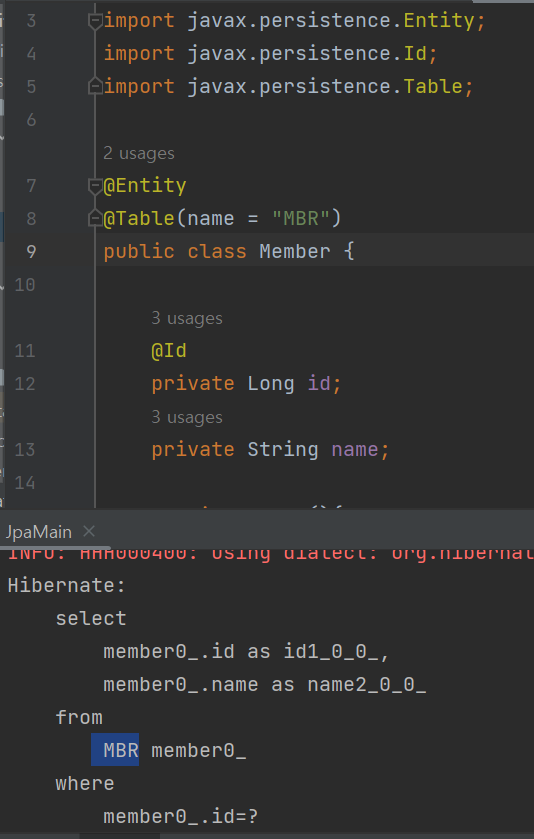
**#section 4**

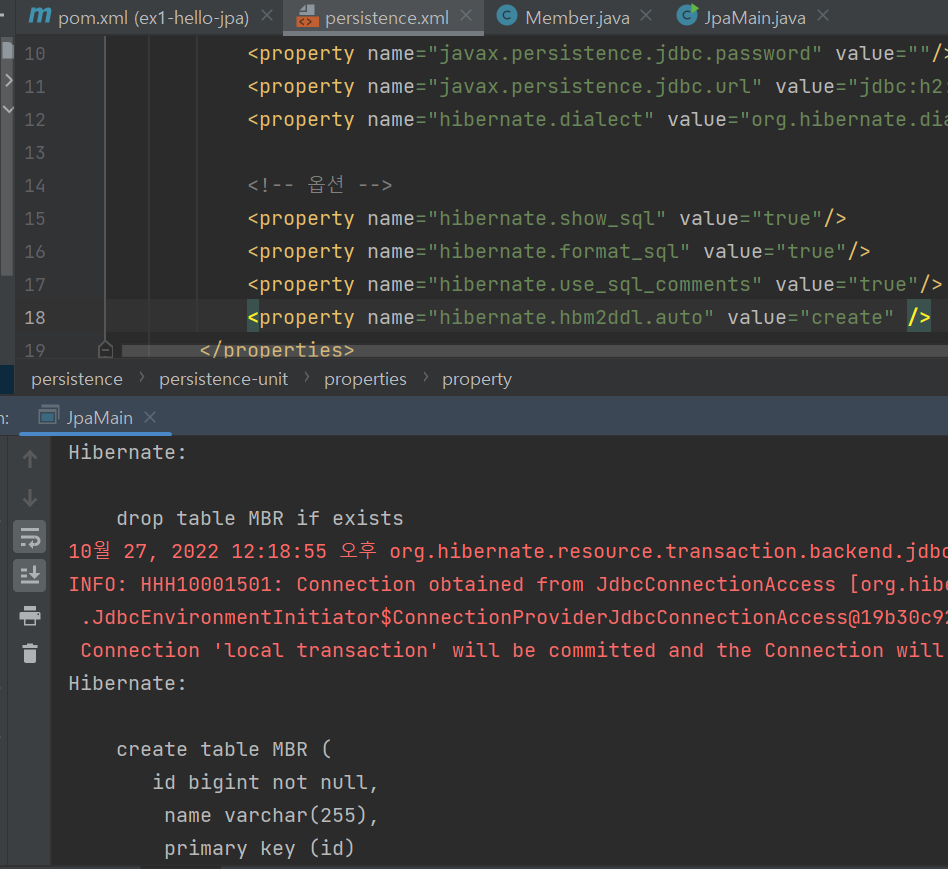
**1. 객체와 테이블 매핑**

- table 이름 바꾸기

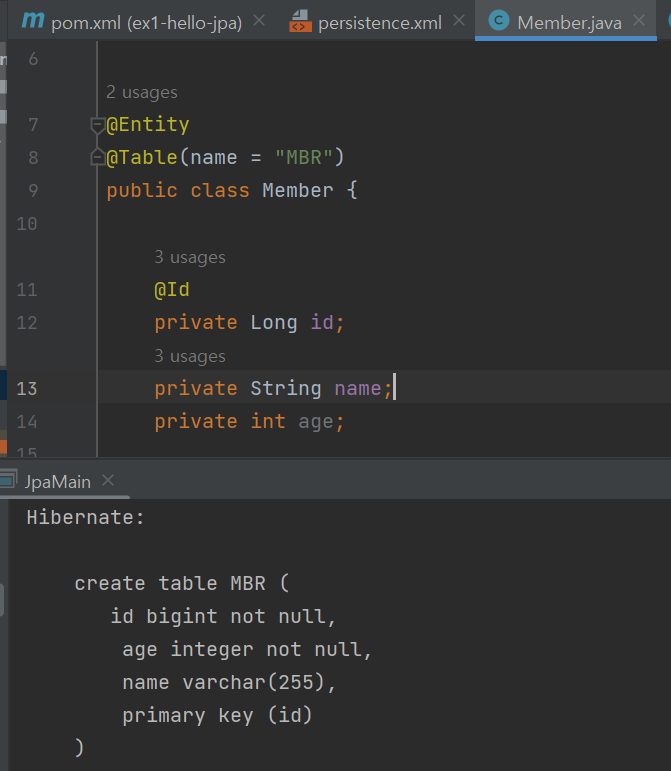
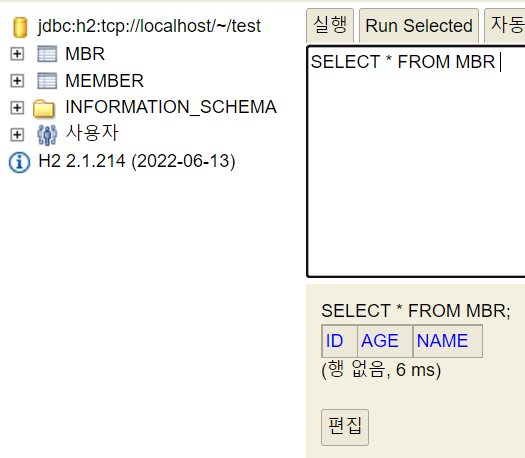


