고급 매핑

상속 관계 매핑

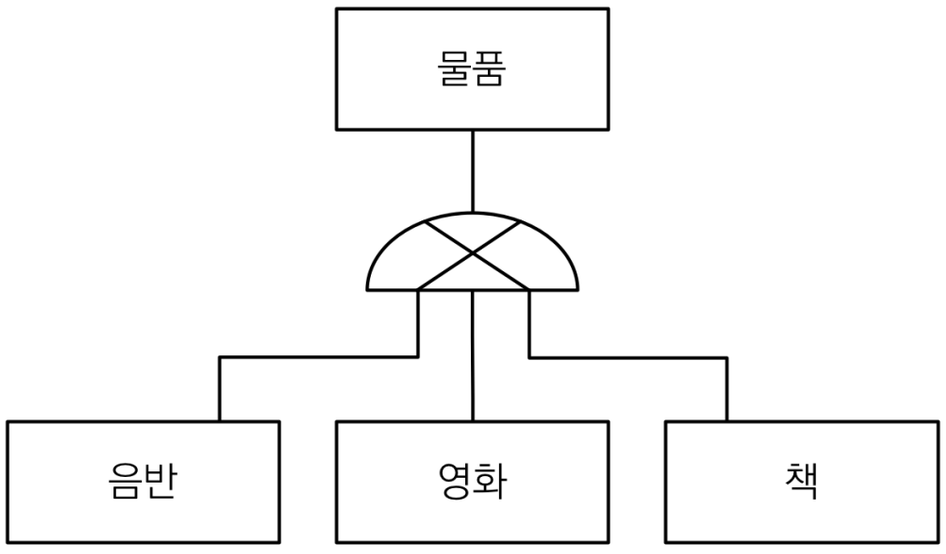
목차

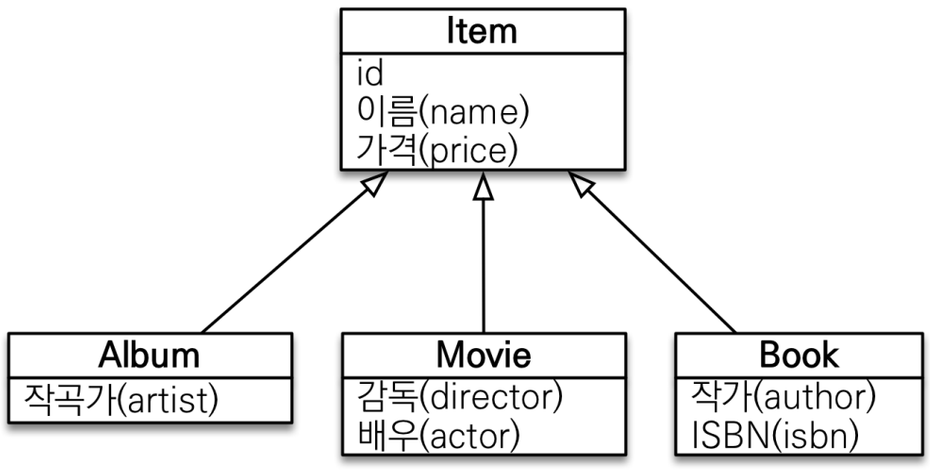
- 상속관계 매핑
- @MappedSuperclass
- 실전 예제 - 4. 상속관계 매핑

상속관계 매핑

상속관계 매핑

- 관계형 데이터베이스는 상속 관계X
- 슈퍼타입 서브타입 관계라는 모델링 기법이 객체 상속과 유사
- 상속관계 매핑: 객체의 상속과 구조와 DB의 슈퍼타입 서브타입 관계를 매핑





상속관계 매핑

- 슈퍼타입 서브타입 논리 모델을 실제 물리 모델로 구현하는 방법
  - 각각 테이블로 변환 -> 조인 전략
  - 통합 테이블로 변환 -> 단일 테이블 전략
  - 서브타입 테이블로 변환 -> 구현 클래스마다 테이블 전략

주요 어노테이션

- @Inheritance(strategy=InheritanceType.XXX)
  - **JOINED**: 조인 전략
  - **SINGLE_TABLE**: 단일 테이블 전략 - 기본 전략
  - **TABLE_PER_CLASS**: 구현 클래스마다 테이블 전략
- @DiscriminatorColumn(name="DTYPE")
- @DiscriminatorValue("XXX")

```java
@Entity
@DiscriminatorValue("A")
public class Album extends Item {
    private String artist;
}
```

기본값 = entity name

설정가능

```java
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn *
public class Item {
    @Id @GeneratedValue
    private Long id;
}
```

abstract

```
Hibernate:

    create table Item (
        DTYPE varchar(31) not null, *
        id bigint not null,
        name varchar(255),
        price integer not null,
        primary key (id)
    )
```

SELECT * FROM ITEM ;

DTYPE	ID	NAME	PRICE
Movie	1	바람과함께사라지다	10000

(1 row, 1 ms)

SELECT * FROM album ;

ARTIST	ID

(행 없음, 0 ms)

SELECT * FROM book ;

