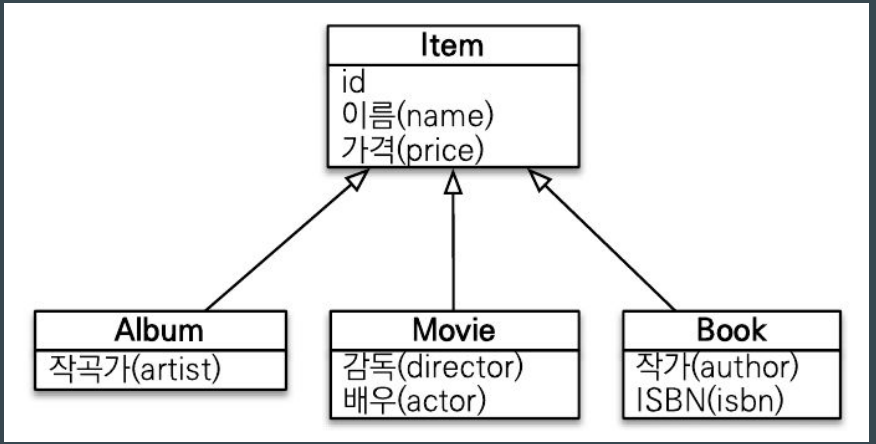
07. 고급 매핑

...

Blur

상속 관계 매핑

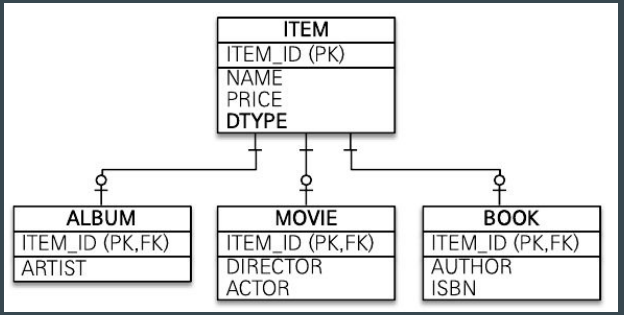
1. Join 전략
2. 단일 테이블 전략
3. 구현 클래스마다 테이블 전략



상속 관계 매핑(Join 전략)

각각의 Table로 나누어서 DTYPE을 이용하여 상속관계를 구분

- 테이블 정규화
- FK 참조 무결성 제약조건 달성
- 저장공간 효율적 달성
- Join이 많아 조회성능 저하
- 조회 쿼리 복잡
- Insert SQL 두번 실행




```java
@Inheritance(strategy = InheritanceType.JOINED)
@DiscriminatorColumn(name = "DTYPE")
public class JoinStrategyItem {
```

```java
@Entity
@DiscriminatorValue("B")
@PrimaryKeyJoinColumn(name = "BOOK_ID")
public class JoinStrategyBook extends JoinStrategyItem {
```

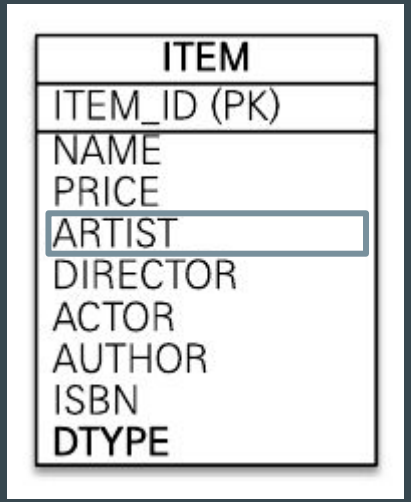
상속 관계 매핑(단일 테이블 전략)

하나의 Table을 사용

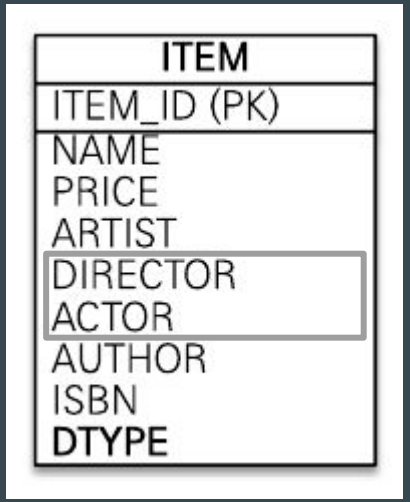
DTYPE로 상속관계를 구분

각 Column에 Null을 허용하여 DTYPE으로 나머지 Column 사용유무를 결정

- 조회 성능 우수, 조회쿼리 단순
- null허용, 테이블이 커질 수 있음
- 상황에 따라 조회 성능이 느려질 수 있음



DTYPE="A"



