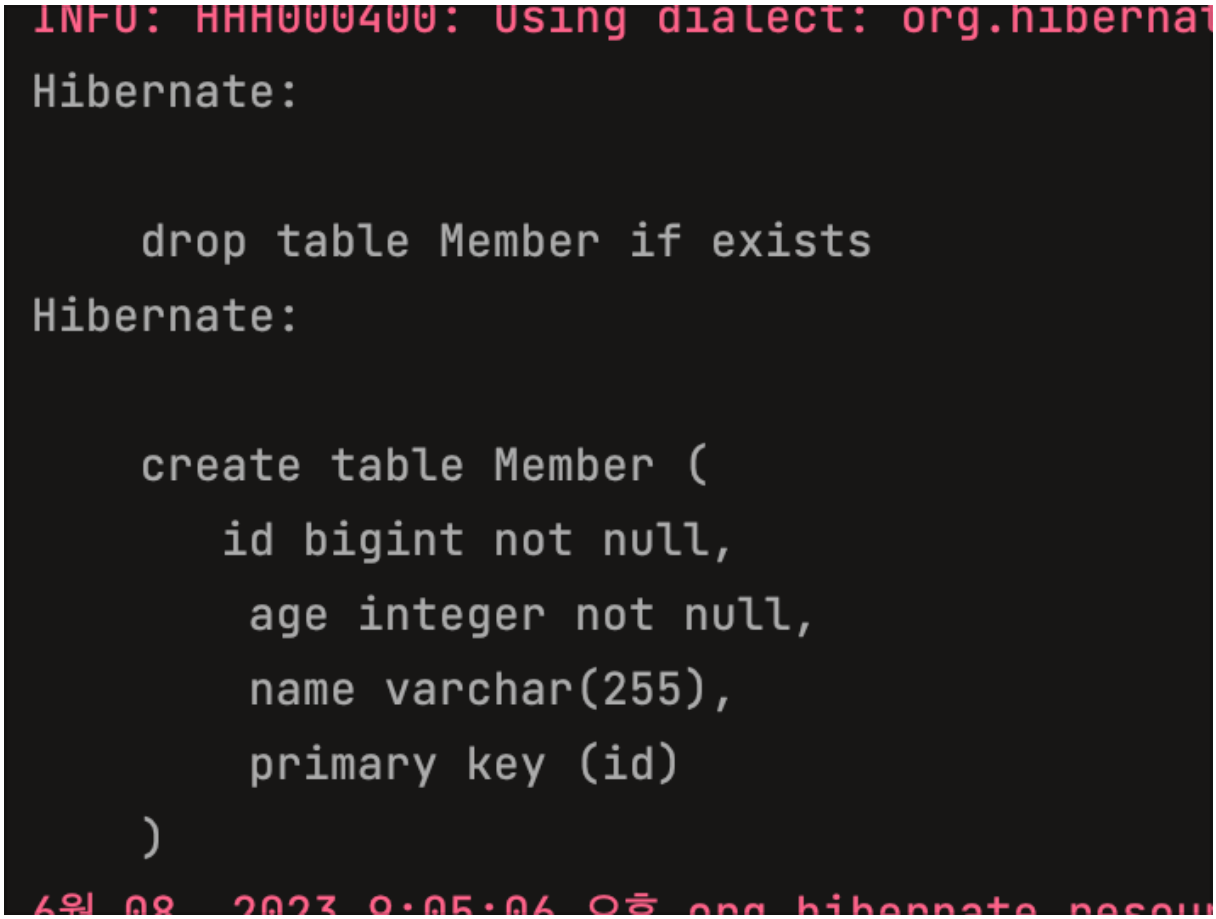
4. 엔티티 매핑

객체와 테이블 매핑

- 객체와 테이블 매핑 : @Entity가 붙으면 jpa가 관리. 이게 안붙으면 그냥 내가 쓰고싶은 걸로 쓸 수 있음.
  - 기본생성자필수
  - final, enum, interface, innner에는 @Entity 사용 불가.
  - 다른 패키지에 같은 이름이 있는 경우에는 클래스 이름 붙여주기 @Entity(name)
- 필드와 컬럼 매핑
- 기본 키 매핑
- 연관관계 매핑