AUTHOR	ISBN	ID

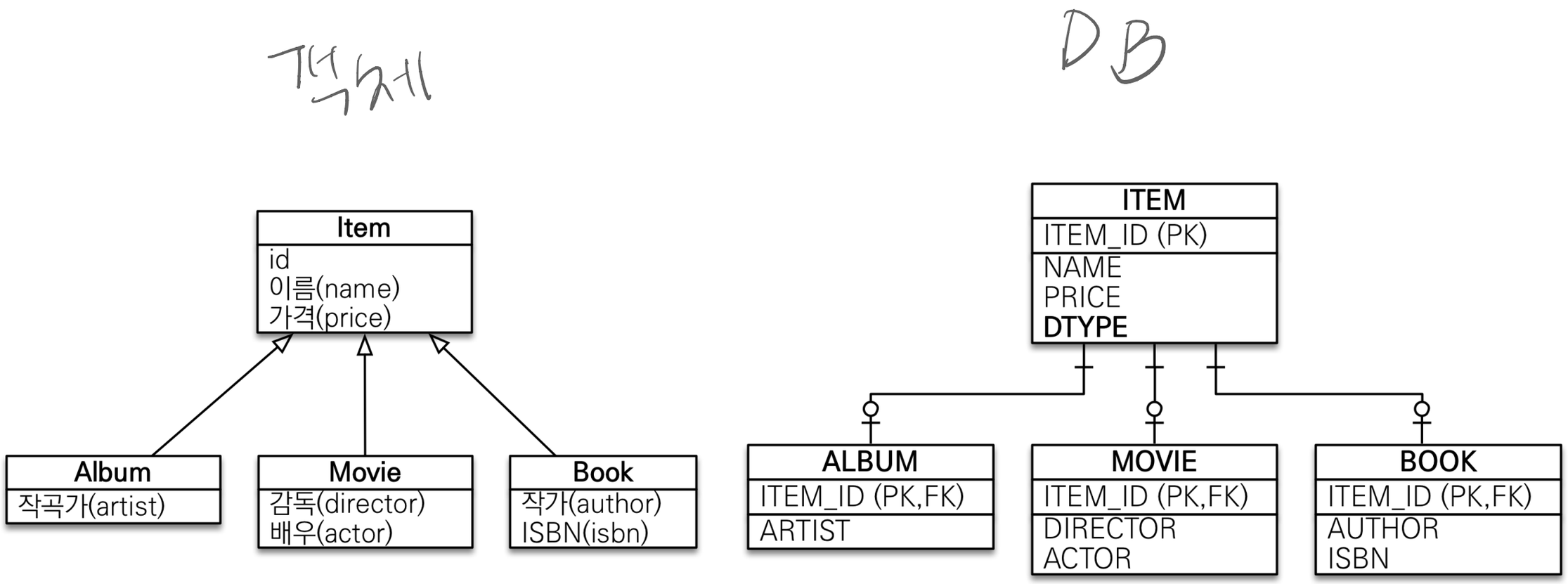
(행 없음, 0 ms)

SELECT * FROM movie ;

ACTOR	DIRECTOR	ID
bbbb	aaaa	1

(1 row, 0 ms)