- 스키마 생성 : 데이터베이스 방언을 활용해서 데이터 베이스에 맞는 적절한 DDL 생성. 개발 장비에서만 사용

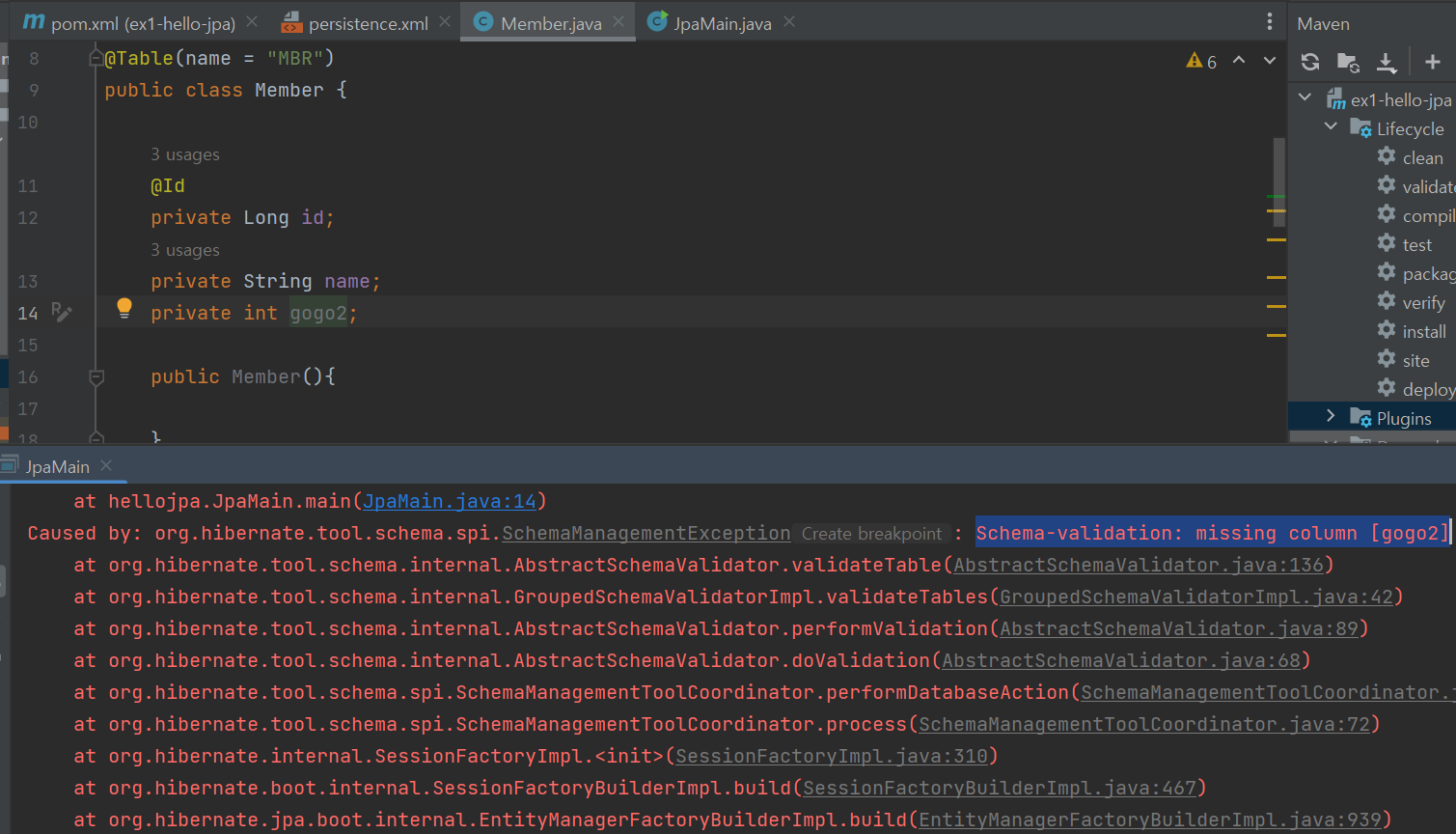
- create : 기존 테이블 삭제 후 다시 생성

- column 추가 ex) age 변수 추가

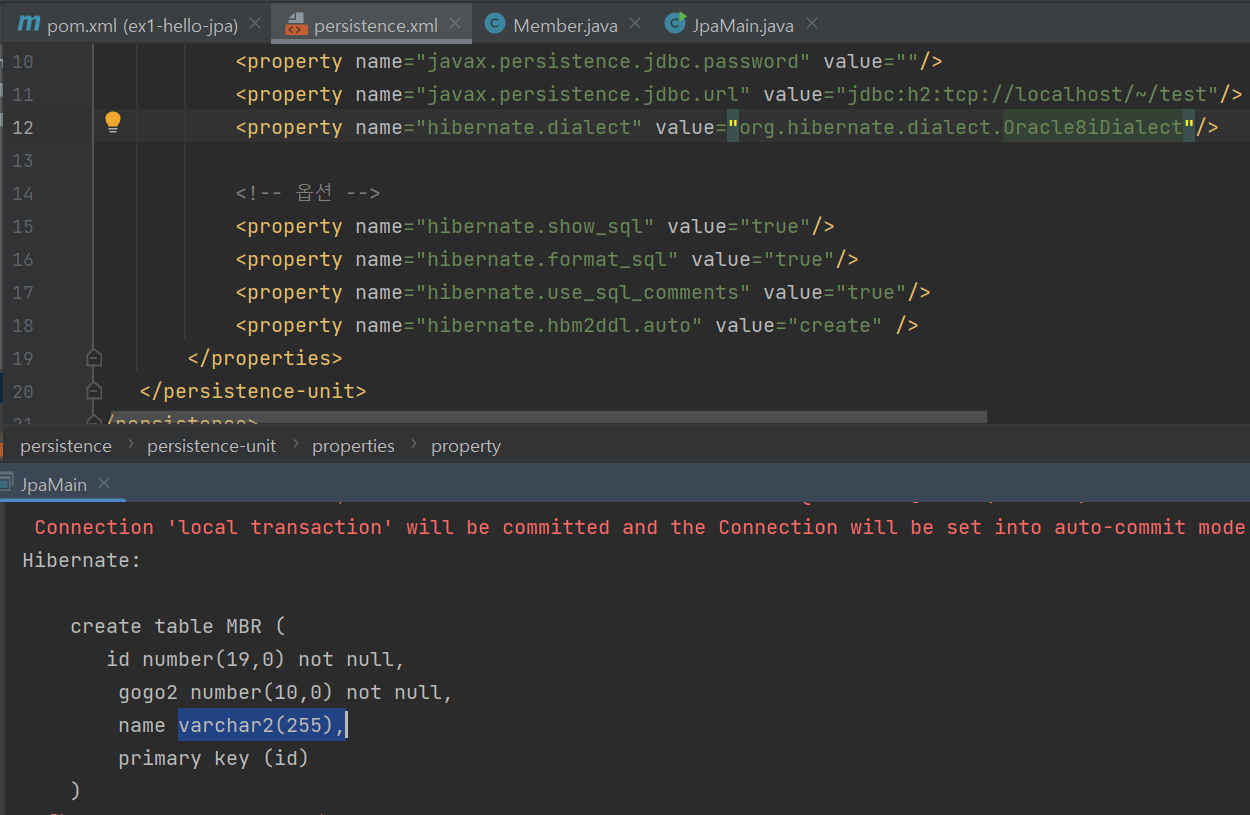
 

-validate



**2. 데이터베이스 스키마 자동 생성 – 실습>**

- 데이터 방언 별로 달라지는 것을 확인 varchar2

