

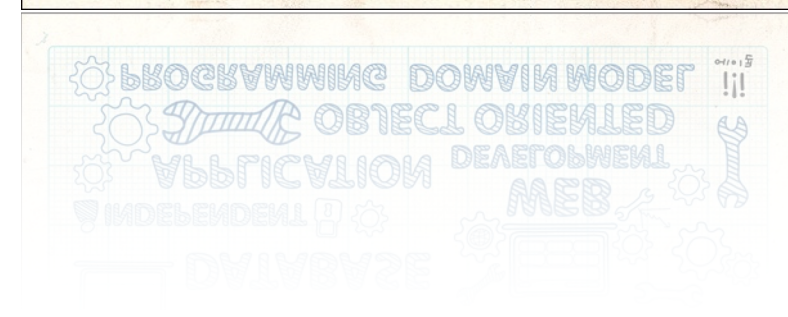
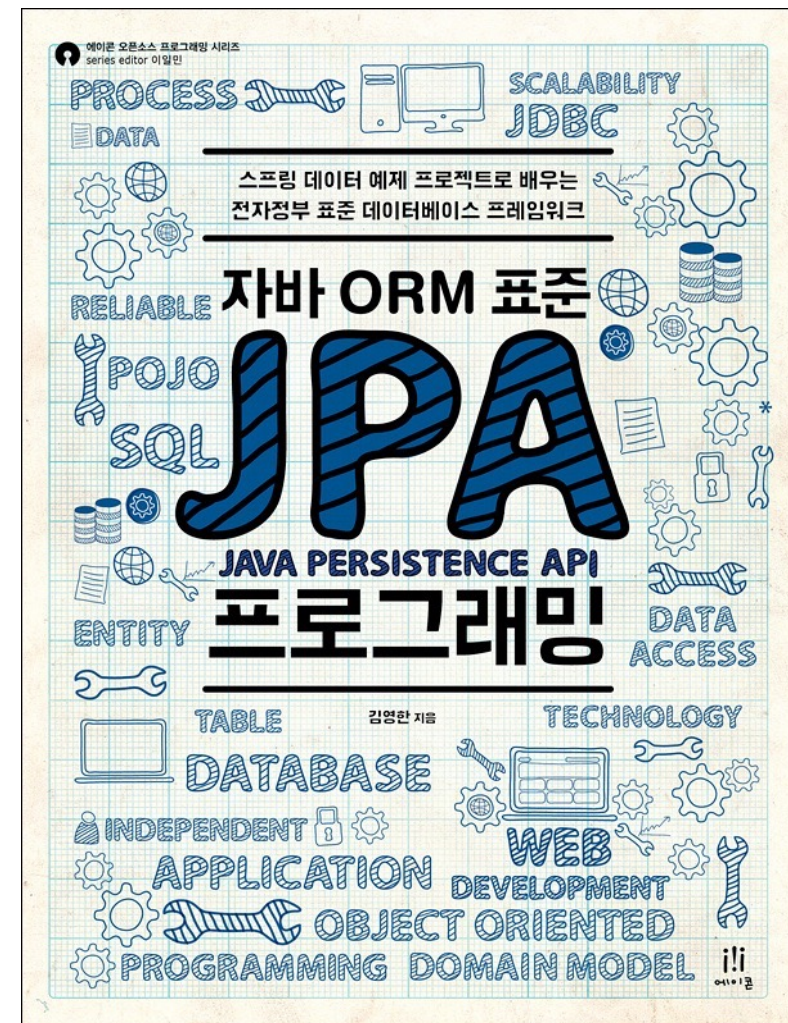
JPA와 모던 데이터 저장 기술

Java Persistence API

김영한

SI, J2EE 강사, DAUM, SK 플래닛

저서: 자바 ORM 표준 **JPA** 프로그래밍



목차

1. JPA 소개
2. 객체 관계 매핑
3. JPA 내부 구조
4. 객체 지향 쿼리
5. 스프링과 JPA

JPA 소개 목차

1. SQL 중심적인 개발의 문제점
2. JPA 소개



애플리케이션 객체 지향 언어

자바



데이터베이스 세계의 헤게모니 관계형 데이터베이스

MySQL, Oracle, PostgreSQL ...

지금 시대는 객체를 관계형 데이터베이스에 관리

SQL! **SQL!!** **SQL!!!**

SQL 중심적인 개발의 문제점



무한 반복, 지루한 코드

CURD ... INSERT INTO ...

객체 CURD

```
public class Member {
```

```
    private String memberId;
```

```
    private String name;
```

```
    ...
```

```
}
```

```
INSERT INTO MEMBER(MEMBER_ID, NAME) VALUES
```

```
SELECT MEMBER_ID, NAME FROM MEMBER M
```

```
UPDATE MEMBER SET ...
```

객체 CURD - 필드 추가

```
public class Member {
```

```
    private String memberId;
```

```
    private String name;
```

```
    private String tel;
```

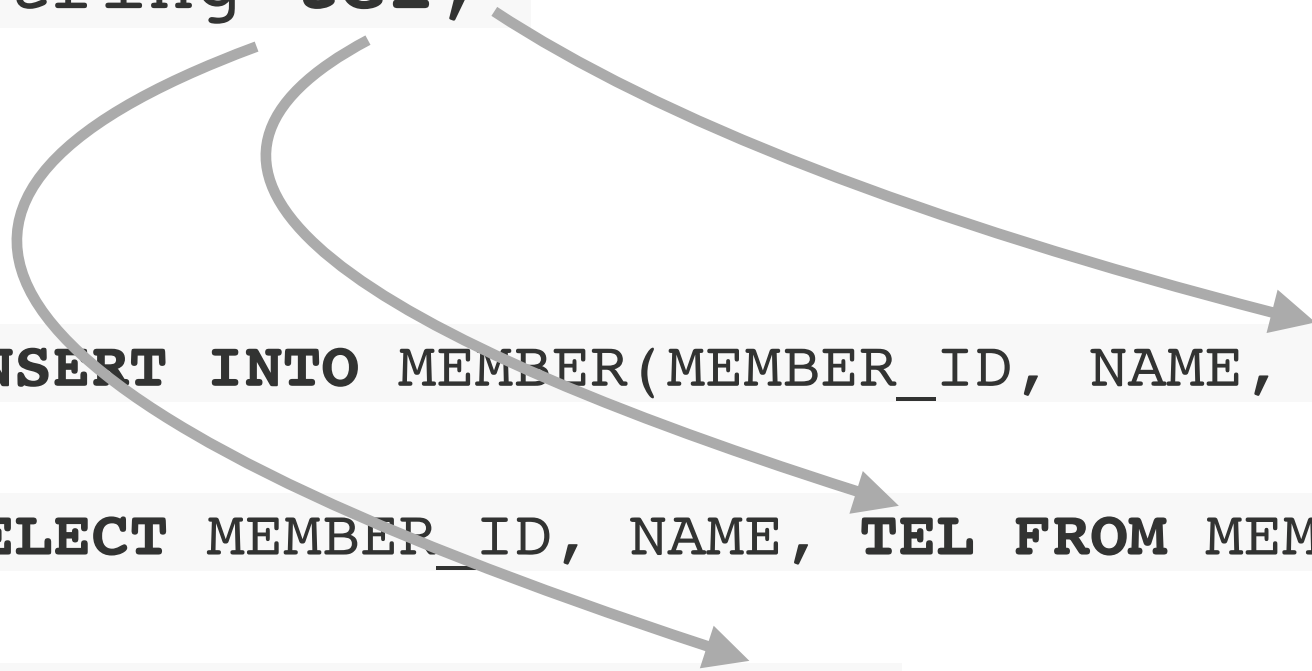
```
    ...
```

```
}
```

INSERT INTO MEMBER(MEMBER_ID, NAME, TEL) VALUES ...