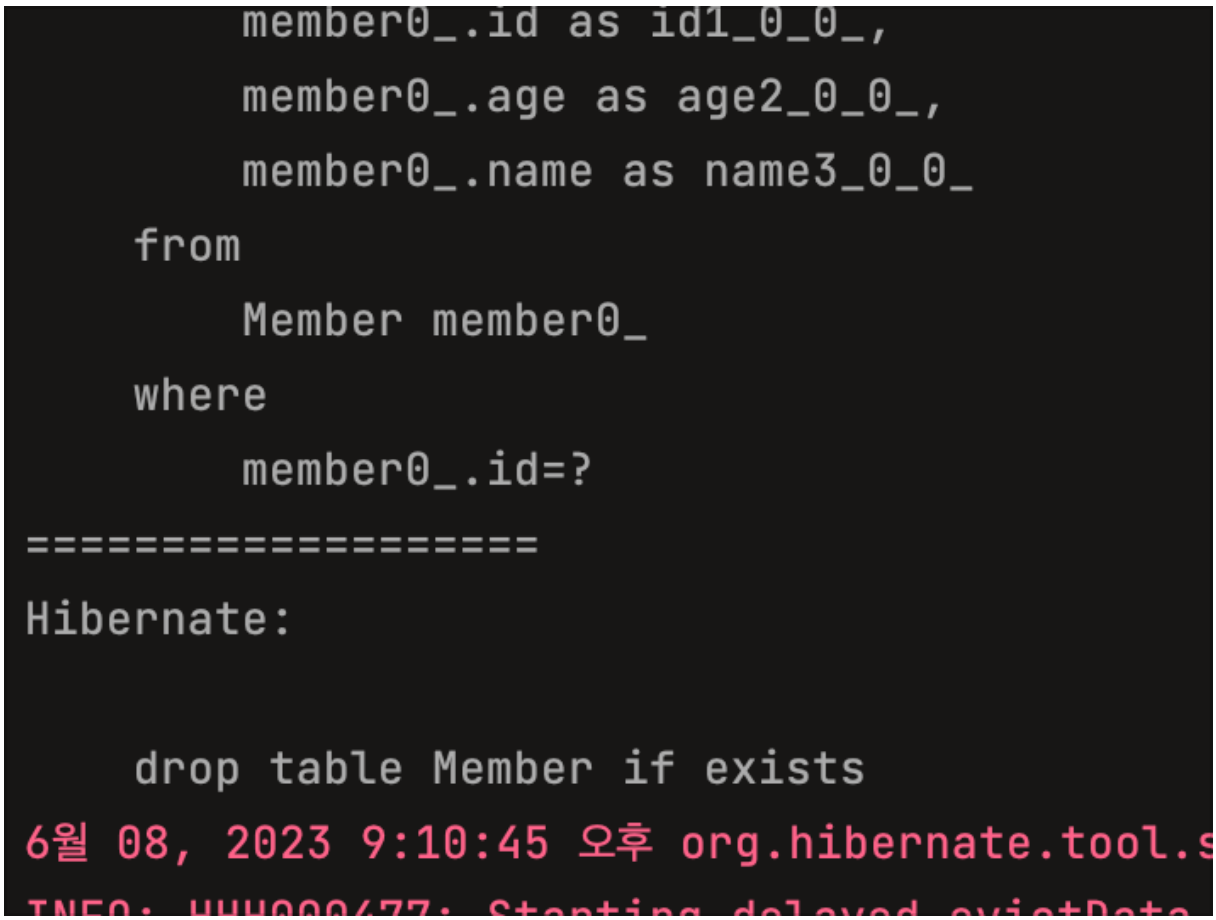
데이터베이스 스키마 자동 생성 기능

JPA에서는 애플리케이션 로딩 시점에 DB 테이블을 자동 생성하는 기능. (create문으로 테이블 생성)
객체 매핑을 다 해놓으면 앱 쓸때 그냥 쓸 수 있게 해줌.
DDL이 생성될 때 방언에 맞춰서 바로 적절한 DDL 생성. 개발 장비에서만 사용!!!! 운영 서버에서는 사용하지말자.
persistence.xml의 주석되어있는 부분.
create는 원래 있던 table을 drop하고 생성하기 때문에 주의하자!

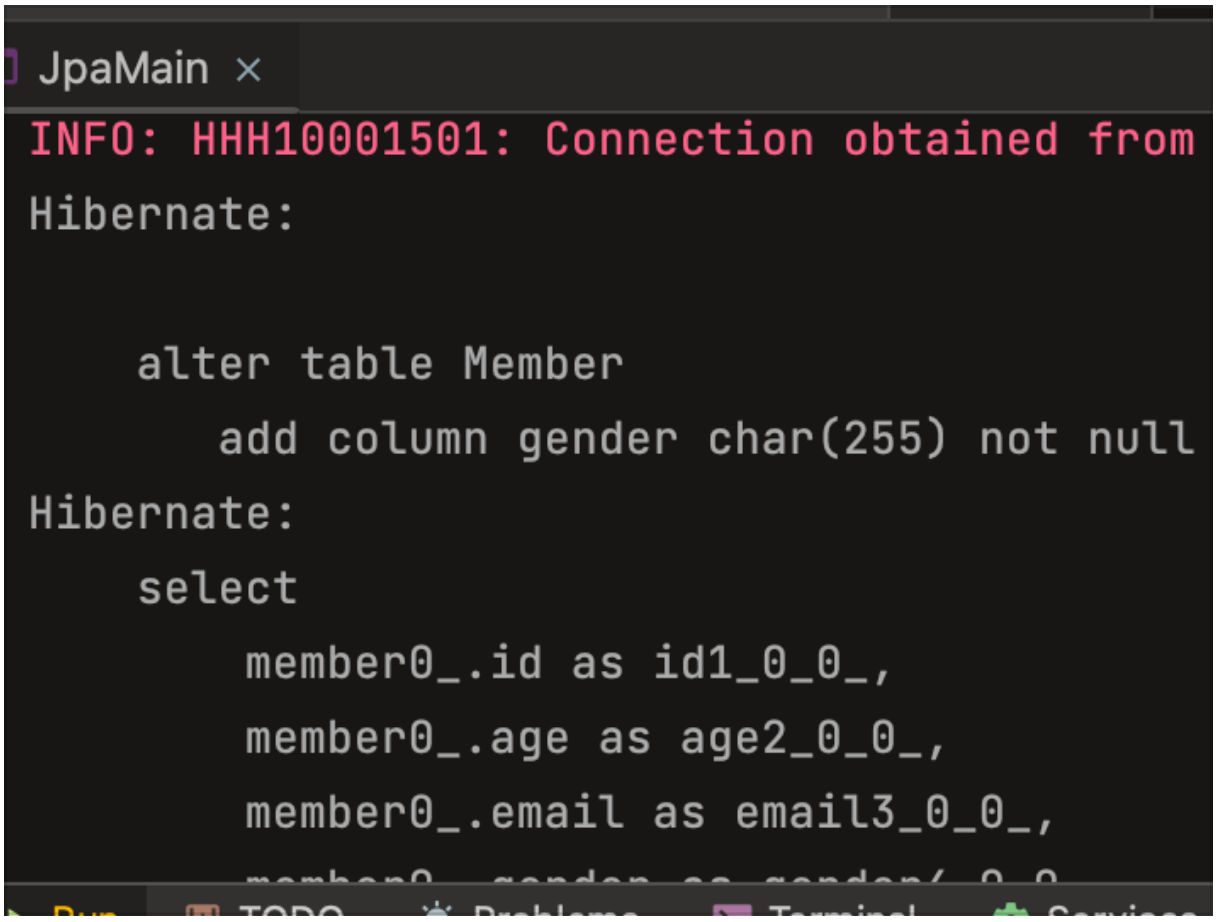


<property name="hibernate.hbm2ddl.auto" value="create" />의 value에 옵션들을 넣을 수 있다.