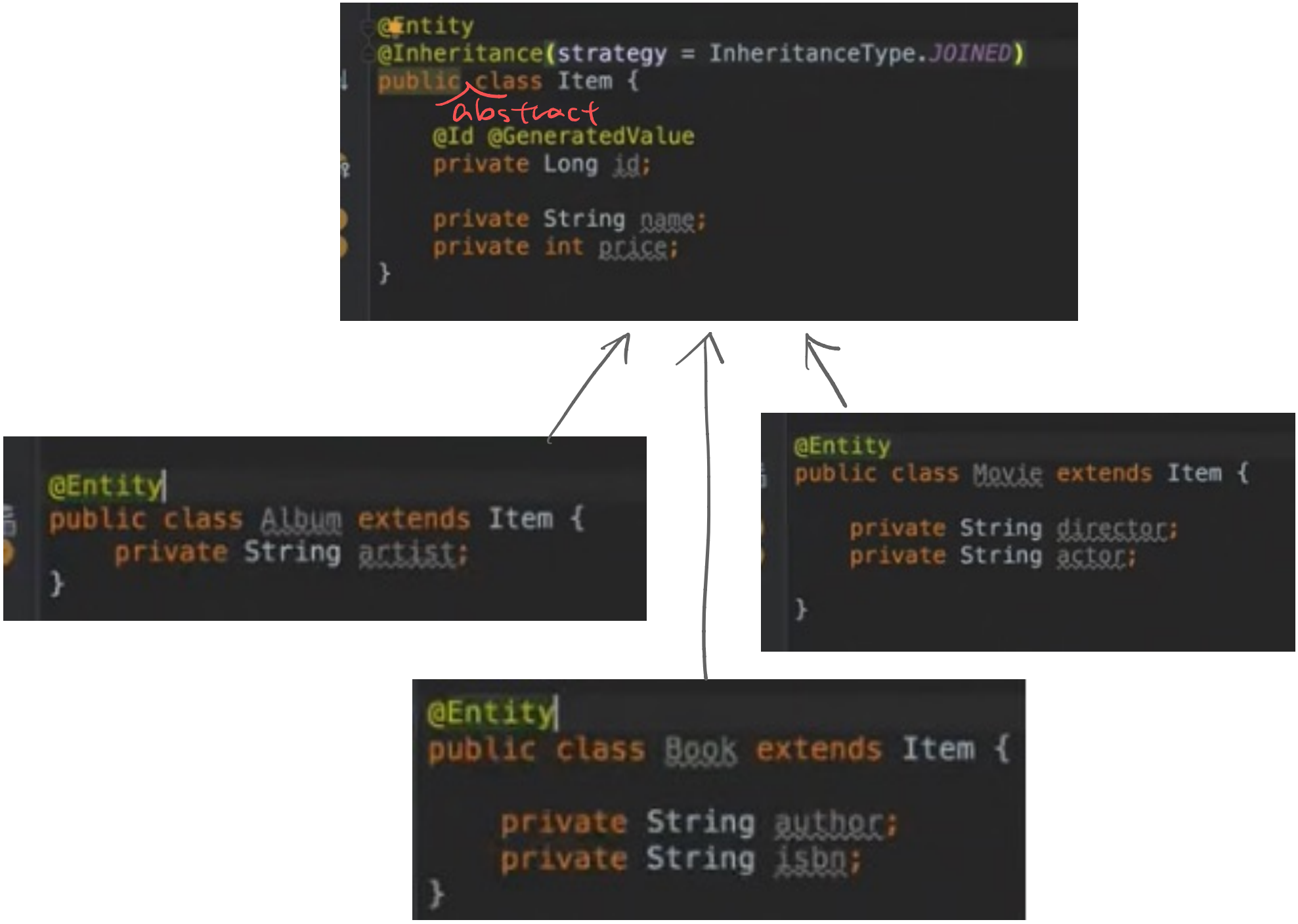
조인 전략 - 가장 정규화된 방식



앨범을 하나 추가하면 ITEM 테이블, ALBUM 테이블에 두번 insert

조회시는 앨범 테이블의 ITEM_ID 를 가지고 ITEM 과 조인해서 조회

앨범인지, 영화인지, 책인지는 DTYPE 이라는 컬럼으로 구분


@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Item {
abstract
@Id @GeneratedValue
private Long id;
private String name;
private int price;
}
@Entity
public class Album extends Item {
private String artist;
}
@Entity
public class Movie extends Item {
private String director;
private String actor;
}
@Entity
public class Book extends Item {
private String author;
private String isbn;
}

삽입

```
Movie movie = new Movie();
movie.setDirector("aaaa");
movie.setActor("bbbb");
movie.setName("바람과함께사라지다");
movie.setPrice(10000);

em.persist(movie);
```

SELECT * FROM ITEM ;

ID	NAME	PRICE
1	바람과함께사라지다	10000

(1 row, 0 ms)

SELECT * FROM album ;

ARTIST	ID

(행 없음, 1 ms)

SELECT * FROM book ;

AUTHOR	ISBN	ID

(행 없음, 0 ms)

SELECT * FROM movie ;

ACTOR	DIRECTOR	ID
bbbb	aaaa	1

(1 row, 1 ms)

같은 값

(PK, FK)

조회

```
Movie findMove = em.find(Movie.class, movie.getId());
System.out.println("findMove = " + findMove);

tx.commit();
```

```
        values
            (?, ?, ?)
Hibernate:
    select
        movie0_.id as id1_2_0_,
        movie0_1_.name as name2_2_0_,
        movie0_1_.price as price3_2_0_,
        movie0_.actor as actor1_6_0_,
        movie0_.director as director2_6_0_
    from
        Movie movie0_
    inner join
        Item movie0_1_
            on movie0_.id=movie0_1_.id
    where
        movie0_.id=?
findMove = hellojpa.Movie@534ca02b
Jun 06, 2019 11:58:28 PM org.hibernate.engine.jdbc.connections.internal.DriverManage
```

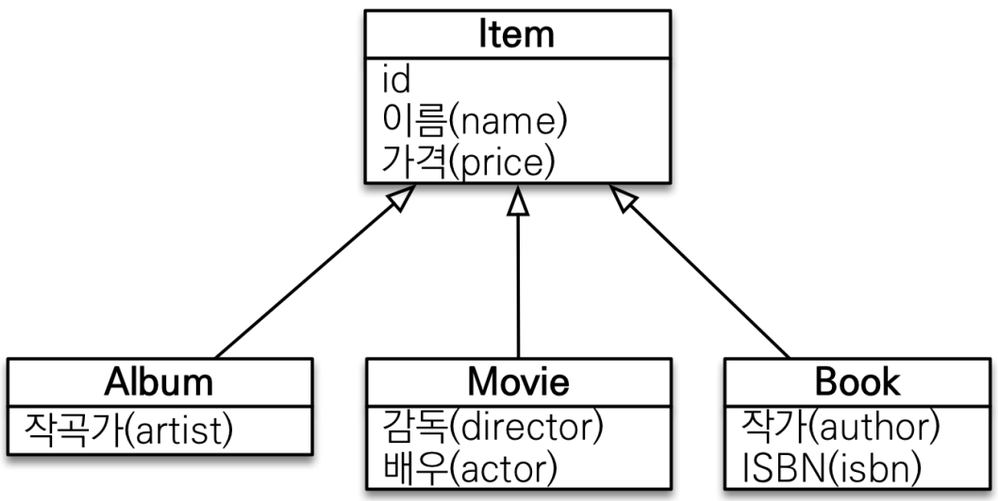
조인해서 조회해온다.