SELECT MEMBER_ID, NAME, TEL FROM MEMBER M

UPDATE MEMBER SET ... TEL=?



SQL을 확인하기 전까진 엔티티를 신뢰하기 어렵다.

```
class MemberService {  
    ...  
    public void process(String id) {  
        Member member = memberDAO.find(id);  
        member.getTeam(); //???  
        member.getOrder().getDelivery(); // ???  
    }  
}
```

계층형 아키텍처, 진정한 의미의 계층 분할이 어렵다.

SQL에 의존적인 개발을 피하기 어렵다.



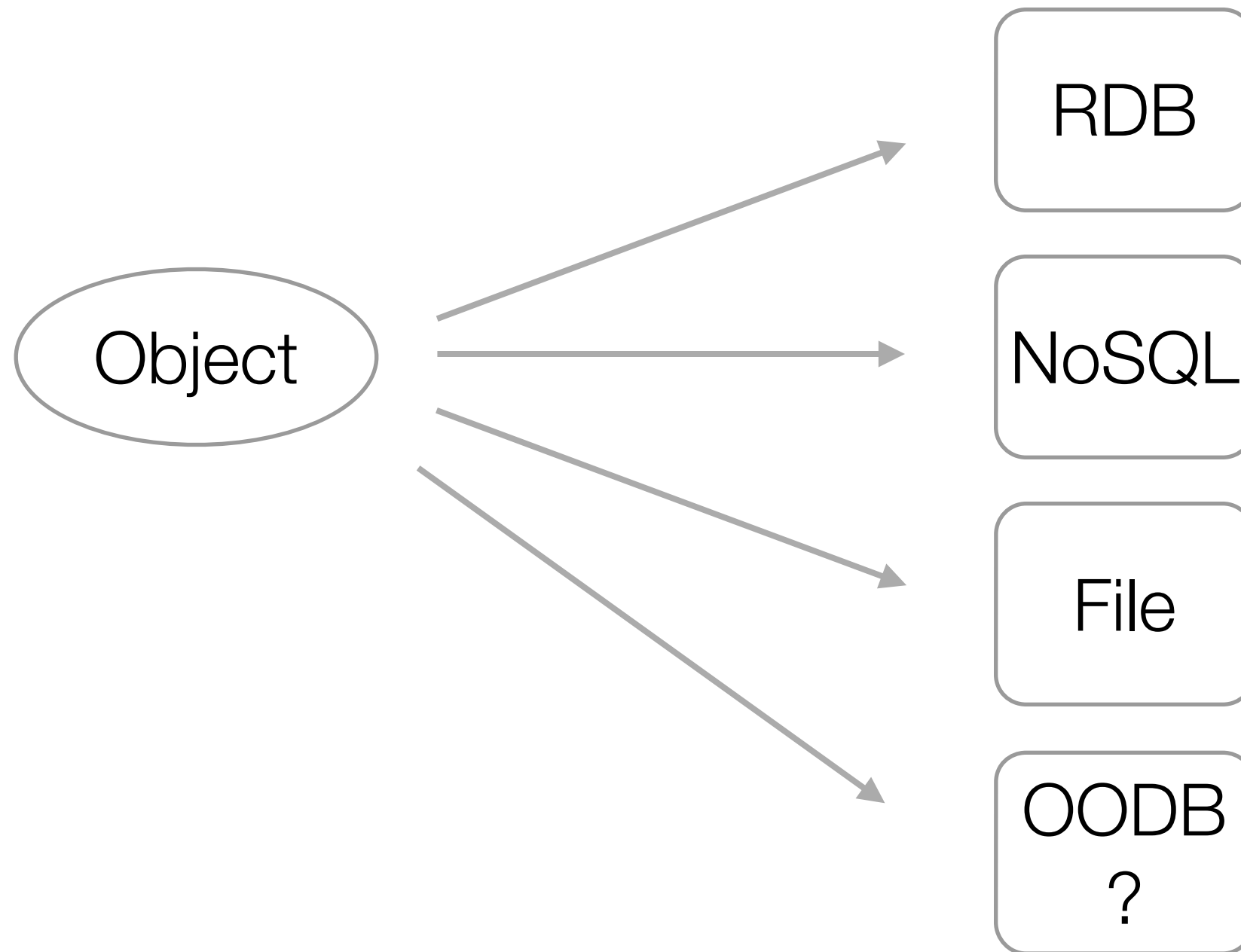
패러다임의 불일치

객체 vs 관계형 데이터베이스

‘객체 지향 프로그래밍은 추상화, 캡슐화, 정보은닉, 상속, 다형성 등 시스템의 복잡성을 제어할 수 있는 다양한 장치들을 제공한다.’