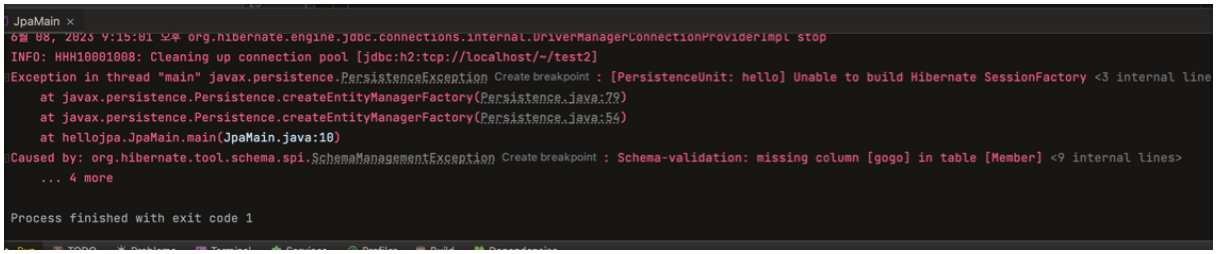
create-drop은 테스트한다음 바로 지우고 싶을 때 사용



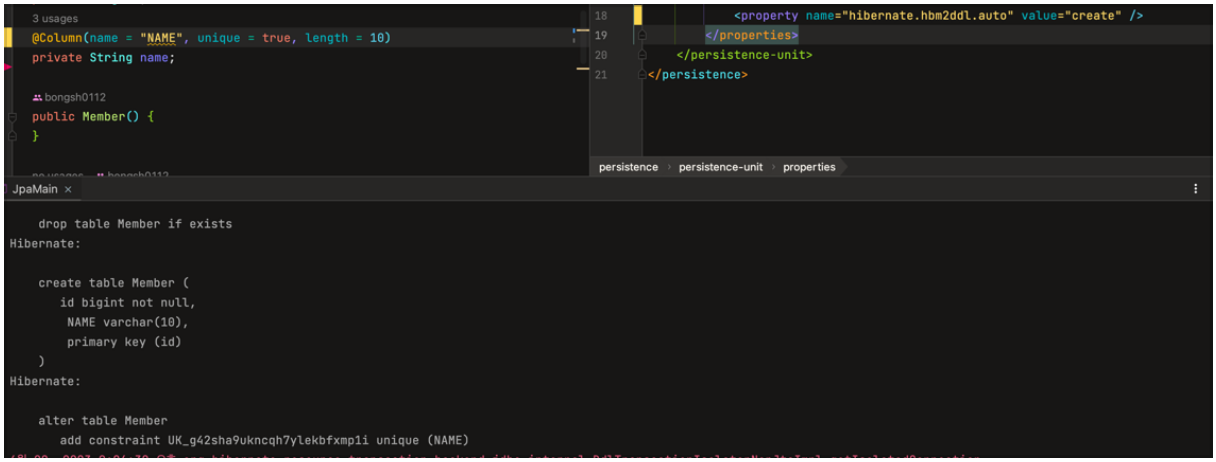
update에서 삭제는 안됨! 수정만 가능. 테이블의 변경사항이 있을 때 삭제하지 않고 alter(테이블 차원의 수정 / update는 객체 차원의 수정) 하고 싶을 때



validate는 엔티티와 테이블의 매핑 상태를 확인. gogo라는 컬럼은 매핑이 안되어있으므로 에러를 띄움.
➡ 검증이 무엇인가? 테이블에 이미 있는 컬럼들만 넣고 실행했을 때 오류가 없어야 함. 즉 그냥 just 확인임. DB를 건들거나 그렇지 않다!



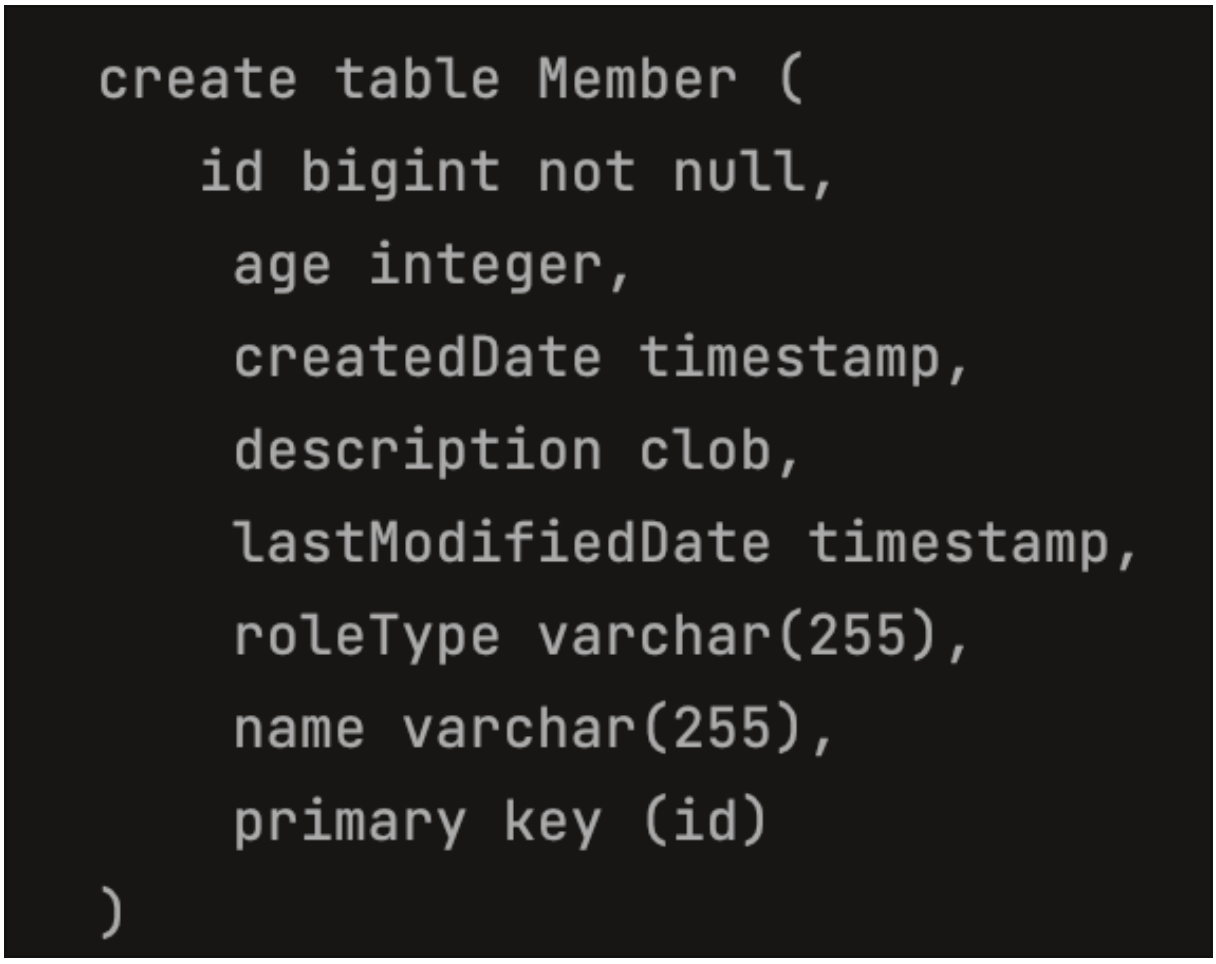
만약에 dialect에 다른 방언을 넣으면 그 방언에 따라서 적절하게 해석하여 실행해준다. create 테이블이라던지, drop, alter라던지…