조인 전략 (정석)

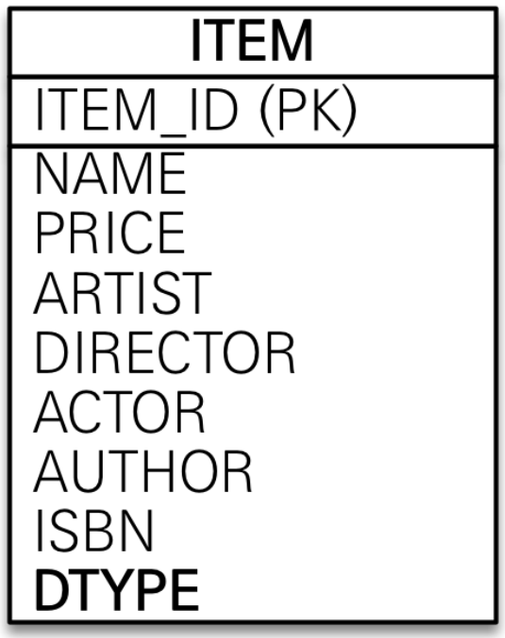
- 장점
  - 테이블 정규화
  - 외래 키 참조 무결성 제약조건 활용가능 - Item_id 값
  - 저장공간 효율화
- 단점
  - 조회시 조인을 많이 사용, 성능 저하
  - 조회 쿼리가 복잡함
  - 데이터 저장시 INSERT SQL 2번 호출

단일 테이블 전략

객체



DB



```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn
public class Item {
```

abstract

SELECT * FROM ITEM ;

DTYPE	ID	NAME	PRICE	ARTIST	AUTHOR	ISBN	ACTOR	DIRECTOR
M	1	바람과함께사라지다	10000	*null*	*null*	*null*	bbbb	aaaa

(1 row, 2 ms)

단일 테이블 전략

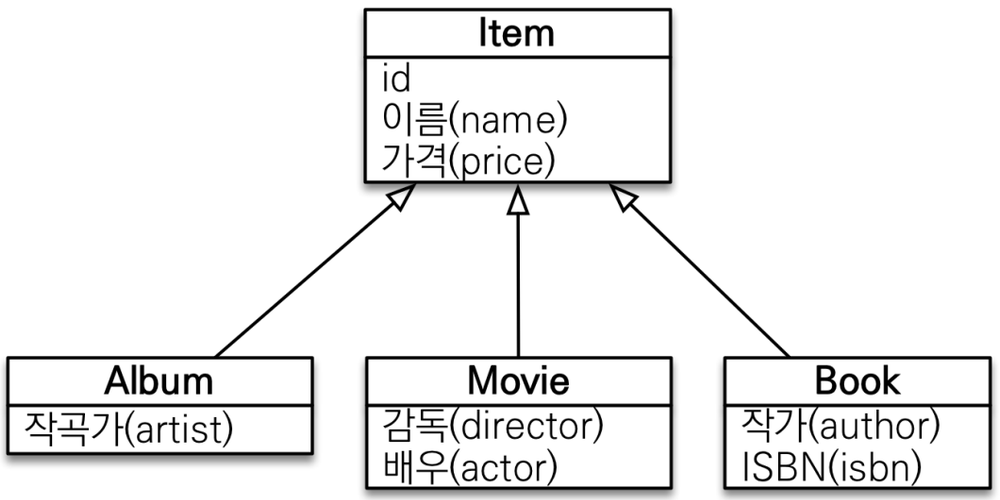
- 장점
  - 조인이 필요 없으므로 일반적으로 조회 성능이 빠름
  - 조회 쿼리가 단순함
- 단점

Movie 를 삽입했다면 Book 의 컬럼들은 모두 null 이 들어가야 함으로

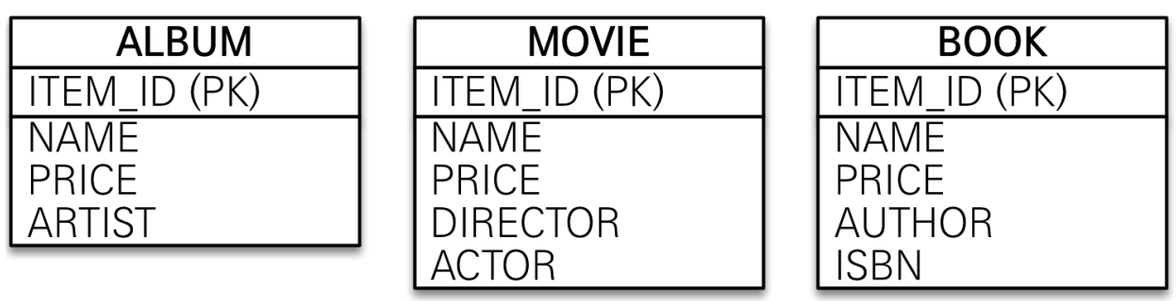
  - 자식 엔티티가 매핑한 컬럼은 모두 null 허용
  - 단일 테이블에 모든 것을 저장하므로 테이블이 커질 수 있다. 상황에 따라서 조회 성능이 오히려 느려질 수 있다.