–어느 객체지향 개발자가

객체를 영구 보관하는 다양한 저장소

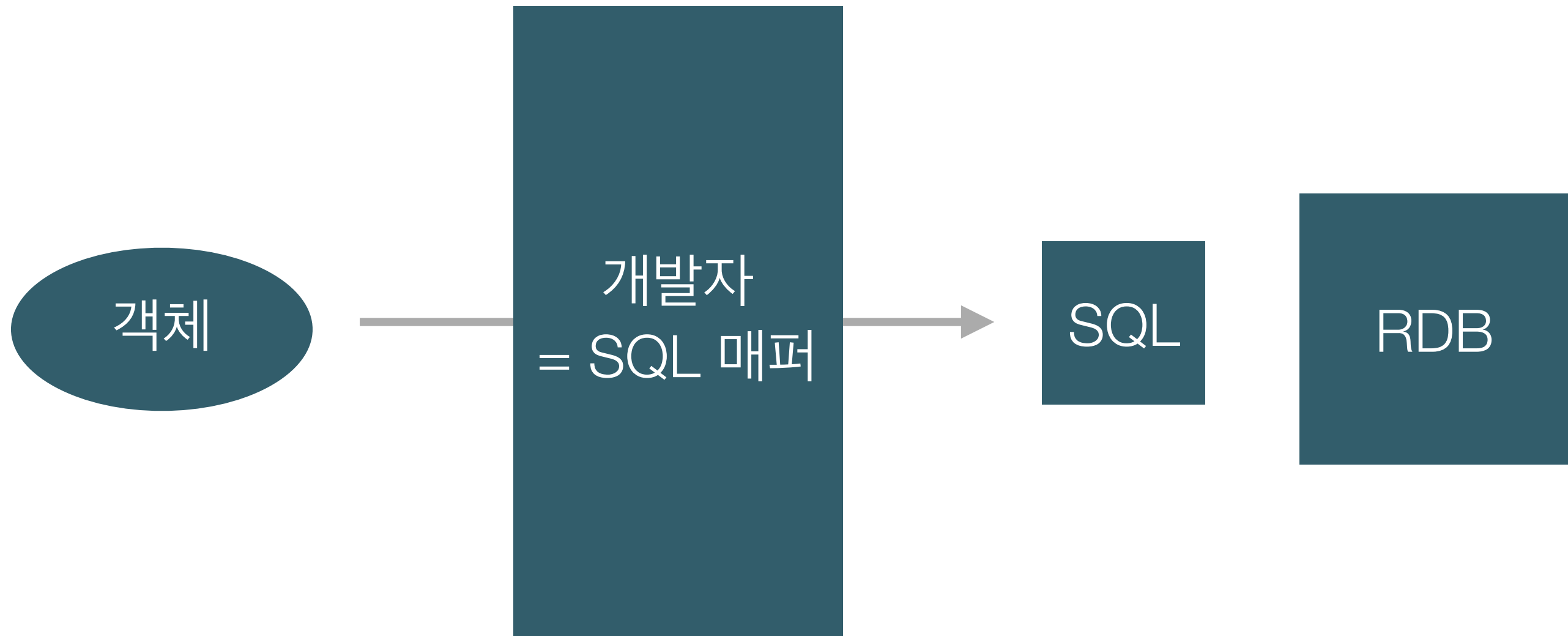


가장 현실적인 대안은 관계형 데이터베이스

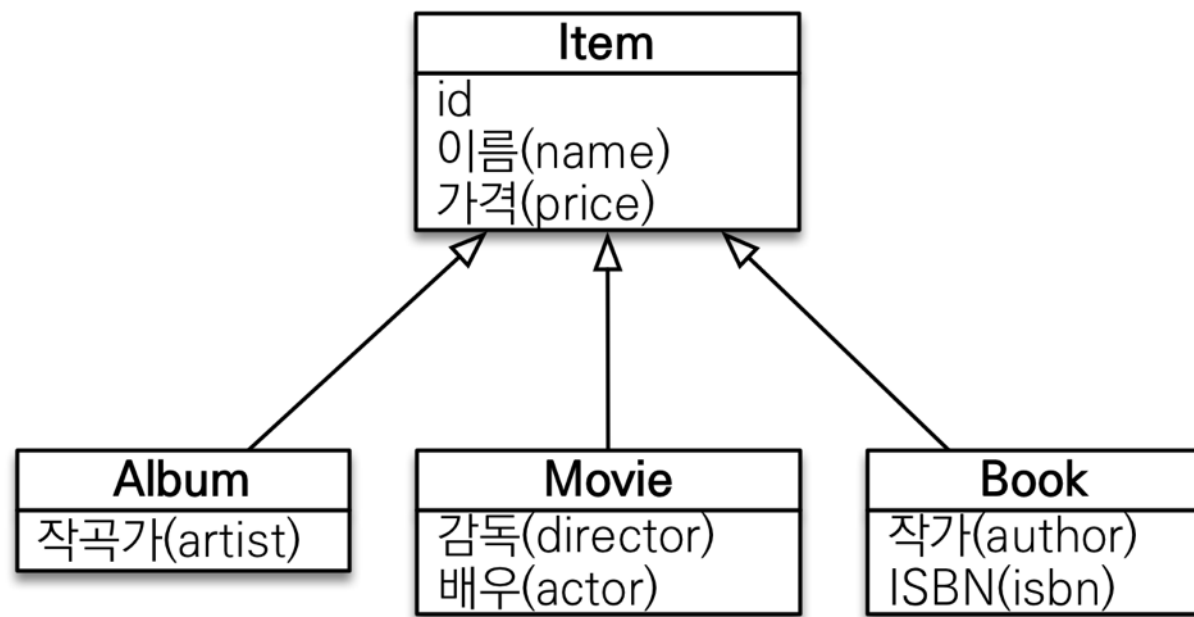
객체와 관계형 데이터베이스의 차이

- 상속
- 연관관계
- 데이터 타입
- 데이터 식별 방법

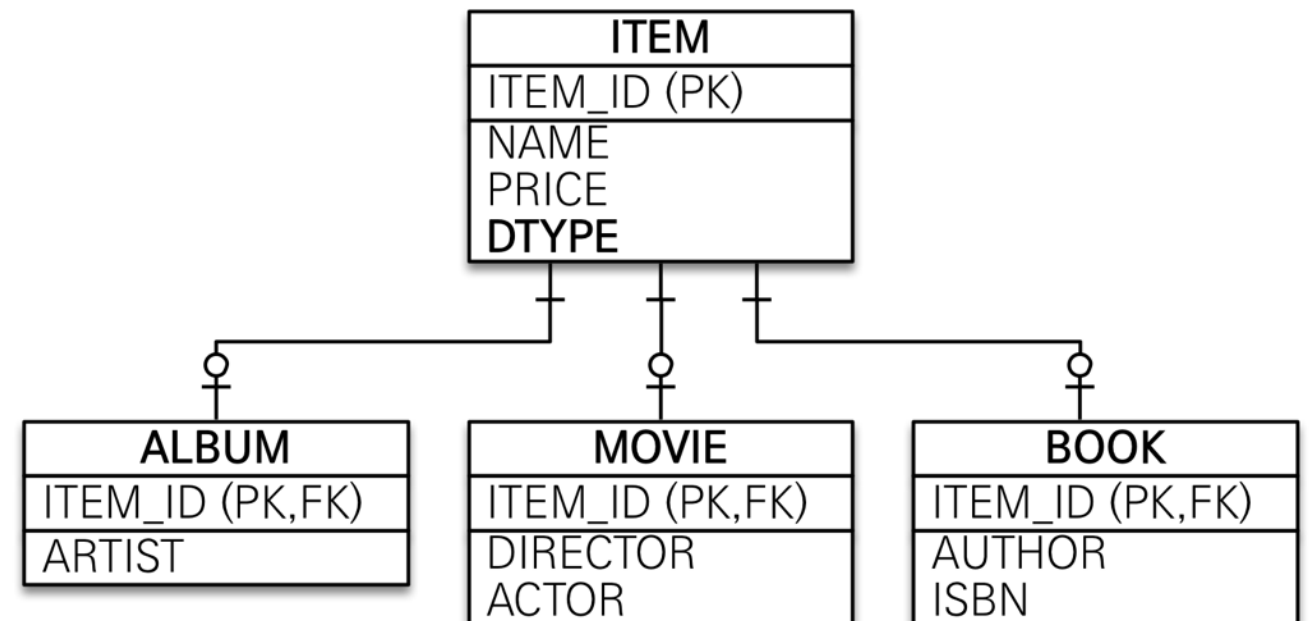
객체를 관계형 데이터베이스에 저장



상속



[객체 상속 관계]



[Table 슈퍼타입 서브타입 관계]

Album 저장

- 객체 분해
- INSERT INTO ITEM ...
- INSERT INTO ALBUM ...

Album 조회

- 각각의 테이블에 따른 조인 SQL 작성...
- 각각의 객체 생성...
- 상상만 해도 복잡
- 더 이상의 설명은 생략한다.
- 그래서 저장할 객체에는 상속 관계 안쓴다.