DDL 생성 기능…! @Column에 속성을 주고 create로 drop & create table을 하니까 데이터베이스에 영향을 주는 쿼리문들이 적용됨.
만약 테이블의 이름을 바꾼다면 런타임에 영향을 주겠지만 우리가 한 속성을 준 것은 그렇지 않으므로 이러한 기능을 제공.

필드와 컬럼 매핑

enum타입은 디비에 비슷한건 있으나 똑같은건 없다!!! 따라서 여기서 써줄 땐 @Enumerated 꼭 써주기
TemporalType : datetime은 자바에만 있고 디비에는 날짜, 시간을 구분해서 씀. 따라서 datetime을 쓸 거면 꼭 @TemporalType써주기
@Lob : 데이터베이스에 varchar를 넘어서는 큰 컨텐츠 넣기.



clob, blob 찾아보기

@Transient는 디비에 안넣고 메모리에서만 쓰기

@Column

- insertable, updatable : 수정에 대한 권한 가능 여부
- nullable : false를 주면 not null!
  - hibernate의 경우 notnull이면 먼저 체크도 해줌
- unique : 유니크 걸어주기. 잘 안쓰긴하는데 왜 안쓰냐? 이상한 이름이 나오기 때문. 위에서 g42sha…이런거임.
  - 만약 주려면 @Table에서 uniqueConstraints에서 이름으로 주자
- length = 10 : varchar10으로 됨.
- columnDefinition = "varchar(100) default 'EMPTY'" ➡ 디비 컬럼 정보를 직접 줄 수 잇음

@Enumerated

- ORDINAL : enum 순서를 데이터베이스에 저장.

JpaMain.java × persistence.xml × Member.java × Enumerated.class ×

Decompiled .class file, bytecode version: 52.0 (Java 8)

```
1   /.../
5
6   package javax.persistence;
7
8   import ...
12
13  @Target({ElementType.METHOD, ElementType.FIELD})
14  @Retention(RetentionPolicy.RUNTIME)
15  public @interface Enumerated {
16    EnumType value() default EnumType.ORDINAL;
17  }
```

실행 | Run Selected | 자동 완성 | 지우기 | SQL 문:

```
SELECT * FROM MEMBER
```

```
SELECT * FROM MEMBER;
```

ID	AGE	CREATEDDATE	DESCRIPTION	LASTMODIFIEDDATE	ROLETYPE	NAME
1	*null*	*null*	*null*	*null*	0	B

(1 row, 3 ms)

편집

디폴트가 오디널임

- STRING : enum이름을 데이터베이스에 저장.

실행 Run Selected 자동 완성 지우기 SQL 문:

```
SELECT * FROM MEMBER
```

SELECT * FROM MEMBER;

ID	AGE	CREATEDDATE	DESCRIPTION	LASTMODIFIEDDATE	ROLETYPE	NAME
2	*null*	*null*	*null*	*null*	USER	B

(1 row, 2 ms)

편집

=> ORDINAL을 쓰면 안되는 이유는 딱 알겠다. 그냥 직관적이지가 않고 순서만 보이기 때문에. 또 만약 enum에서 USER, ADMIN이었는데 하나가 더 추가돼서 GUEST, USER, ADMIN이 되면 GUEST가 0이 되므로 원래 0이었던 USER와 겹친다.

기본 키 매핑

@Id : 직접 할당

@GeneratedValue : 자동 할당

- strategy
  - GenerationType.IDENTITY : 기본키 생성을 mysql에 위임함.

실행 Run Selected 자동 완성 지우기 SQL 문:

```
SELECT * FROM MEMBER
```

SELECT * FROM MEMBER;

ID	NAME
1	C
2	C

(2 행, 2 ms)

  - GenerationType.SEQUENCE : 시퀀스 오브젝트를 만들어서 데이터베이스의 시퀀스 오브젝트에 따라서 값을 만들어냄.

- sequence는 id가 숫자값이어야한다!! 위에꺼랑 다른건 없으나 위는 id가 String이라는 것이 다름.
- sequencegenerator는 두번째 사진.