구현 클래스마다 테이블 전략

객체



DB



구현 클래스마다 테이블 전략

- **이 전략은 데이터베이스 설계자와 ORM 전문가 둘 다 추천X**
- 장점
  - 서브 타입을 명확하게 구분해서 처리할 때 효과적
  - not null 제약조건 사용 가능
- 단점

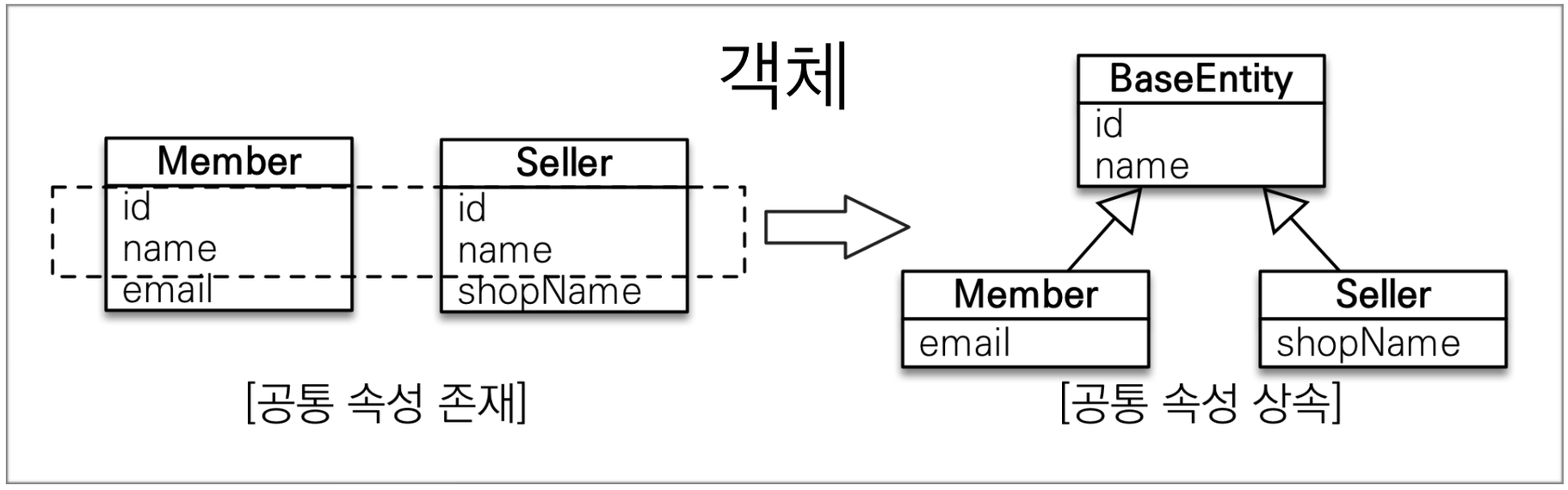
Ex) item id 가 5라는 것만 알고 조회하려고 할 때 union 을 써서 조회해옴 - 느림!

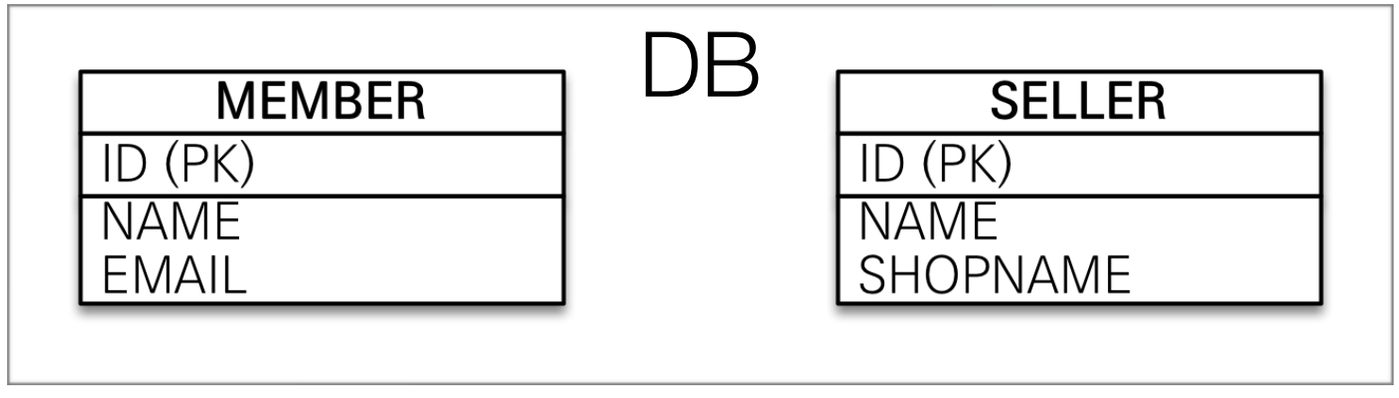
  - 여러 자식 테이블을 함께 조회할 때 성능이 느림(UNION SQL 필요)
  - 자식 테이블을 통합해서 쿼리하기 어려움

@MappedSuperclass 상속과는 크게 상관 없음

@MappedSuperclass 상속 개념보다는 같은 컬럼이 계속 반복되어 귀찮을 때

- 공통 매핑 정보가 필요할 때 사용(id, name)