자바 컬렉션에 저장하면?

```
list.add(album);
```

자바 컬렉션에서 조회하면?

```
Album album = list.get(albumId);
```

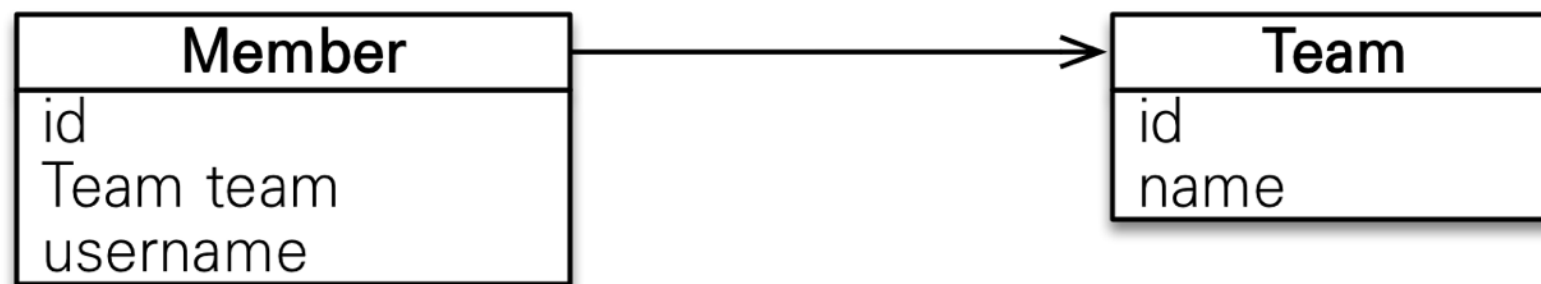
```
Item item = list.get(albumId);
```

부모 타입으로 조회 후 다형성 활용

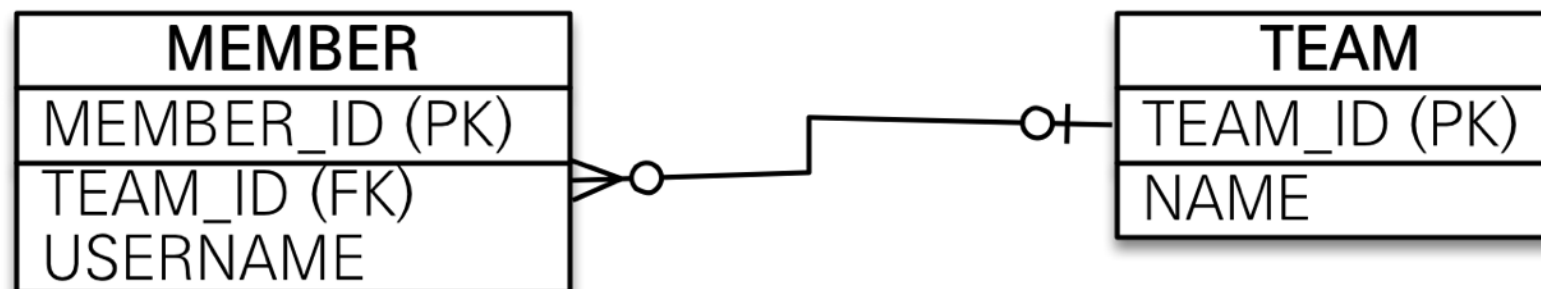
연관관계

- 객체는 참조를 사용: `member.getTeam()`
- 테이블은 외래 키를 사용: `JOIN ON M.TEAM_ID = T.TEAM_ID`

[객체 연관관계]



[테이블 연관관계]



객체를 테이블에 맞추어 모델링

```
class Member {  
    String id;           //MEMBER_ID 컬럼 사용  
    Long teamId;         //TEAM_ID FK 컬럼 사용 /**  
    String username;     //USERNAME 컬럼 사용  
}
```

```
class Team {  
    Long id;             //TEAM_ID PK 사용  
    String name;         //NAME 컬럼 사용  
}
```

테이블에 맞춘 객체 저장

```
class Member {  
    String id;           //MEMBER_ID 컬럼 사용  
    Long teamId;         //TEAM_ID FK 컬럼 사용 /**  
    String username;     //USERNAME 컬럼 사용  
}
```

```
INSERT INTO MEMBER(MEMBER_ID, TEAM_ID, USERNAME) VALUES ...
```



The diagram illustrates the mapping between the class fields and the SQL columns. Three curved arrows originate from the class definition and point to the corresponding columns in the SQL statement: one from 'id' to 'MEMBER_ID', one from 'teamId' to 'TEAM_ID', and one from 'username' to 'USERNAME'.

객체 다운 모델링

```
class Member {  
    String id;           //MEMBER_ID 컬럼 사용  
    Team team;           //참조로 연관관계를 맺는다. /**  
    String username;     //USERNAME 컬럼 사용  
  
    Team getTeam() {  
        return team;  
    }  
}
```

```
class Team {  
    Long id;             //TEAM_ID PK 사용  
    String name;         //NAME 컬럼 사용  
}
```


객체 모델링 저장

```
class Member {  
    String id;           //MEMBER_ID 컬럼 사용  
    Team team;          //참조로 연관관계를 맺는다. /**  
    String username;    //USERNAME 컬럼 사용  
}
```

```
member.getTeam().getId();
```



```
INSERT INTO MEMBER(MEMBER_ID, TEAM_ID, USERNAME) VALUES ...
```

객체 모델링 조회

```
SELECT M.*, T.*  
FROM MEMBER M  
JOIN TEAM T ON M.TEAM_ID = T.TEAM_ID
```

```
public Member find(String memberId) {  
    //SQL 실행  
    ...  
    Member member = new Member();  
    //데이터베이스에서 조회한 회원 관련 정보를 모두 입력  
  
    Team team = new Team();  
    //데이터베이스에서 조회한 팀 관련 정보를 모두 입력  
  
    //회원과 팀 관계 설정  
    member.setTeam(team); /**  
    return member;  
}
```

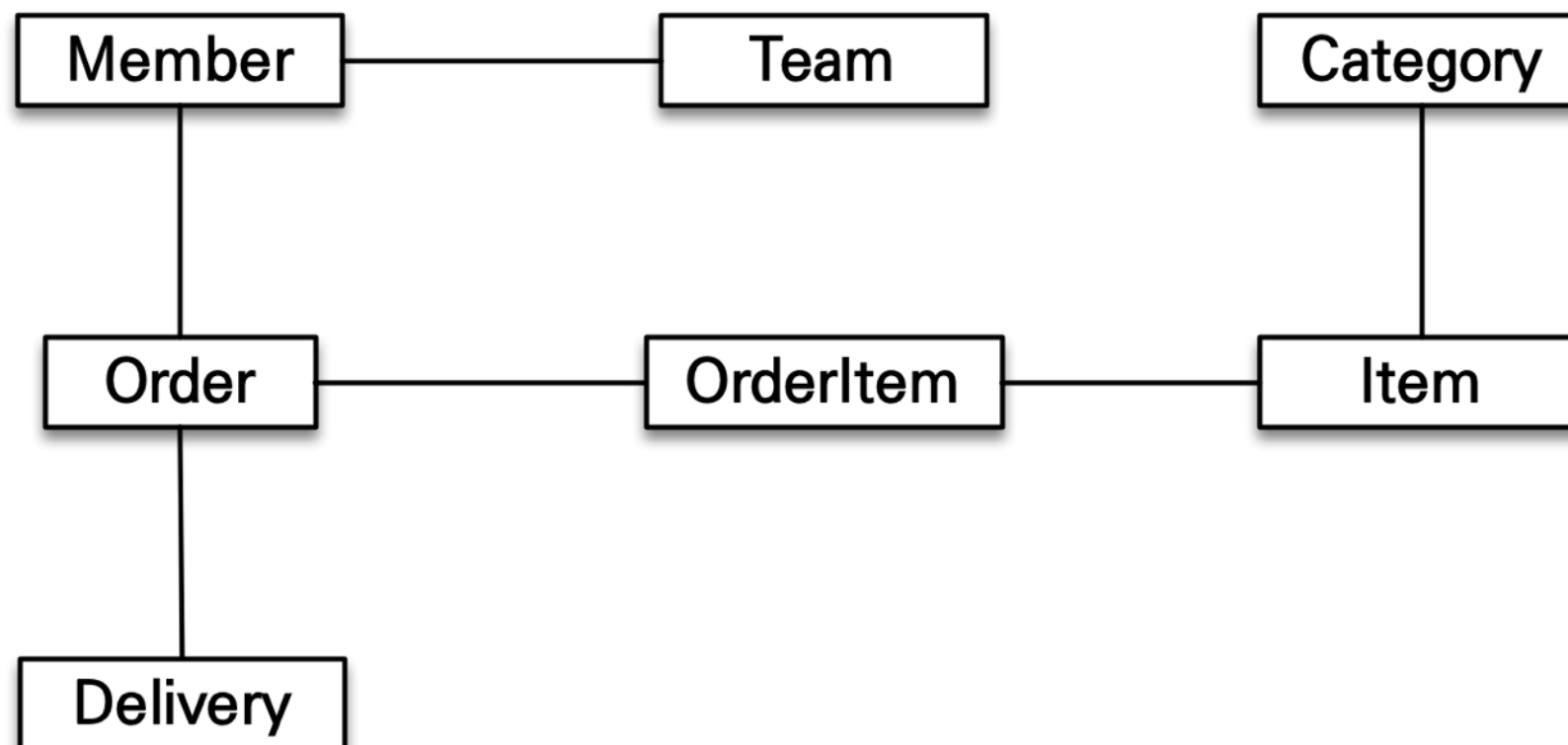
객체 모델링 자바 컬렉션에 관리

```
list.add(member);
```

```
Member member = list.get(member);  
member.getTeam();
```