실행 Run Selected 자동 완성 지우기 SQL 문:

```
SELECT * FROM MEMBER
```

```
SELECT * FROM MEMBER;
```

ID	NAME
1	C
2	C

(2 행, 3 ms)

실행 Run Selected 자동 완성 지우기 SQL 문:

```
SELECT * FROM MEMBER
```

```
SELECT * FROM MEMBER;
```

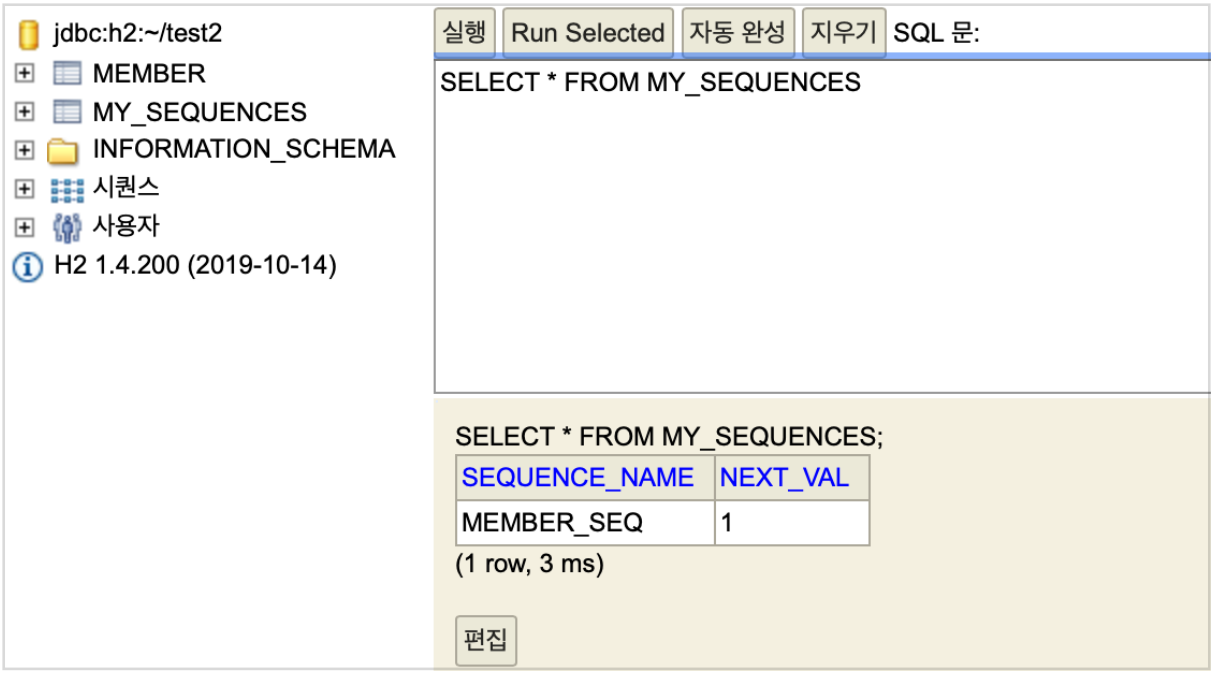
ID	NAME
1	C
52	C
102	C

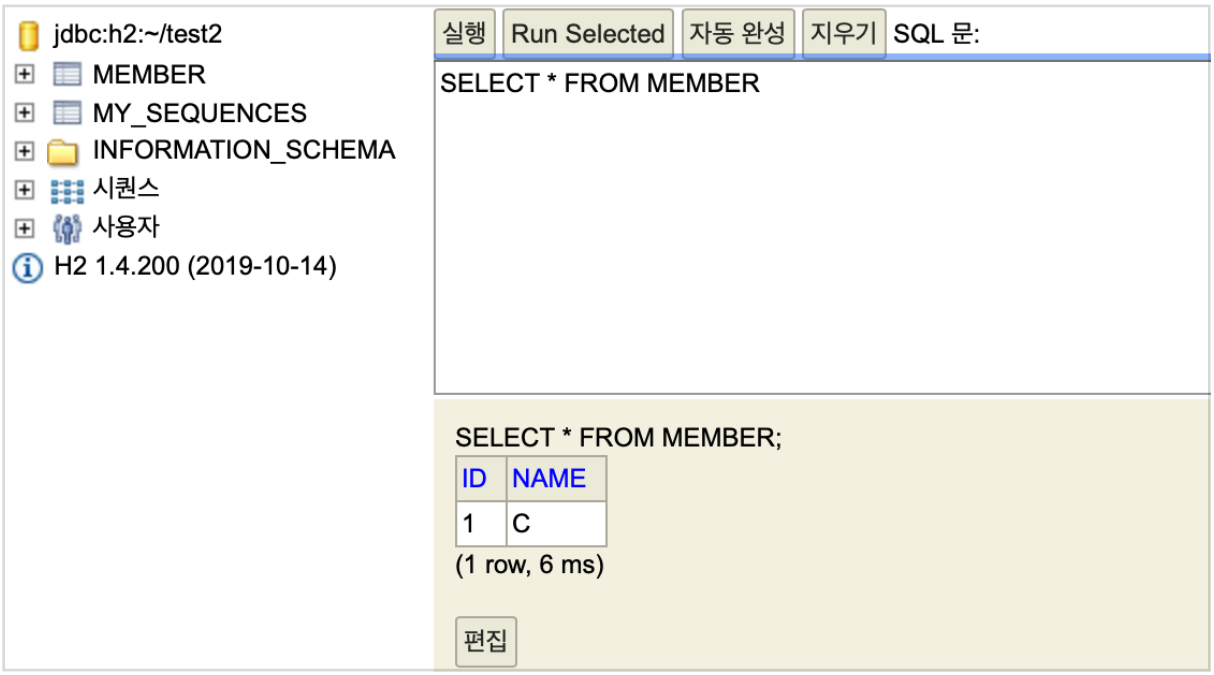
(3 행, 2 ms)

```
6월 08, 2023 11:10:40 오후 org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
6월 08, 2023 11:10:40 오후 org.hibernate.resource.transaction.backend.j
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org
Hibernate:
    call next value for member_seq
Hibernate:
    /* insert hellojpa.Member
        */ insert
        into
            Member
            (name, id)
        values
            (?, ?)
```