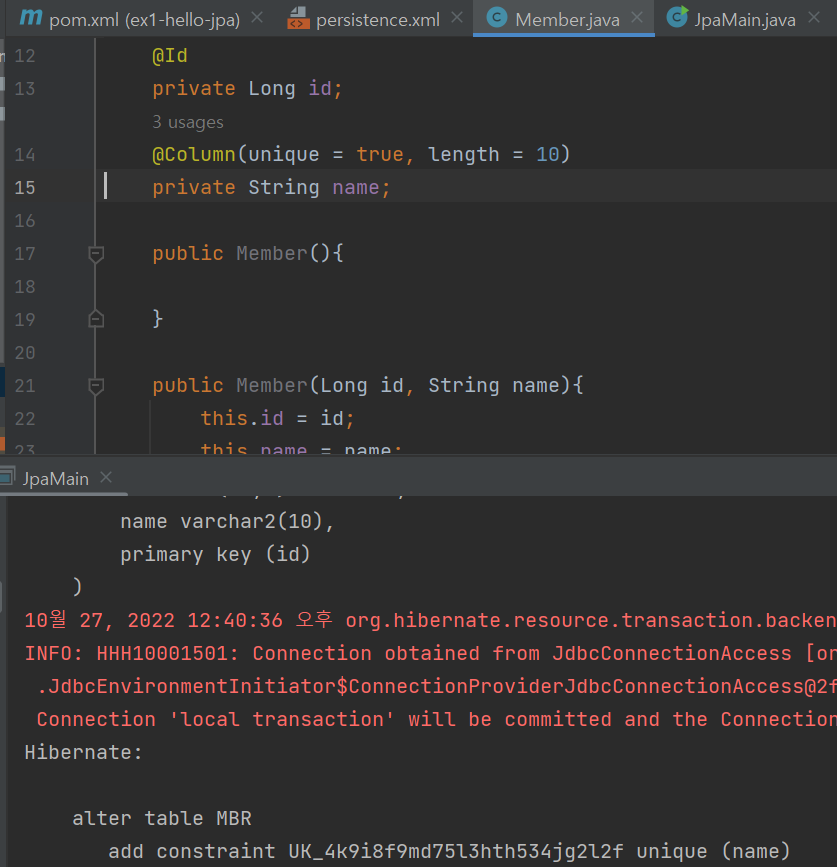


- 데이터베이스 스키마 자동 생성 주의점 : 운영 장비에는 create, create-drop, update 금지!

개발 초기 단계 – ~~create~~, drop // 테스트 서버 – ~~update~~, validate //

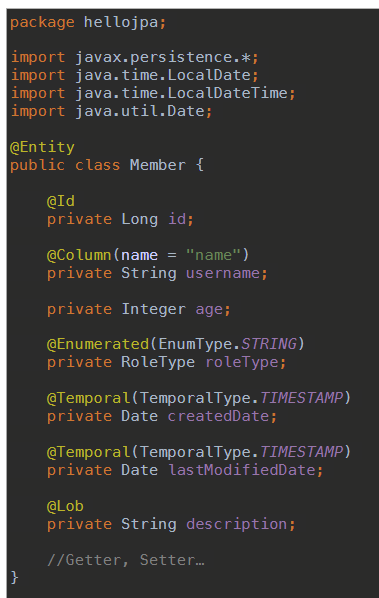
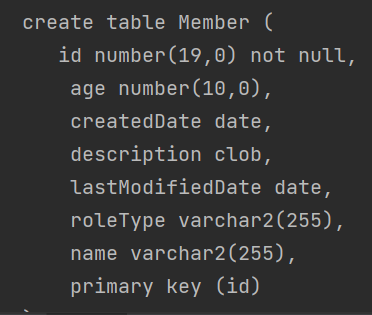
스테이징과 운영 서버 – validate, none

- DDL 생성 기능 : JPA 실행 로직에는 영향을 주지 않는다.