객체 그래프 탐색

- 객체는 자유로운 객체 그래프 탐색
- `member.getOrder().getOrderItem()...`



처음 실행하는 SQL에 따라서 탐색 가능 범위 결정

```
SELECT M.*, T.*  
  FROM MEMBER M  
  JOIN TEAM T ON M.TEAM_ID = T.TEAM_ID
```

```
member.getTeam(); //OK
```

```
member.getOrder(); //null
```

처음 실행하는 SQL에 따라서 탐색 가능 범위 결정

```
class MemberService {  
    ...  
    public void process() {  
        Member member = memberDAO.find(memberId);  
        member.getTeam(); //???  
        member.getOrder().getDelivery(); // ???  
    }  
}
```

모든 객체를 미리 로딩할 수는 없다.

- 상황에 따라 동일한 회원 조회 메서드가 여러벌 생성

```
memberDAO.getMember(); //Member만 조회
```

```
memberDAO.getMemberWithTeam(); //Member와 Team 조회
```

```
//Member, Order, Delivery
```

```
memberDAO.getMemberWithOrderWithDelivery();
```

비교하기

```
String memberId = "100";  
Member member1 = memberDAO.getMember(memberId);  
Member member2 = memberDAO.getMember(memberId);
```

```
member1 == member2; //다르다.
```

```
class MemberDAO {
```


```
    public Member getMember(String memberId) {  
        String sql = "SELECT * FROM MEMBER WHERE MEMBER_ID = ?";  
        ...  
        //JDBC API, SQL 실행  
        return new Member(...);  
    }
```

```
}
```


비교하기 - 자바 컬렉션에서 조회

```
String memberId = "100";  
Member member1 = list.get(memberId);  
Member member2 = list.get(memberId);
```

```
member1 == member2; //같다.
```



객체답게 모델링 할수록
매핑 작업만 늘어난다.

어느 객체 지향 개발자의 고민



객체를 자바 컬렉션에 저장 하듯이
데이터베이스에 저장할 순 없을까?

어느 객체 지향 개발자의 고민



JPA 소개

Java Persistence API

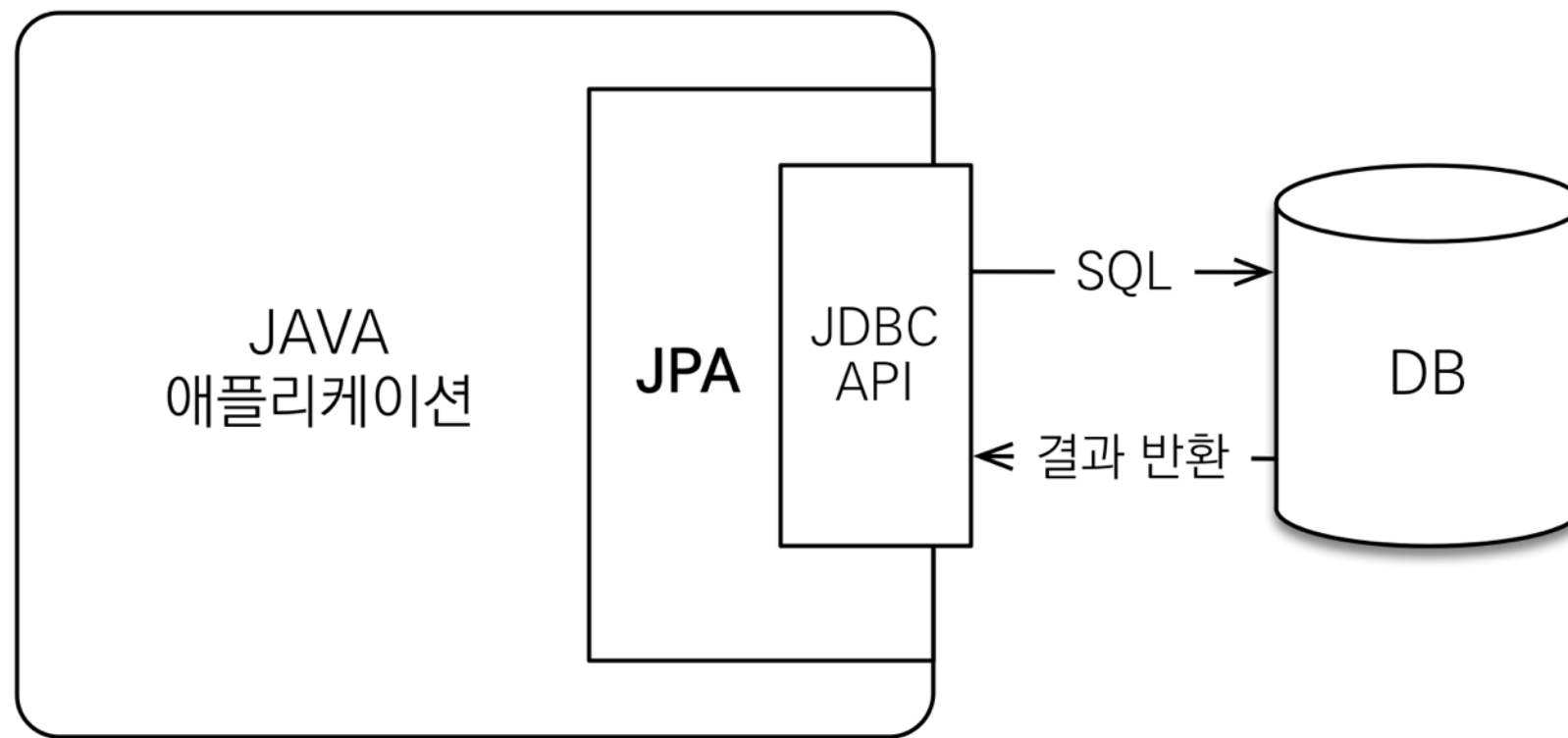
JPA?