- @Table 전략의 매핑 : 어떤 데이터베이스는 오토 인크리먼트, 어떤 디비는 시퀀스… 하지만 이거는 모든 데이터 베이스에 적용 가능하다. 키 생성 전용 테이블을 만들기 때문. 아마 자신만의 방법이 있는 듯 하다.
  - 지정해준 이름의 테이블에 키값들이 모여있음. 그리고 그 키값들의 모임에서 키값들을 뽑아 자동 할당. 운영에서는 아무래도 테이블은 좀 부담스럽다. 디비에서 정해주는 관례에 따라서 해보자.
    - initialValue, allocationValue만 좀 중요함

```
        id bigint not null,
         name varchar(255) not null,
         primary key (id)
    )
Hibernate:

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

    insert into MY_SEQUENCES(sequence_name, next_val) values ('MEMBER
```





권장하는 식별자 전략

변하면 안된다가 제일 어려움. 정말 먼 미래까지 변하지 않아야함. 애플리케이션이 없어질때까지 변하면안됨. 예를 들어 주민번호? 하지만 미래까지 이 조건을 만족하는 자연키는 찾기 어렵다.
대신에 대리키(비즈니스와 정말 상관없는 것. 주민번호같은 건 정말 적절하지못함.)
권장에 나오는걸 추천! ex) uuid아니면 시퀀스… 절대 비즈니스에 관련된걸 키로 끌고오지말자!

IDENTITY 전략에서 애매한것

이 전략은 내가 pk에 값을 넣지 않아야 함. 그래서 db에 들어가 봐야 id가 뭔지 알 수 있음.

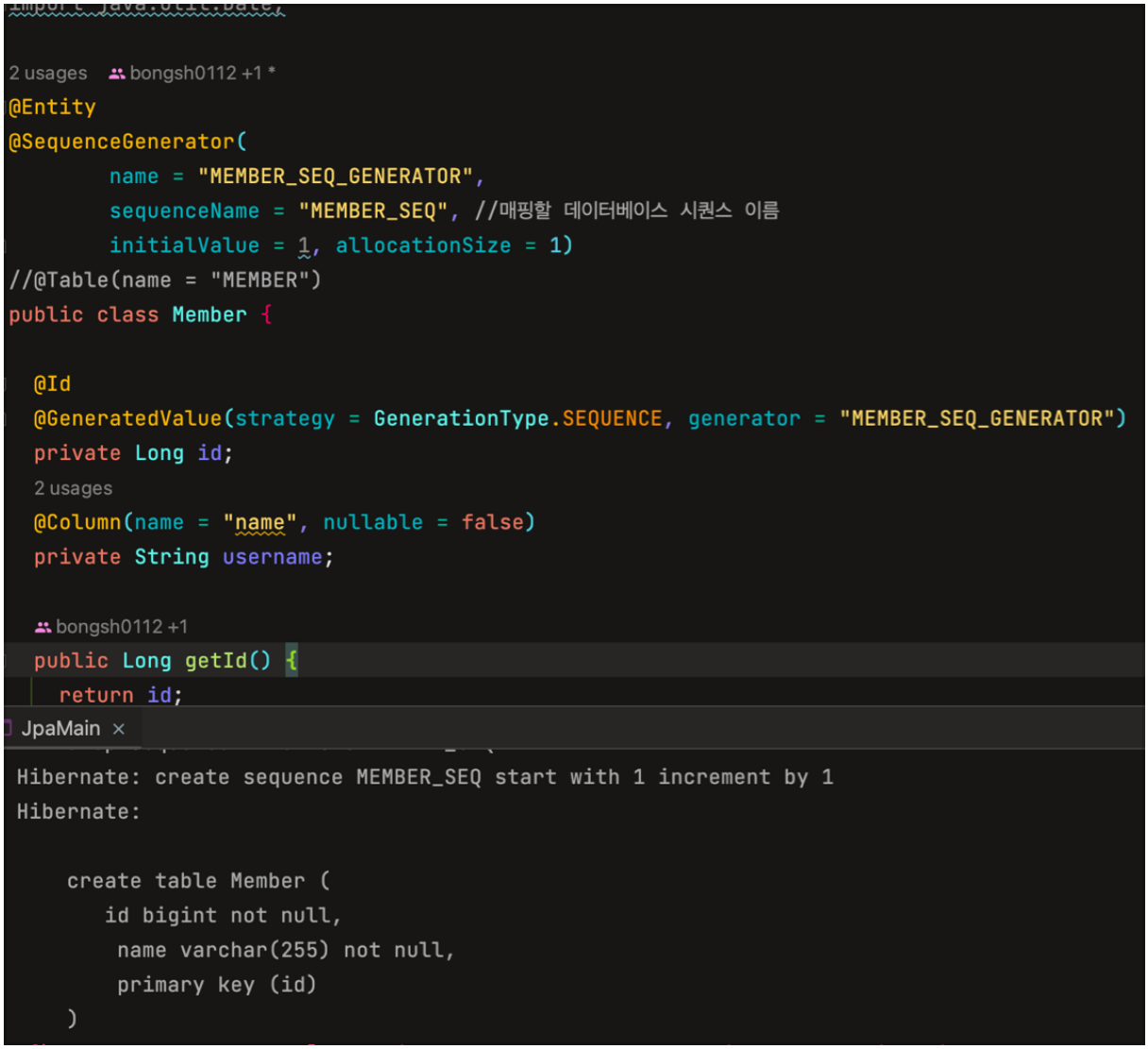
영속성 컨텍스트에서 관리되려면 무조건 pk값을 알아야 하는데 이것은 그렇지 않음. 영속성 ppt의 엔티티 조회, 1차 캐시를 참조하면 @id를 알아야 캐시에서 관리 가능!!
그러나 IDENTITY 전략에선 불가능…
그래서 이 전략에서만 em.persist(member)를 호출한 시점에 db에 insert 쿼리를 날려보냄. 보통은 commit 시점이지만 이것은 좀 특별함.
identity 전략은 db에 넣어봐야 id를 알 수 있기 때문에
=> 밑 그림처럼 persist 하자마자 insert query를 바로 날림
그러면 디비에서 generatevalue로 id를 생성. 그리고 나서 jpa 내부적으로 그 id를 찾아가지고 가져옴.
=> 그래서 시간 지연 저장소에 모아서 한꺼번에 분출하는 그런 기능이 이 전략에서는 불가능!
이 전략이 아니면 그렇지 않음!!

```java
EntityTransaction tx = em.getTransaction();
tx.begin();
try {

  Member member = new Member();
  member.setUsername("C");

  System.out.println("====================");
  em.persist(member);
  System.out.println("member.id = " + member.getId());
  System.out.println("====================");

  tx.commit();
} catch (Exception e) {
  tx.rollback();
} finally {
    em.close();
}
```