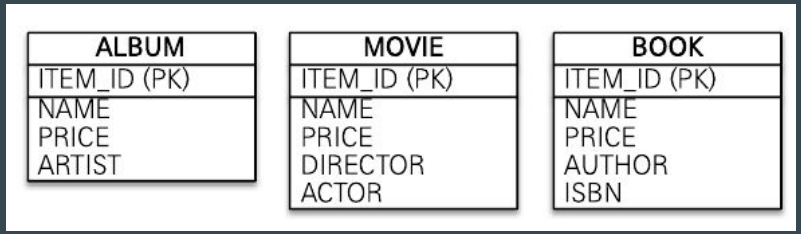
DTYPE="M"

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
```

상속 관계 매핑(구현 클래스마다 테이블 전략)

각 테이블에 모든 ITEM Field 존재

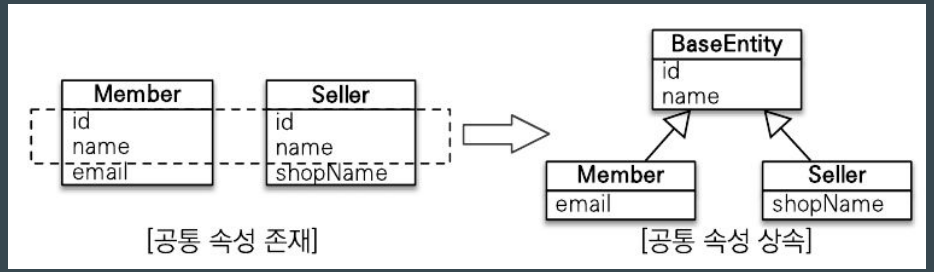
일반적으로 추천하지 않음



상속 관계 매핑(@MappedSuperclass)

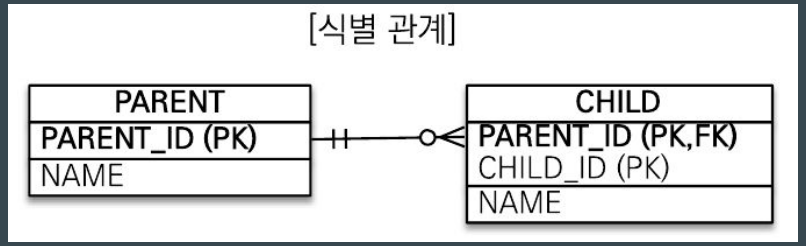
테이블과 매핑하지 않고 매핑 정보만 제공

- 추상클래스와 비슷
- 상속받은 Field Name을 변경하기 위해 AttributeOverride 사용



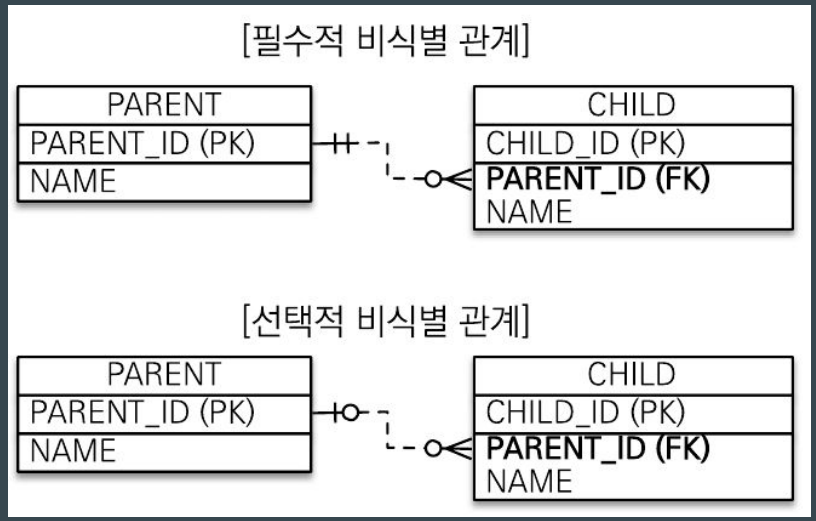

```
@Entity
@AttributeOverrides({
        @AttributeOverride(name = "id", column = @Column(name = "MEMBER_ID")),
        @AttributeOverride(name = "name", column = @Column(name = "MEMBER_NAME"))
})
public class Seller extends BaseEntity{
    private String shopName;
}
```

식별관계 vs 비식별관계



부모 테이블의 PK를 받아서

자식 테이블의 PK+FK로 사용하는 관계



필수적 : Null 허용X

선택적 : Null 허용O

복합 키

- Serializable 인터페이스 구현
- equals, hashCode 구현
- NoargsConstructor
- public class