@MappedSuperclass

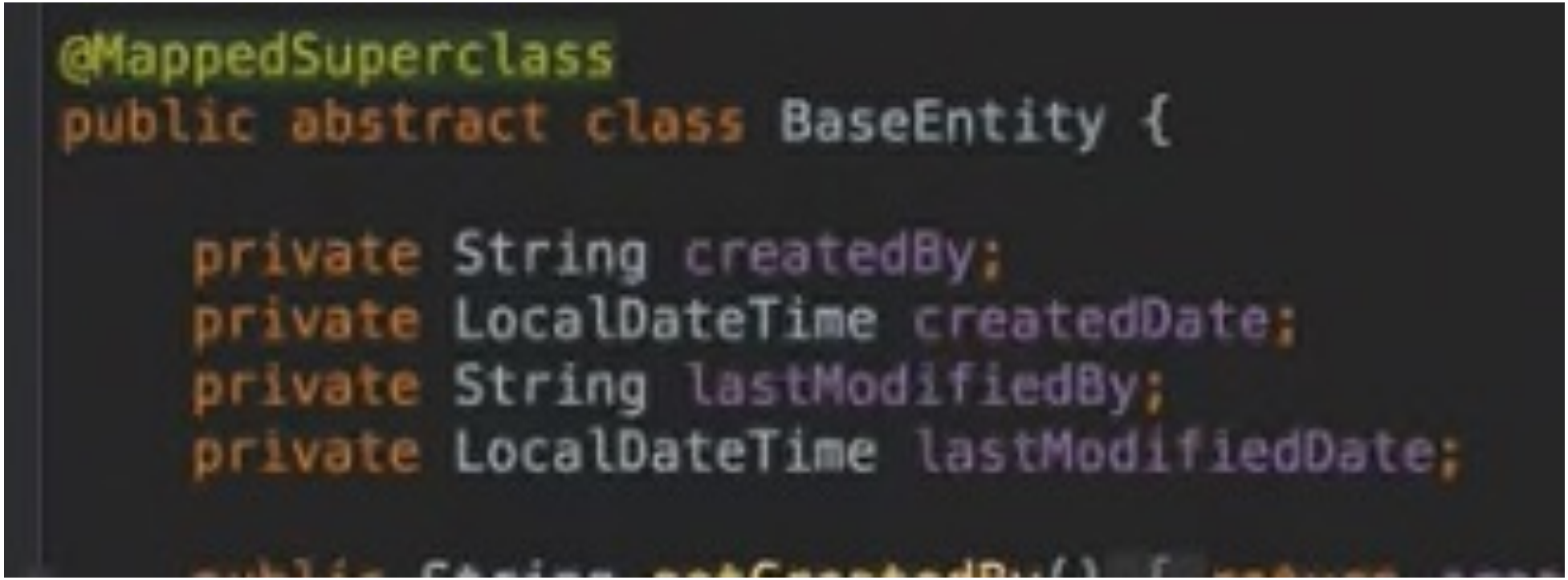
- 상속관계 매핑X
- 엔티티X, 테이블과 매핑X
- 부모 클래스를 상속 받는 **자식 클래스에 매핑 정보만 제공**
- 조회, 검색 불가(**em.find(BaseEntity) 불가**)
- 직접 생성해서 사용할 일이 없으므로 **추상 클래스 권장**