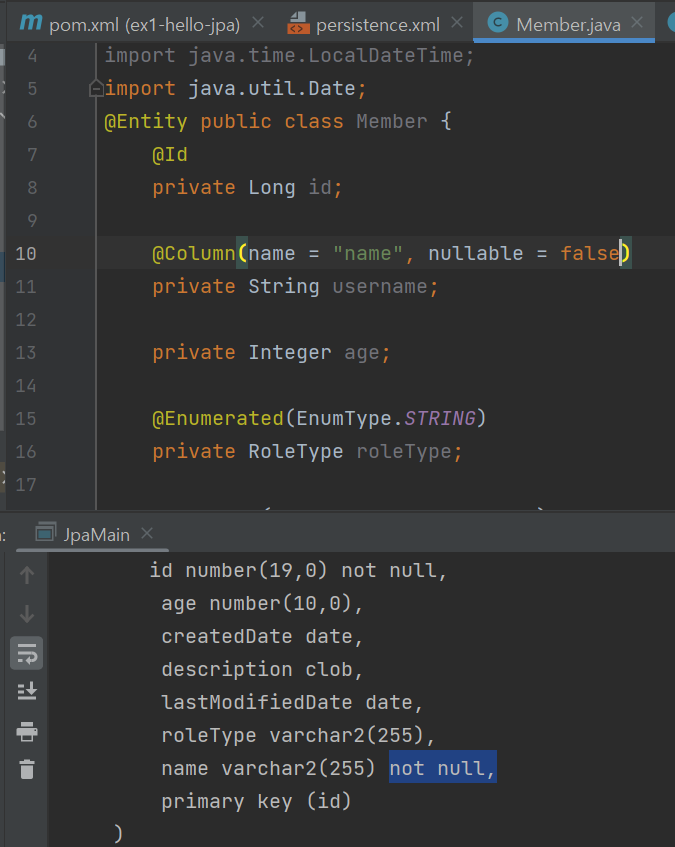


**3. 필드와 컬럼 매핑]**

- 요구사항 추가 실습

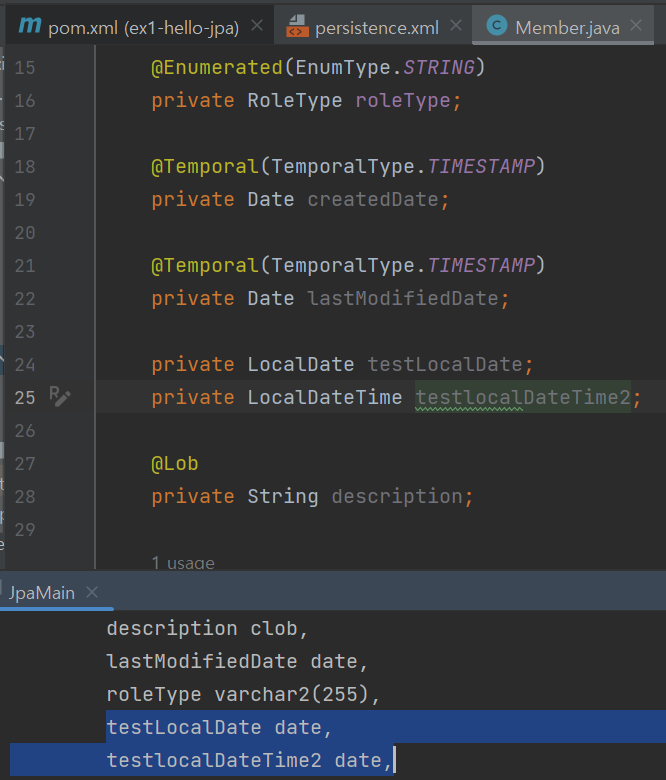
  결과

column 속성이 가장 중요

 -> not null 많이 씀~

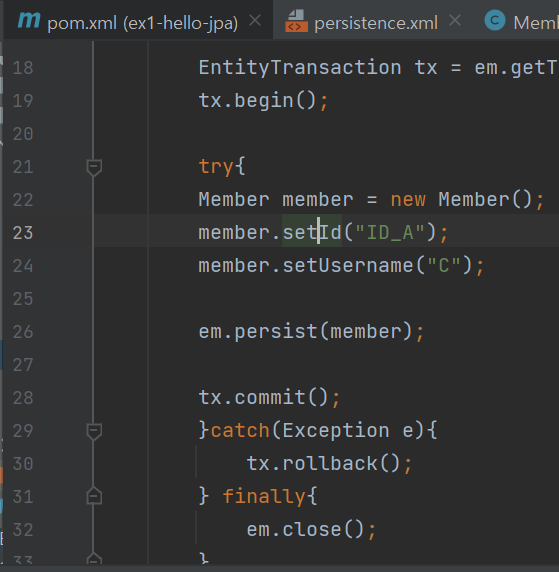
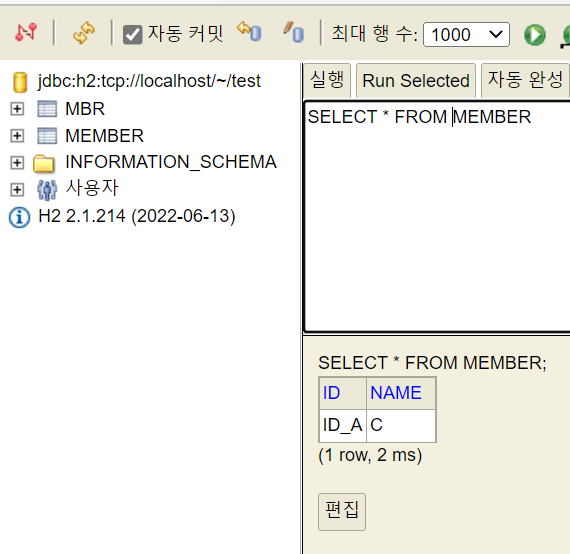
@Enumerated : 주의! ORDINAL 사용X