- Java Persistence API
- 자바 진영의 ORM 기술 표준

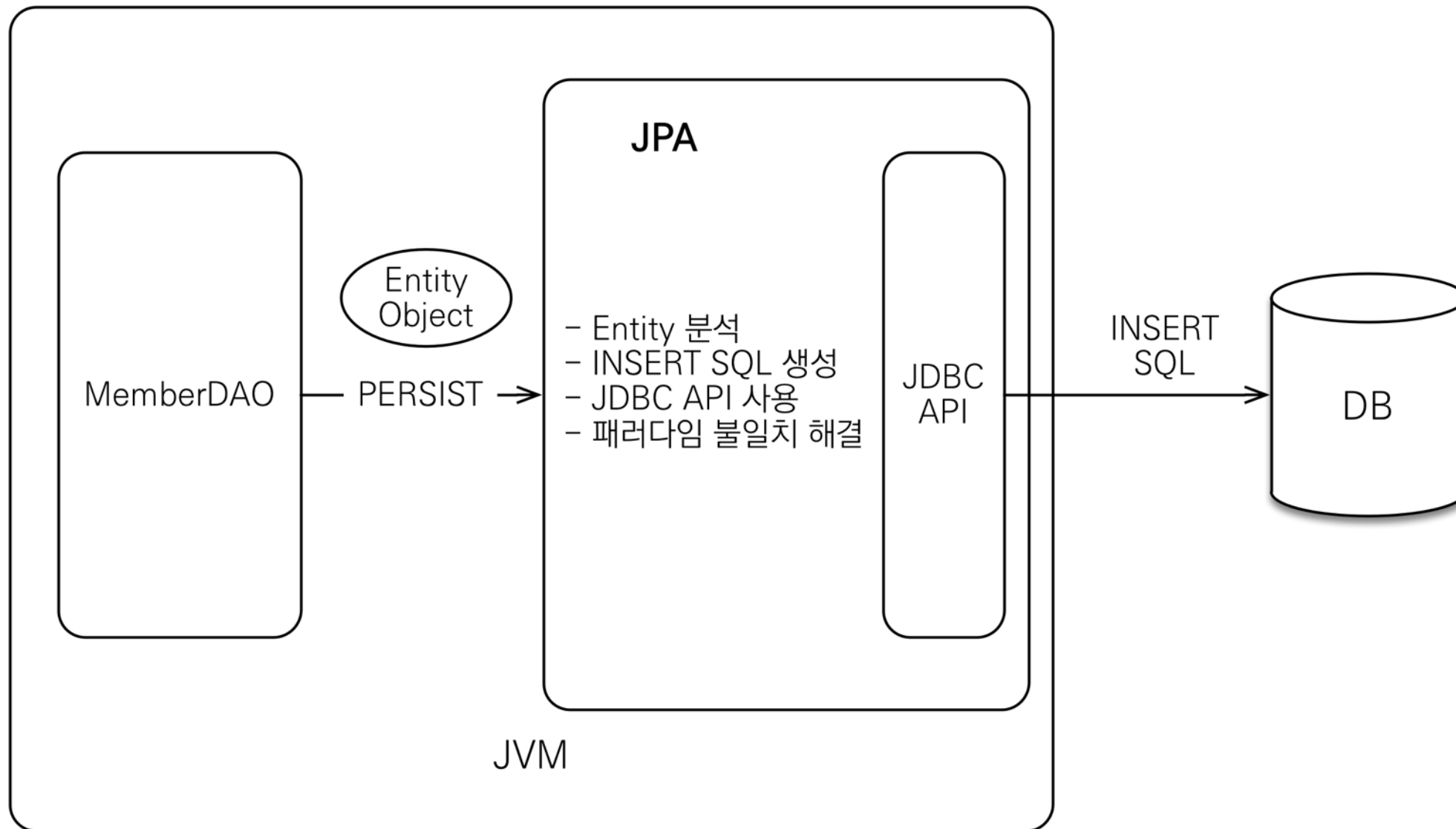
ORM?

- Object-relational mapping
- 객체 관계 매핑
- 객체는 객체대로 설계
- 관계형 데이터베이스는 관계형 데이터베이스대로 설계
- ORM 프레임워크가 중간에서 매핑
- 대중적인 언어에는 대부분 ORM 기술이 존재