@MappedSuperclass

- 테이블과 관계 없고, 단순히 엔티티가 공통으로 사용하는 매핑 정보를 모으는 역할

- 주로 등록일, 수정일, 등록자, 수정자 같은 전체 엔티티에서 공통으로 적용하는 정보를 모을 때 사용

- 참고: @Entity 클래스는 엔티티나 @MappedSuperclass로 지정한 클래스만 상속 가능

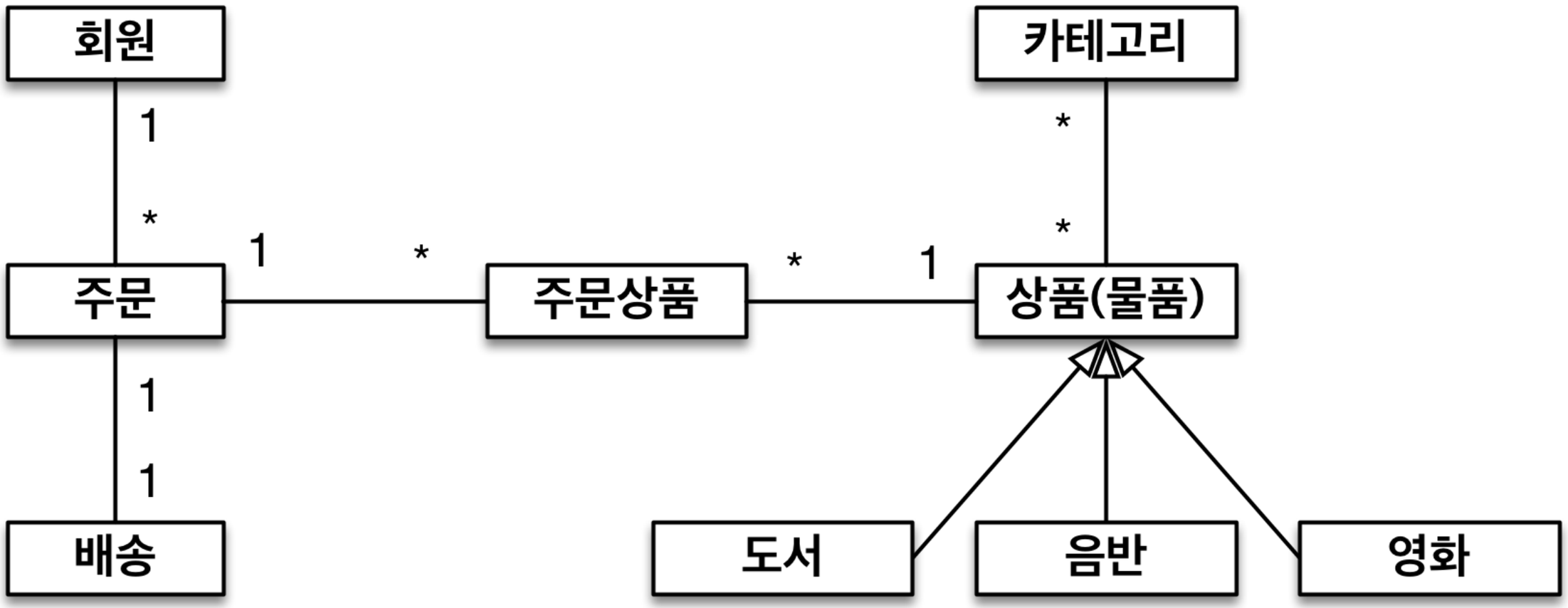

@MappedSuperclass
public abstract class BaseEntity {
private String createdBy;
private LocalDateTime createdDate;
private String lastModifiedBy;
private LocalDateTime lastModifiedDate;

실전 예제 - 4. 상속관계 매핑

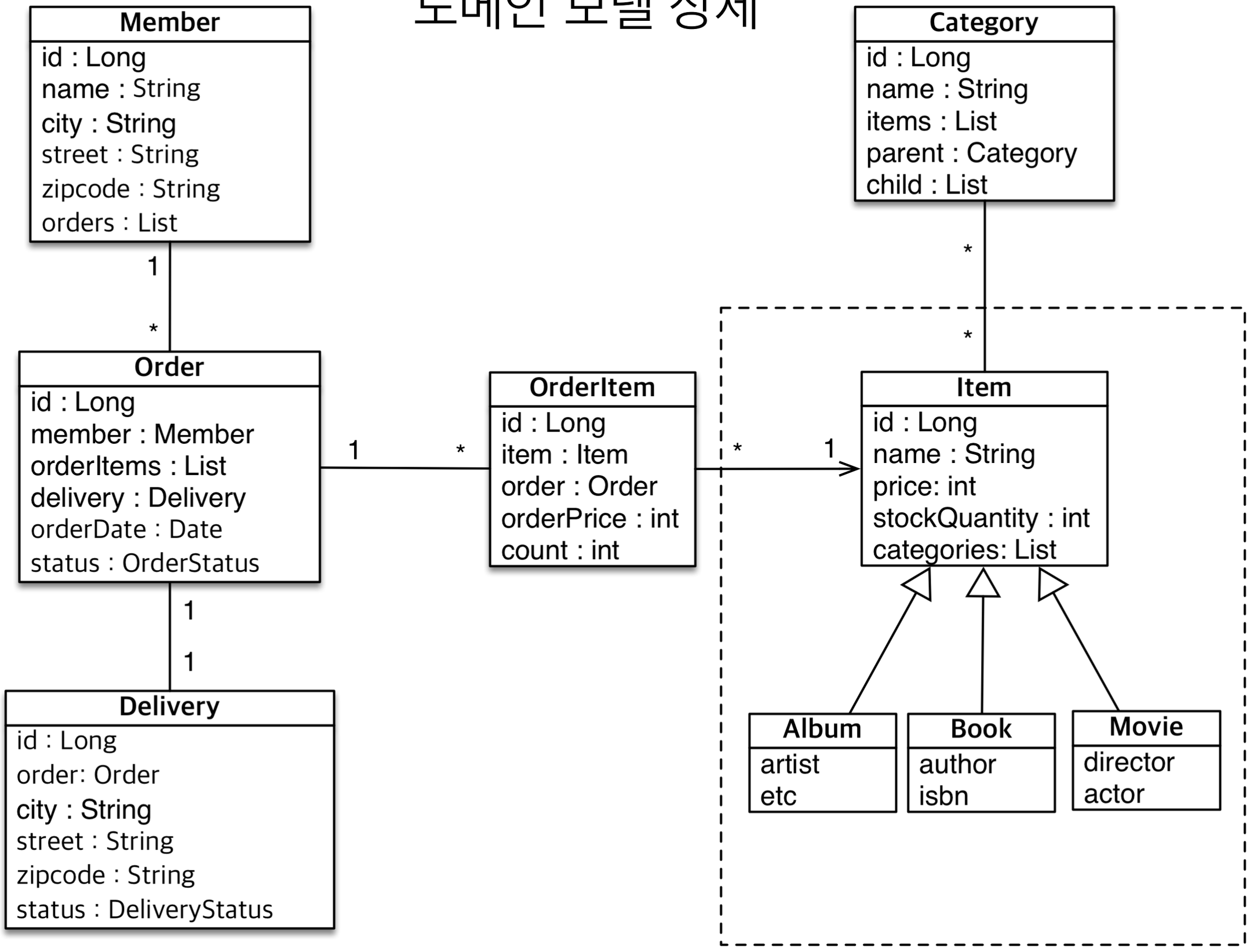
요구사항 추가

- 상품의 종류는 음반, 도서, 영화가 있고 이후 더 확장될 수 있다.
- 모든 데이터는 등록일과 수정일이 필수다.