temporal

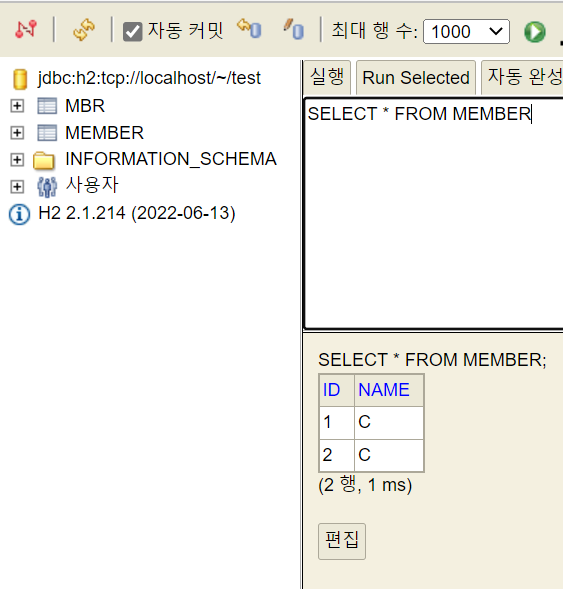


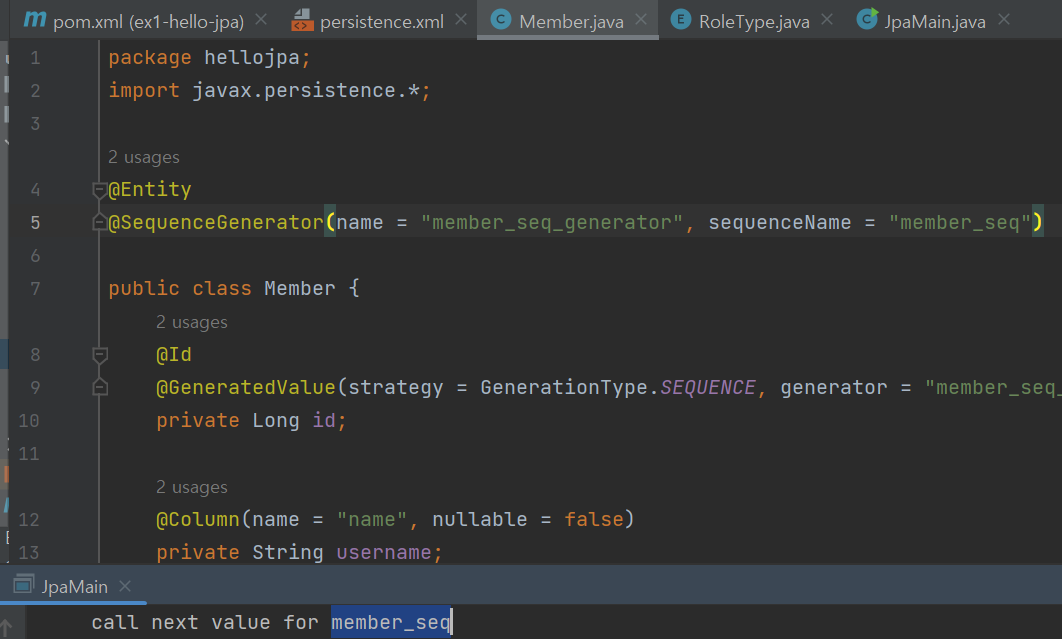
4. 기본 키 매핑

- 기본 키 매핑 방법 : ID 할당

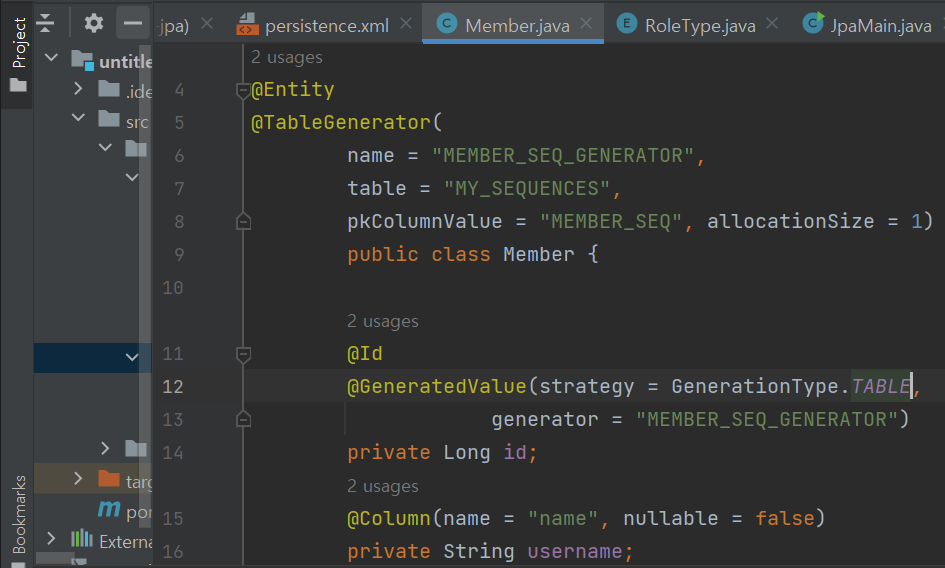
 