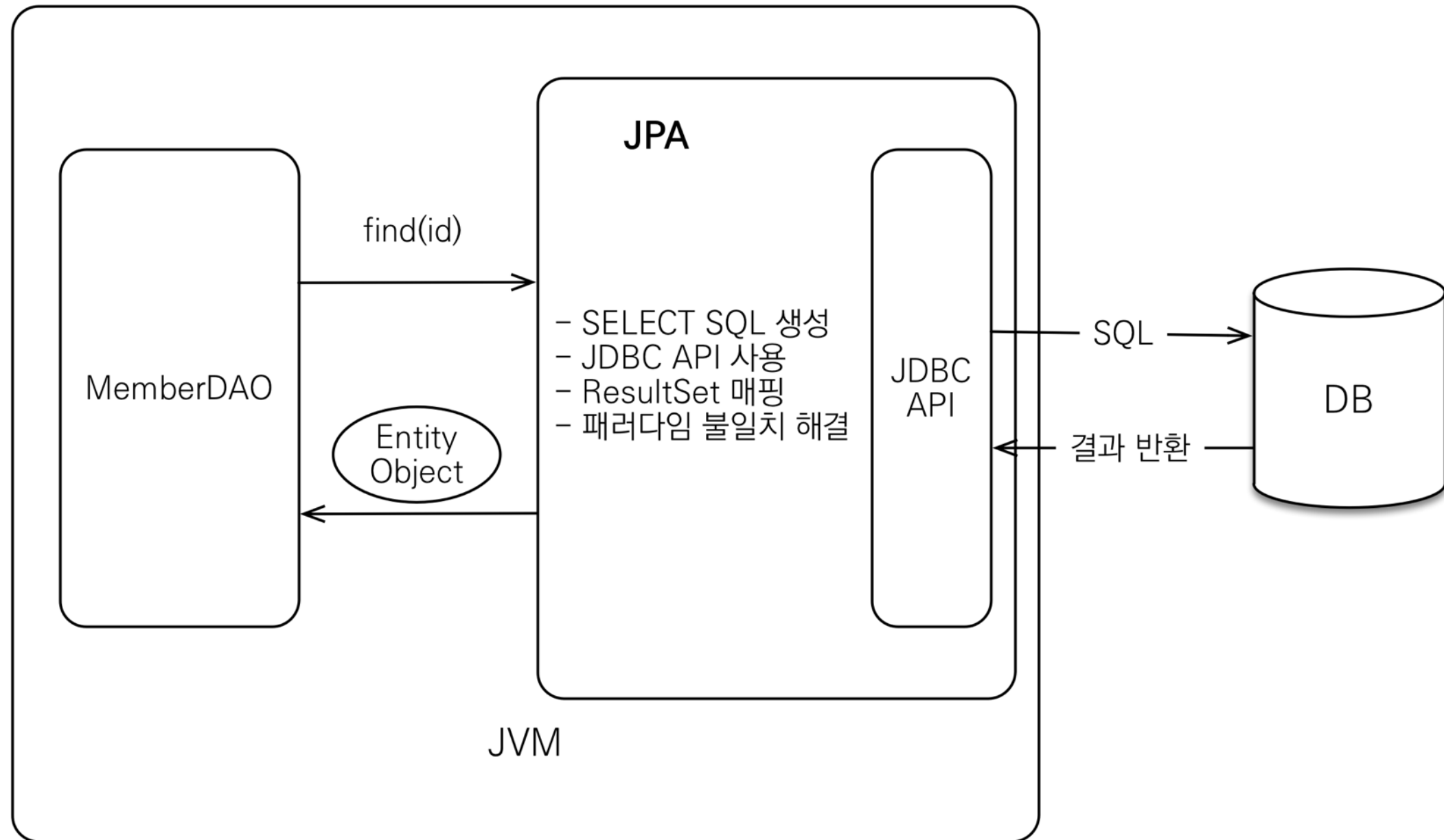
JPA는 애플리케이션과 JDBC 사이에서 동작



JPA 동작 - 저장



JPA 동작 - 조회



JPA 소개

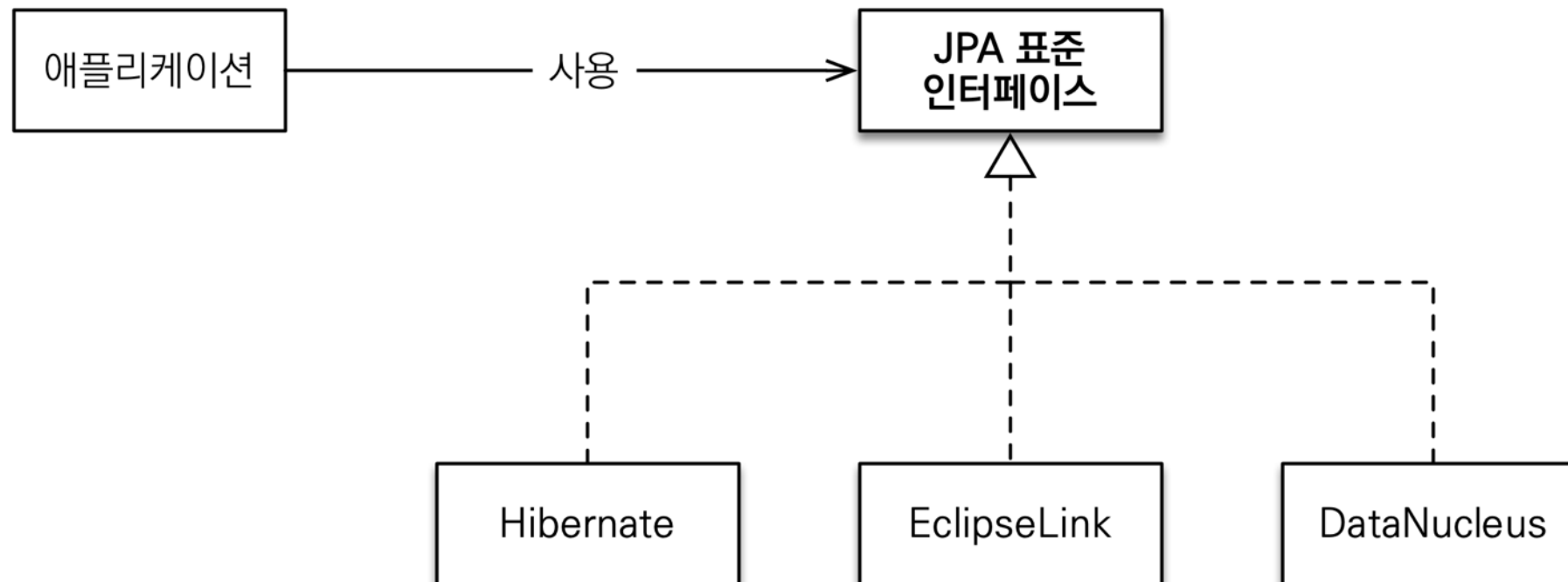
하이버네이트(오픈 소스)

EJB - 엔티티 빈(자바 표준)

JPA(자바 표준)

JPA는 표준 명세

- JPA는 인터페이스의 모음
- JPA 2.1 표준 명세를 구현한 3가지 구현체
- 하이버네이트, EclipseLink, DataNucleus



JPA 버전

- JPA 1.0(JSR 220) 2006년 : 초기 버전. 복합 키와 연관관계 기능이 부족
- JPA 2.0(JSR 317) 2009년 : 대부분의 ORM 기능을 포함, JPA Criteria 추가
- JPA 2.1(JSR 338) 2013년 : 스토어드 프로시저 접근, 컨버터 (Converter), 엔티티 그래프 기능이 추가

JPA를 왜 사용해야 하는가?

- SQL 중심적인 개발에서 객체 중심으로 개발
- 생산성
- 유지보수
- 패러다임의 불일치 해결
- 성능
- 데이터 접근 추상화와 벤더 독립성
- 표준

생산성 - JPA와 CRUD

- `jpa.persist(member);` //저장
- `Member member = jpa.find(memberId);` //조회
- `member.setName("변경할 이름")` // 수정
- `jpa.remove(member)` //삭제

유지보수 - 필드 변경시 모든 SQL 수정

```
public class Member {
```

```
    private String memberId;
```

```
    private String name;
```

```
    private String tel;
```

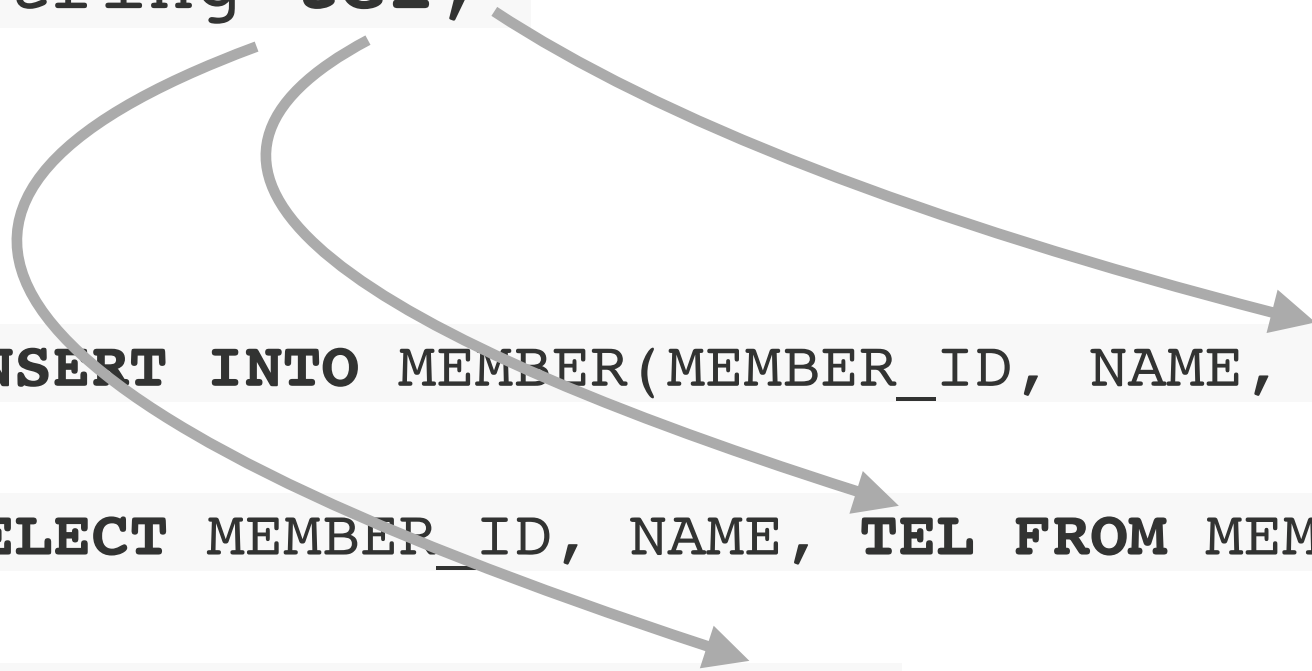
```
    ...
```

```
}
```

INSERT INTO MEMBER(MEMBER_ID, NAME, TEL) VALUES ...

SELECT MEMBER_ID, NAME, TEL FROM MEMBER M

UPDATE MEMBER SET ... TEL=?