JpaMain ×

```
    /* insert hellojpa.Member
        */ insert
        into
            Member
            (id, name)
        values
            (null, ?)
member.id = 1
====================
6월 08, 2023 11:31:42 오후 org.hibernate.engine.jdbc.connection
INFO: HHH10001008: Cleaning up connection pool [jdbc:h2:tcp:/
```

SEQUENCE 전략의 특징

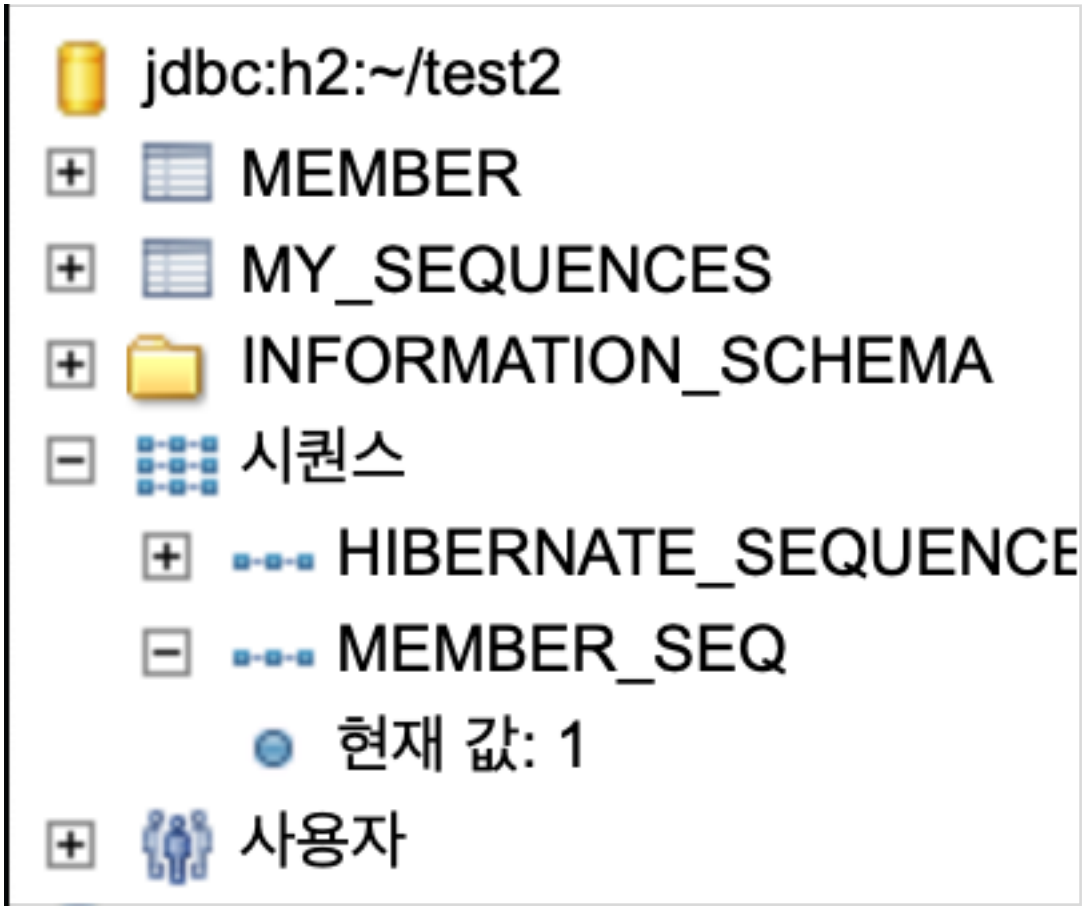


```
2 usages  bongsh0112 +1 *
@Entity
@SequenceGenerator(
        name = "MEMBER_SEQ_GENERATOR",
        sequenceName = "MEMBER_SEQ", //매핑할 데이터베이스 시퀀스 이름
        initialValue = 1, allocationSize = 1)
//@Table(name = "MEMBER")
public class Member {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "MEMBER_SEQ_GENERATOR")
  private Long id;
  2 usages
  @Column(name = "name", nullable = false)
  private String username;

  bongsh0112 +1
  public Long getId() {
    return id;
```

```
JpaMain ×
Hibernate: create sequence MEMBER_SEQ start with 1 increment by 1
Hibernate:

    create table Member (
       id bigint not null,
        name varchar(255) not null,
        primary key (id)
    )
```

위에서 start with~~~은 옵션에서 주어질 수 있음. 이니셜 밸류와 얼로케이션사이즈가 그것.

jpa 동작 방식



영속성 컨텍스트에 들어가기 위해서 ==== 사이에 있는게 실행됨.

```
    drop sequence if exists MEMBER_SEQ
Hibernate: create sequence MEMBER_SEQ start with 1 increment by 1
Hibernate:

    create table Member (
       id bigint not null,
        name varchar(255) not null,
        primary key (id)
    )
6월 08, 2023 11:35:38 오후 org.hibernate.resource.transaction.backend
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [or
6월 08, 2023 11:35:38 오후 org.hibernate.resource.transaction.backend
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [or
6월 08, 2023 11:35:38 오후 org.hibernate.tool.schema.internal.SchemaC
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema
=====================
Hibernate:
    call next value for MEMBER_SEQ
member.id = 1
=====================
Hibernate:
    /* insert hellojpa.Member
        */ insert
        into
            Member
            (name, id)
        values
            (?, ?)
```

next call

pk값만 MEMBER_SEQ에서 얻고 나중에 commit 할 때 쿼리날림.