- sequnece 전략





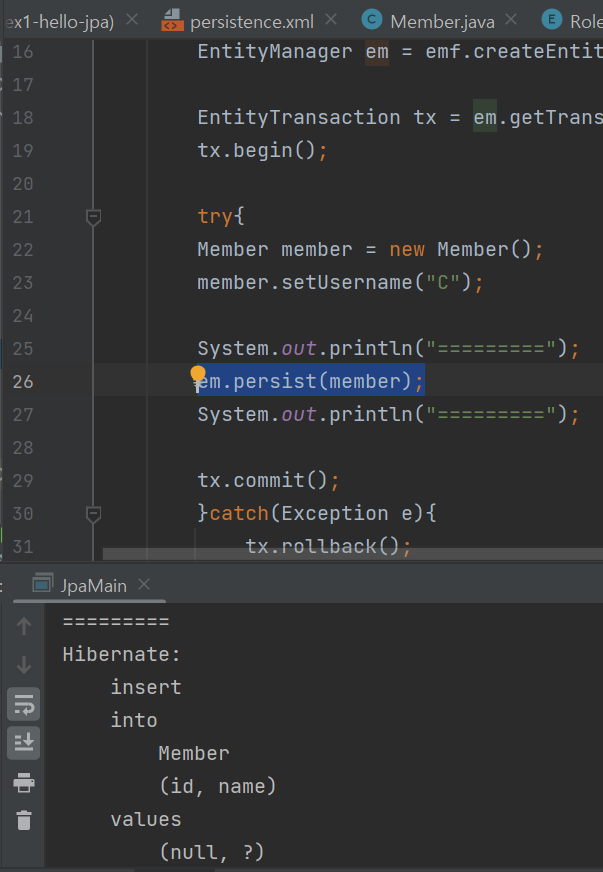
- table 전략



- 권장하는 식별자 전략

자연키 = 식별번호 = long형+ 대체키+ 키 생성전략 사용 (주민등록번호는 nope)