```java
@AllArgsConstructor
@NoArgsConstructor
public class ParentId implements Serializable {

    private String id1;
    private String id2;

    @Override
    public int hashCode() { return super.hashCode(); }

    @Override
    public boolean equals(Object obj) { return super.equals(obj); }
}
```

식별자 매핑(@IdClass)

- 데이터 베이스 지향적 방법

```java
@Entity
@IdClass(GrandChildId.class)
public class GrandChild {

    @Id
    @ManyToOne
    @JoinColumns({
            @JoinColumn(name = "PARENT_ID"),
            @JoinColumn(name = "CHILD_ID")
    })
    private Child child;

    @Id @Column(name = "GRANDCHILD_ID")
    private String id;

    private String name;
```

```java
public class GrandChildId implements Serializable {

    private ChildId childId;
    private String Id;

    @Override
    public int hashCode() { return super.hashCode(); }

    @Override
    public boolean equals(Object obj) { return super.equals(obj); }
}
```

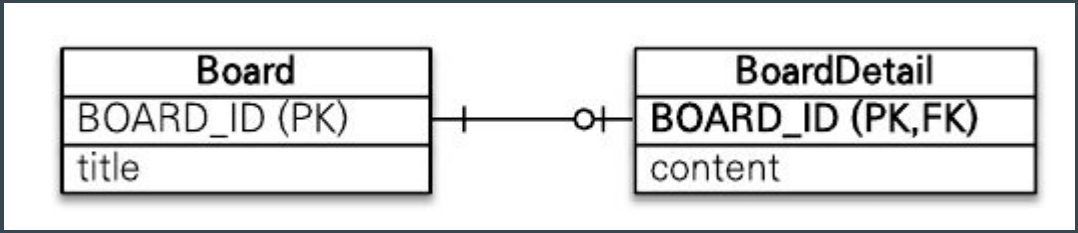
식별자 매핑(@EmbeddedId)

- 객체 지향적 방법

```java
@Entity
public class GrandChild {

    @EmbeddedId
    private GrandChildId id;

    @MapsId("childId")
    @ManyToOne
    @JoinColumns({
            @JoinColumn(name = "PARENT_ID"),
            @JoinColumn(name = "CHILD_ID")
    })
    private Child child;

    private String name;
}
```

```java
@Embeddable
public class GrandChildId implements Serializable {
    private ChildId childId;

    @Column(name = "GRANDCHILD_ID")
    private String id;

    @Override
    public int hashCode() { return super.hashCode(); }

    @Override
    public boolean equals(Object obj) { return super.equals(obj); }
}
```

1:1 식별 관계

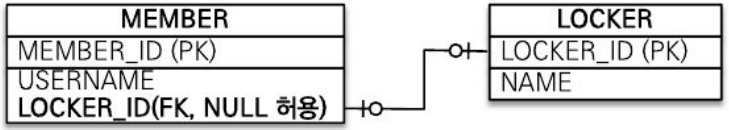



```java
@Entity
public class Board {
    @Id @GeneratedValue
    @Column(name = "BOARD_ID")
    private Long id;

    private String title;

    @OneToOne(mappedBy = "board")
    private BoardDetail boardDetail;
}
```

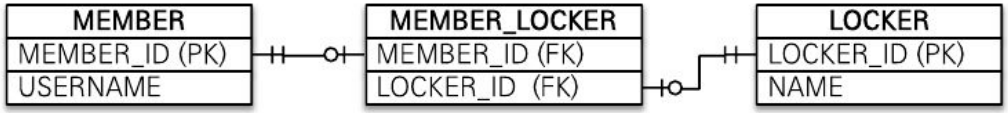
```java
@Entity
public class BoardDetail {
    @Id
    private Long boardId;
    @MapsId
    @OneToOne
    @JoinColumn(name = "BOARD_ID")
    private Board board;

    private String contents;
}
```

Join Table

[조인 컬럼 사용]



[조인 테이블 사용]




```java
@OneToMany
@JoinColumn(name = "PARENT_CHILD")
private List<Child> childList = new ArrayList<Child>();
```

- 기본적으로는 JoinColumn을 사용,
  필요한 경우에만 JoinTable 사용

```java
@OneToMany
@JoinTable(name = "PARENT_CHILD",
        joinColumns = @JoinColumn(name = "PARENT_ID"),
        inverseJoinColumns = @JoinColumn(name = "CHILD_ID")
)
private List<Child> childList = new ArrayList<Child>();
```

Thank you