유지보수 - 필드만 추가하면 됨, SQL은 JPA가 알아서 처리

```
public class Member {
```

```
    private String memberId;
```

```
    private String name;
```

```
    private String tel;
```

```
    ...
```

```
}
```

```
INSERT INTO MEMBER(MEMBER_ID, NAME, TEL) VALUES ...
```

```
SELECT MEMBER_ID, NAME, TEL FROM MEMBER M
```

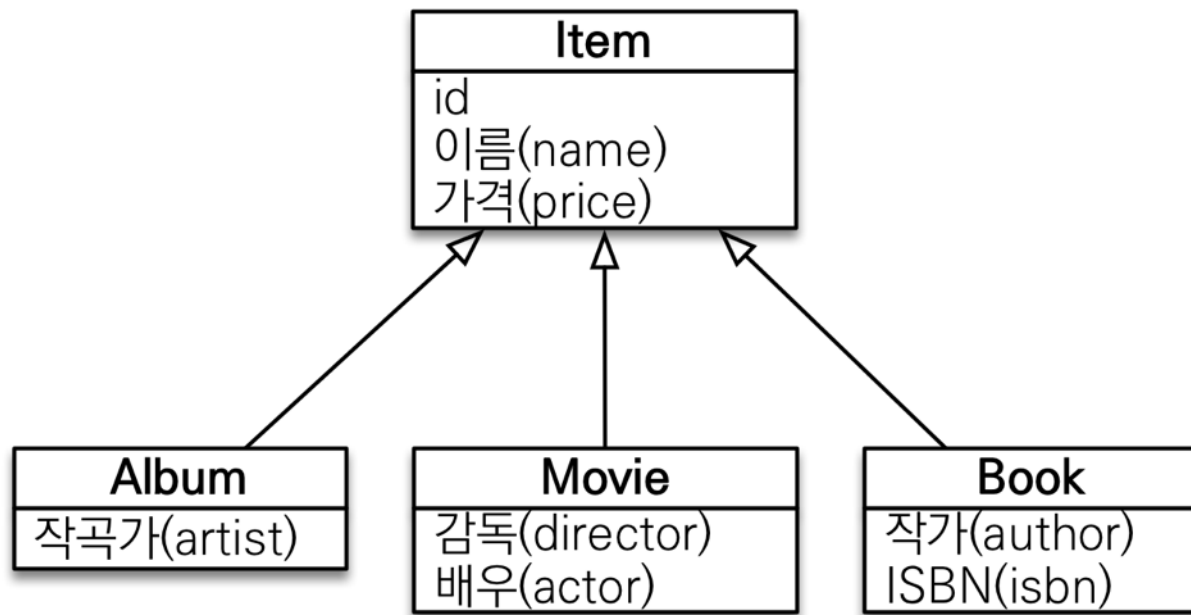
```
UPDATE MEMBER SET ... TEL=?
```



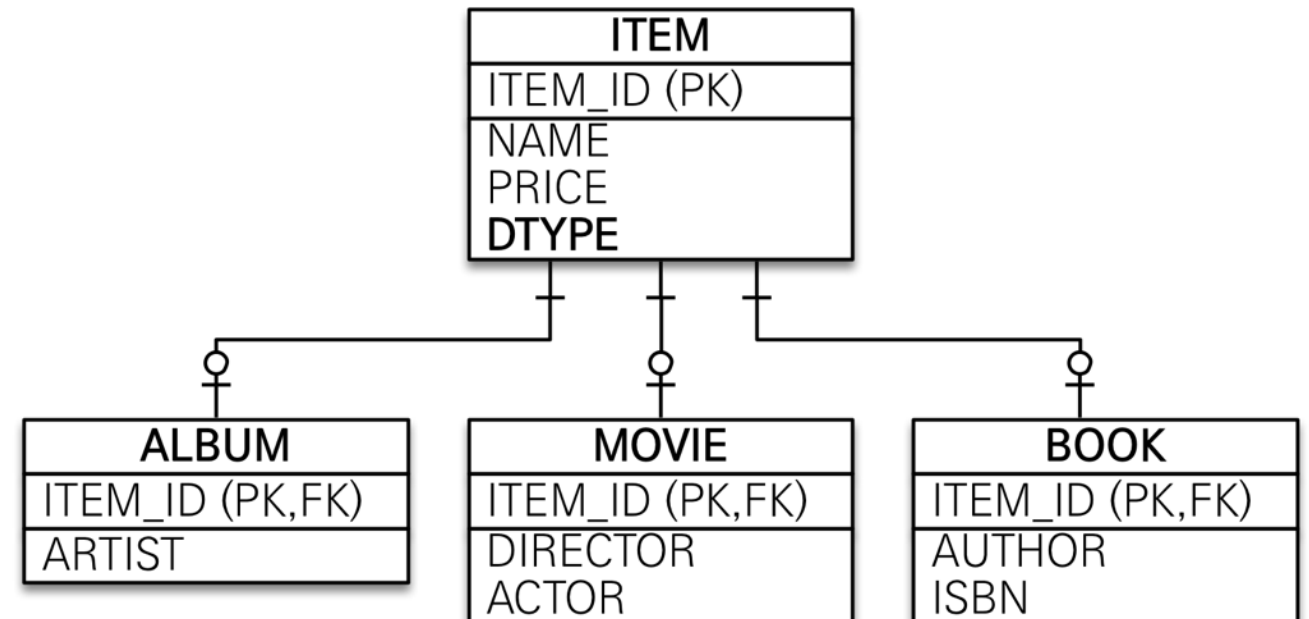
JPA와 패러다임의 불일치 해결

- JPA와 상속
- JPA와 연관관계
- JPA와 객체 그래프 탐색
- JPA와 비교하기

JPA와 상속



[객체 상속 관계]



[Table 슈퍼타입 서브타입 관계]

JPA와 상속

```
jpa.persist(album);
```

개발자가 할일

```
INSERT INTO ITEM ...  
INSERT INTO ALBUM ...
```

나머진 JPA가 처리

JPA와 상속

```
Album album = jpa.find(Album.class, albumId);
```

개발자가 할일

```
SELECT I.*, A.*  
FROM ITEM I  
JOIN ALBUM A ON I.ITEM_ID = A.ITEM_ID
```

나머진 JPA가 처리

JPA와 연관관계, 객체 그래프 탐색

```
member.setTeam(team);  
jpa.persist(member);
```

연관관계 저장

```
Member member = jpa.find(Member.class, memberId);  
Team team = member.getTeam();
```

객체 그래프 탐색

신뢰할 수 있는 엔티티, 계층

```
class MemberService {  
    ...  
    public void process() {  
        Member member = memberDAO.find(memberId);  
        member.getTeam(); //자유로운 객체 그래프 탐색  
        member.getOrder().getDelivery();  
    }  
}
```

JPA와 비교하기

```
String memberId = "100";  
Member member1 = jpa.find(Member.class, memberId);  
Member member2 = jpa.find(Member.class, memberId);  
  
member1 == member2; //같다.
```

동일한 트랜잭션에서 조회한 엔티티는 같음을 보장

많이 하는 질문

- ORM 프레임워크를 사용하면 SQL과 데이터베이스는 잘 몰라도 되나요?
- 성능이 느리진 않나요?
- 통계 쿼리처럼 매우 복잡한 SQL은 어떻게 하나요?
- MyBatis와 어떤 차이가 있나요?
- 하이버네이트 프레임워크를 신뢰할 수 있나요?
- 제 주위에는 MyBatis(iBatis, myBatis)만 사용하는데요?
- 학습곡선이 높다고 하던데요?



ORM은 객체와 RDB
두 기둥위에 있는 기술

객체 지향, 테이블 모델링 ...