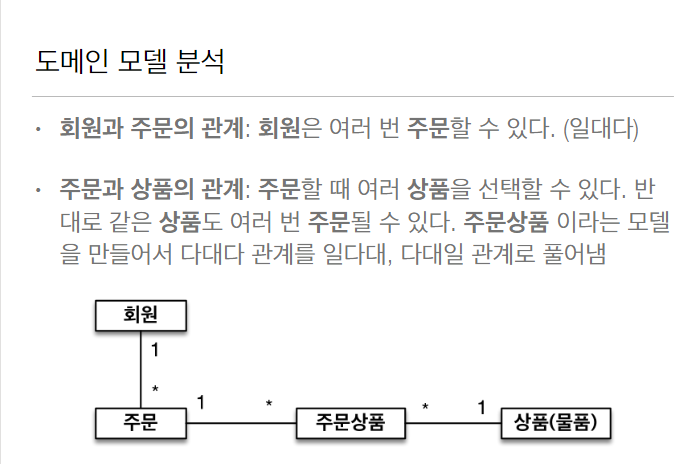
commit 시점이 아닌 em.persist(member); 한 시점에 insert 쿼리를 바로 날림



-allocationSize : 시퀀스 한 번 호출에 증가하는 수(성능 최적화에 사용됨) 기본값 = 50

미리 데이터 값을 올려두는 방식

\*\*<실전예제 - 쇼핑몰>



# section 5

중요 내용 : 객체의참조와 테이블의 외래키를 매핑 / 연관관계의 주인

-연관계가 필요한 이유 : 협력 공동체를 만들기

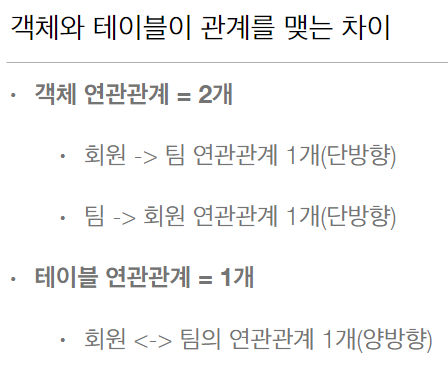
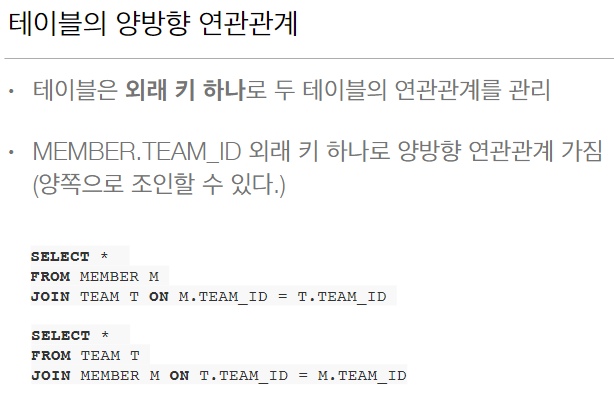
- 객체를 테이블에 맞추어 데이터 중심으로 모델링하면, 협력 관계를 만들 수 없다.

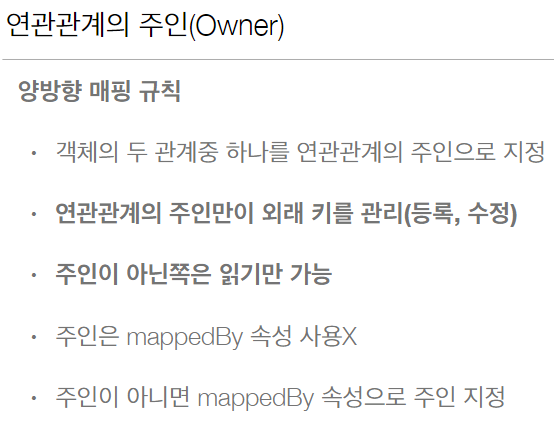
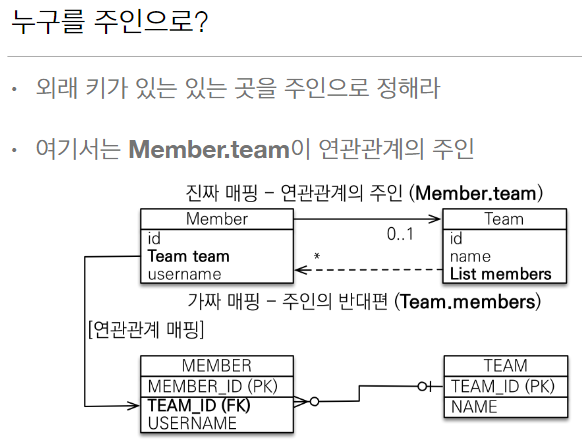
- 단방향 연관관계

**<양방향 연관관계와 연관관계의 주인>**

- 양방향 매핑 : 양방향으로 데이터를 얻을 수 있게, 외래키 하나로 양방향이 모두 존재함. List members가 있어야 함.

