Server, DB2에서 사용 (예: MySQL의 AUTO_ INCREMENT)
- 영속성 컨텍스트는 트랜잭션 commit 시점에 최종 쿼리를 날리지만 `IDENTITY`전략을 쓰는 엔티티는 persist() 호출 시 바로 INSERT 쿼리를 실행함.

  [이유] AUTO_ INCREMENT는 데이터베이스에 INSERT SQL을 실행 한 후에야 ID 값을 알 수 있음.

```
em.persist(user1);
em.persist(user2);
em.persist(user3);
// AUTO_INCREMENT 가 중간 중간 수행되지 않으면 user1, 2, 3 모두 같은 번호를 가지게 될 것임...
// 따라서 이러한 문제점 때문에 AUTO_INCREMENT를 제때 실행하기 위해서 persist()만 실행해도 쿼리가 바로 flushing 됨.
```

자동 생성 전략2 - @GeneratedValue(strategy = GenerationType.SEQUENCE)

- 데이터베이스 시퀀스는 유일한 값을 순서대로 생성하는 특별한 데이터베이스 오브젝트(예: 오라클 시퀀스)
- 오라클, PostgreSQL, DB2, H2 데이터베이스에서 사용

예시

```
@Entity
@SequenceGenerator(
        name = "MEMBER_SEQ_GENERATOR",
        sequenceName = "MEMBER_SEQ", //매핑할 데이터베이스 시퀀스 이름
        initialValue = 1, allocationSize = 1)
public class Member {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE,
            generator = "MEMBER_SEQ_GENERATOR")
    private Long id;
```

@SequenceGenerator

속성	설명	기본값
name	식별자 생성기 이름	필수
sequenceName	데이터베이스에 등록되어 있는 시퀀스 이름	`"hibernate_sequence"`
initialValue	DDL 생성 시에만 사용됨, 시퀀스 DDL을 생성할 때 처음 1 시작하는 수를 지정한다.	1
allocationSize	시퀀스 한 번 호출에 증가하는 수 (성능 최적화에 사용됨) 데이터베이스 시퀀스 값이 하나씩 증가하도록 설정되어 있으면 이 값 을 반드시 **1**로 설정해야 한다	**50**
catalog, schema	데이터베이스 catalog, schema 이름	

자동 생성 전략3 - @GeneratedValue(strategy = GenerationType.TABLE)

- 키 생성 전용 테이블을 하나 만들어서 데이터베이스 시퀀스를 흉내내는 전략
- 장점: 모든 데이터베이스에 적용 가능
- 단점: 성능

예시

```
create table MY_SEQUENCES (
    sequence_name varchar(255) not null,
    next_val bigint,
    primary key ( sequence_name )
)
```

```
@Entity
@TableGenerator(
        name = "MEMBER_SEQ_GENERATOR",
        table = "MY_SEQUENCES",
        pkColumnValue = “MEMBER_SEQ", allocationSize = 1)
public class Member {

    @Id
    @GeneratedValue(strategy = GenerationType.TABLE,
                    generator = "MEMBER_SEQ_GENERATOR")
    private Long id;
```

속성	설명	기본값
name	식별자 생성기 이름	필수
table	키생성 테이블명	`hibernate_sequences`
pkColumnName	시퀀스 컬럼명	`sequence_name`
valueColumnName	시퀀스 값 컬럼명	`next_val`
pkColumnValue	키로 사용할 값 이름	엔티티 이름
initialValue	초기 값, 마지막으로 생성된 값이 기준이다.	0
allocationSize	시퀀스 한 번 호출에 증가하는 수(성능 최적화에 사용됨)	50
catalog, schema	데이터베이스 catalog, schema 이름	
uniqueConstraints(DDL)	유니크 제약 조건	

권장하는 식별자 전략

- 기본 키 제약 조건: null 아님, 유일, 변하면 안된다.
- 미래까지 이 조건을 만족하는 자연키는 찾기 어렵다. 대리키(대 체키)를 사용하자.
- 예를 들어 주민등록번호도 기본 키로 적절하기 않다.
- 권장: Long형 + 대체키 + 키 생성전략 사용