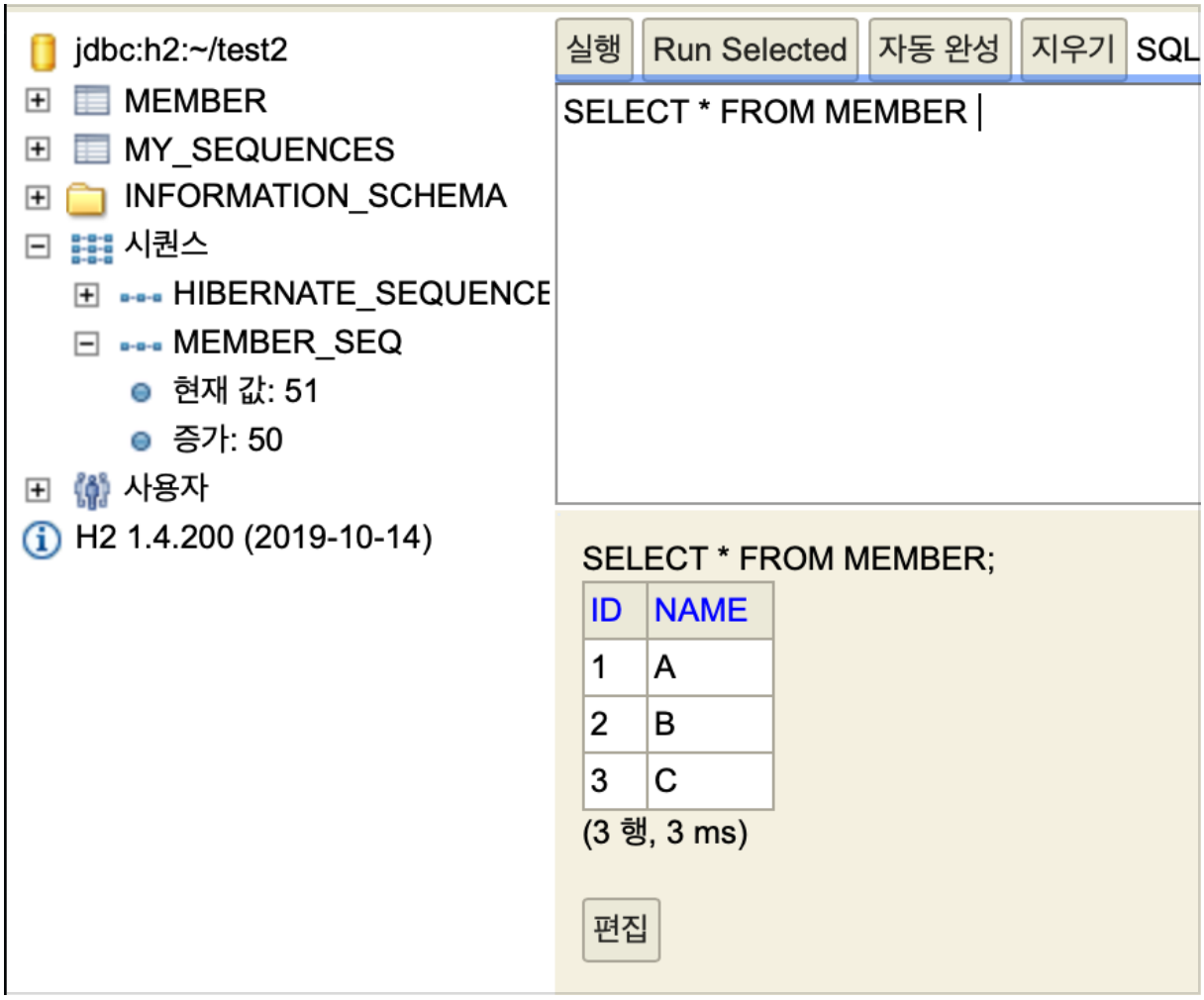
이 전략의 얼로케이션 사이즈가 50이 디폴트인 이유 : 여러 객체를 저장할 때 마다 굳이 계속 next call로네트워크를 써야하나???에 대한 답변을 위함. db에는 미리 먼저 50개 올려놓고 메모리 상에서는 1씩 씀. 그러고 50개가 끝나면 다시 next call을 해서 51~100쓸수있게… 이러면 여러 웹 서버가 있어도 동시성 이슈 없이 다양한 문제가 해결됨.

```
System.out.println("=====================");
                        //                  app에서 씀
em.persist(member); // DB SEQ = 1    |    1
em.persist(member2); // DB SEQ = 51 |    2
em.persist(member3); // DB SEQ = 51 |    3
```

```
6월 08, 2023 11:51:32 오후 org.hibernate.resource.tr
INFO: HHH10001501: Connection obtained from JdbcCo
6월 08, 2023 11:51:32 오후 org.hibernate.tool.schema
INFO: HHH000476: Executing import script 'org.hibe
=====================
Hibernate:
    call next value for MEMBER_SEQ
Hibernate:
    call next value for MEMBER_SEQ
member.id = 1
member.id = 2
member.id = 3
=====================
Hibernate:
    /* insert hellojpa.Member
        */ insert
        into
```

=> 처음에 50개씩 써야하는데 1밖에없음 ➡ 51까지 호출(dummy call). 현재 내 시퀀스는 1(앱이쓰는 거). 그다음은 2(앱) 2, 3부터는 MEM에서 콜함.

=> 앞으로 4, 5 쭉 돌리면 next call 호출안됨. 그러나 51까지 가면 next call 호출.


jdbc:h2:~/test2
MEMBER
MY_SEQUENCES
INFORMATION_SCHEMA
시퀀스
HIBERNATE_SEQUENCE
MEMBER_SEQ
현재 값: 51
증가: 50
사용자
H2 1.4.200 (2019-10-14)
실행
Run Selected
자동 완성
지우기
SQL
SELECT * FROM MEMBER
SELECT * FROM MEMBER;
ID
NAME
1
A
2
B
3
C
(3 행, 3 ms)
편집

#JPA