- 연관관계와 주인의 mappedBy : 객체와 테이블 간의 연관관계를 맺는 차이를 이해해야 한다.

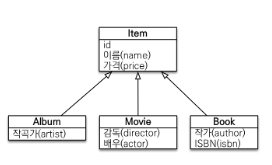
왜래키가 있는 곳이 多, 주인 ex)자동차가 아닌 바퀴!

 양방향 연관 시 양쪽 모두에 값 세팅

**<고급 관계>**

**1. 상속 관계 매핑**

- ORM에서 이야기하는 상속 관계 매핑은 객체의 상속 구조와 데이터베이스의 슈퍼타입 서브타입 관계를 매핑하는 것이다.



**- 슈퍼타입, 서브타입 논리 모델을 물리 모델 구현 방법**

각각의 테이블로 변환 : 슈퍼타입, 서브타입 테이블을 각각 생성하여 조회할 때 조인을 사용한다.

통합 테이블로 변환 : 테이블을 하나만 사용해서 통합한다. (단일 테이블 전략)

서브타입 테이블로 변환 : 서브 타입마다 하나의 테이블을 만든다. (구현 클래스 테이블 전략)

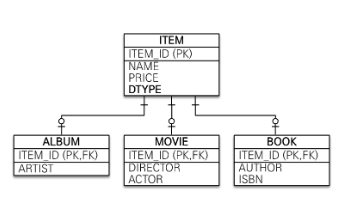
**- 주요 어노테이션**

조인 전략 : 자식 테이블이 부모 테이블의 기본 키를 받아서 기본키+외래키로 사용하는 전략

TYPE을 구분하는 DTYPE 컬럼을 추가해야한다는 점을 유념하자.

- 장점 : 테이블 정규화가 가능, 외래 키 참조 시 무결성 제약조건 활용 가능, 저장 공간 효율화

- 단점 : 조회 시 조인을 많이 사용하여 성능 저하, 조회 쿼리가 복잡, 데이터 저장 시 insert sql 2번 호출

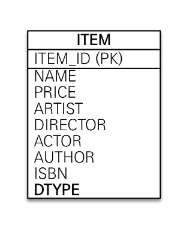


단일 테이블 전략 : 이름 그대로 테이블을 하나만 사용한다. 그리고 구분 컬럼(DTYPE)으로 어떤 자식 데이터가 저장되었는지 구분한다. 조회할 때 조인을 사용하지 않으므로 일반적으로 가장 빠르다. 이 전략을 사용할 때 주의점은 자식 엔티티가 매핑한 컬럼은 모두 null을 허용해야 한다.

- 구분 컬럼을 꼭 사용해야 하고 지정하지 않으면 기본으로 엔티티 이름을 사용한다.

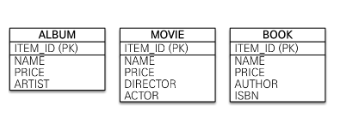
- 장점 : 조인이 필요 없으므로 일반적으로 조회 성능이 빠름, 조회 쿼리가 단순함

- 단점 : 자식 엔티티가 매핑한 컬럼은 모두 null 허용, 단일 테이블에 모든 것을 저장하므로 테이블이 커질 수 있다. 상황에 따라 조회 성능이 오히려 느려질 수 있다.



구현 클래스마다 테이블 전략 : 구현 클래스마다 테이블 전략은 자식 엔티티마다 테이블을 만든다. 그리고 자식 테이블 각각에 필요한 컬럼이 모두 있다. **일반적으로 추천하지 않는 전략이다.**

**구분 컬럼을 사용하지 않음.**

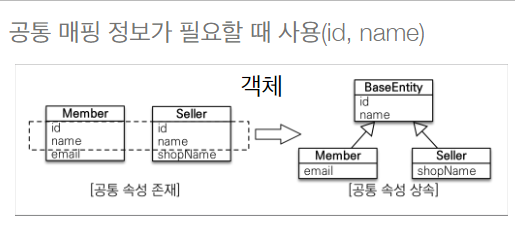


- 장점 : 서브 타입을 명확하게 구분해서 처리할 때 효과적, not null 제약 조건 사용 가능

- 단점 : 여러 자식 테이블을 함께 조회할 때 성능이 느림(union sql 필요) , 자식 테이블을 통합해서 쿼리하기 어려움

**-@MappedSuperclass**

부모 클래스는 테이블과 매핑하지 않고 부모 클래스를 상속 받는 자식 클래스에게 매핑 정보만 제공하고 싶으면 @MappedSuperclass를 사용하면 된다. 실제 테이블과는 매핑되지 않는다. 이것은 단순한 매핑 정보를 상속할 목적으로만 사용된다.



- 조회, 검색 불가 (em.find(BaseEntity) 불가)

- 추상 클래스 권장

- 공통으로 적용하는 정보를 모을 때 사용 (등록일, 수정일, 등록자, 수정자 등)