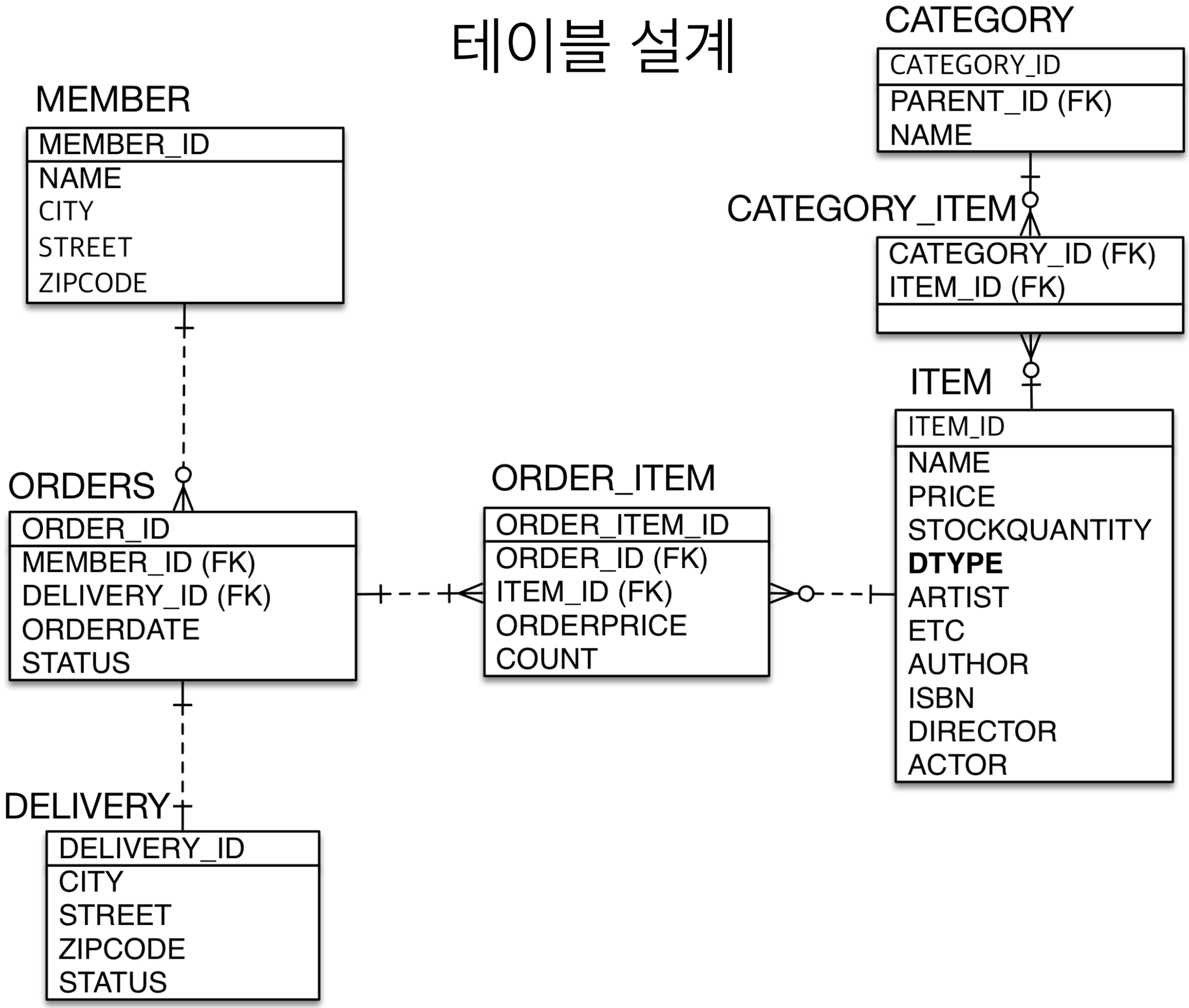
도메인 모델



도메인 모델 상세


Member
id : Long
name : String
city : String
street : String
zipcode : String
orders : List
1
*
Order
id : Long
member : Member
orderItems : List
delivery : Delivery
orderDate : Date
status : OrderStatus
1
1
Delivery
id : Long
order: Order
city : String
street : String
zipcode : String
status : DeliveryStatus
1
*
OrderItem
id : Long
item : Item
order : Order
orderPrice : int
count : int
*
1
Category
id : Long
name : String
items : List
parent : Category
child : List
*
*
Item
id : Long
name : String
price: int
stockQuantity : int
categories: List
Album
artist
etc
Book
author
isbn
Movie
director
actor

테이블 설계


MEMBER
MEMBER_ID
NAME
CITY
STREET
ZIPCODE
CATEGORY
CATEGORY_ID
PARENT_ID (FK)
NAME
CATEGORY_ITEM
CATEGORY_ID (FK)
ITEM_ID (FK)
ITEM
ITEM_ID
NAME
PRICE
STOCKQUANTITY
DTYPE
ARTIST
ETC
AUTHOR
ISBN
DIRECTOR
ACTOR
ORDERS
ORDER_ID
MEMBER_ID (FK)
DELIVERY_ID (FK)
ORDERDATE
STATUS
ORDER_ITEM
ORDER_ITEM_ID
ORDER_ID (FK)
ITEM_ID (FK)
ORDERPRICE
COUNT
DELIVERY
DELIVERY_ID
CITY
STREET
ZIPCODE
STATUS