JPA와 JDBC

JDBC의 단점과 JPA의 장점을 알아보자

발표자 : 다람쥐

발표날짜 : 2020년 4월 25일

JDBC 너는 무엇이냐?

자바에서 데이터베이스에
접속하도록 하는
자바 API

JDBC 너는 무엇이냐?

java.sql 패키지에 인터페이스로 존재
JDBC 드라이버로 실제 DB와 통신 구현

JDBC 너는 무엇이냐?

프로그램 — JDBC Interface — JDBC Driver — DB

코드! 코드를 보자

```
try (Connection conn = DriverManager.getConnection(
      "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
      "myLogin",
      "myPassword" ) ) {
      /* you use the connection here */
       try (Statement stmt = conn.createStatement()) {
           stmt.executeUpdate( "INSERT INTO MyTable( name ) VALUES ( 'my
       name' )" );
       }
}  // the VM will take care of closing the connection
```

코드! 코드를 보자

```
try (Connection conn = DriverManager.getConnection(
      "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
      "myLogin",
      "myPassword" ) ) {
      /* you use the connection here */
       try (Statement stmt = conn.createStatement()) {
           stmt.executeUpdate( "INSERT INTO MyTable( name ) VALUES ( 'my
       name' )" );
       }
}  // the VM will take care of closing the connection
```

언제까지 SQL문을 직접 쓸래?

SQL문을 텍스트로 쓰다보니
실수할 위험이 크다.

SQL Mapper의 등장

SQL문을 작성할 때의
실수를 최대한 줄여보자

SQL Mapper의 등장

한 눈에 알아보기 쉽고
실수하자마자 오류를 발생시키도록

SQL Mapper의 등장

마이바티스
아이바티스

SQL Mapper의 등장

매핑 구문 두둥등장

```
<insert
  id="insertAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  keyProperty=""
  keyColumn=""
  useGeneratedKeys=""
  timeout="20">

<update
  id="updateAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">

<delete
  id="deleteAuthor"
  parameterType="domain.blog.Author"
  flushCache="true"
  statementType="PREPARED"
  timeout="20">
```

SQL Mapper의 장점

복잡한 쿼리 구문을 잘 만들어준다.
까먹지 않도록 파라미터를 명시해준다.

근데 말이야...

SQL 전문을 그대로 쓰지 않을 뿐
SQL 문법을 쓴다고 느끼는건 왜일까?

객체지향과 DB

21세기에 객체지향 프로그래밍을 쓰는데
왜 DB와 객체를 연결시켜
객체지향 개념을 적용할 수 없는 걸까?

객체지향과 DB

객체는 그저 DB의 데이터를 받는
데이터 덩어리여야만 하는걸까?

그 전에...

SQL을 직접 다룰 때의 문제점을
파헤쳐보자

반복, 반복 그리고 반복

1. 객체와 DAO를 만든다.
2. 조회 SQL을 만든다.
3. JDBC API로 SQL을 실행한다.
4. 결과를 객체로 매핑한다.

반복, 반복 그리고 반복

자 이제 등록하는 기능을 만들어볼까?

반복, 반복 그리고 반복

1. 객체와 DAO를 만든다.
2. 등록 SQL을 만든다.
3. JDBC API로 SQL을 실행한다.
4. 결과를 객체로 매핑한다.

반복, 반복 그리고 반복

1. 객체와 DAO를 만든다.
2. 수정 SQL을 만든다.
3. JDBC API로 SQL을 실행한다.
4. 결과를 객체로 매핑한다.

반복, 반복 그리고 반복

잠만.. 뭔가 이상한데?

반복, 반복 그리고 반복

그냥 DB에 객체로 알아서
저장하면은 편할텐데...

```
userList.add(member);
```

SQL에 의존적인 개발

요구사항이 도착했다!

기획자 : 회원의 연락처를 추가해주세요!

SQL에 의존적인 개발

연락처 쯤이야.. 쉽지!

SQL에 의존적인 개발

```
private String tel;
```

SQL에 의존적인 개발

이렇게만 적어놓으면
SQL에 연락처 필드에 등록이
안되겠지?

SQL에 의존적인 개발

조회, 등록, 수정 SQL 코드 수정

객체 Setter,Getter 코드 추가

SQL에 의존적인 개발

연관 객체라면..?

```
private Team team;
```

SQL에 의존적인 개발

Member와 Team 테이블을
Join 시켜 Team 필드까지
매핑시키자

SQL에 의존적인 개발

새로운 메소드

Public Member findWithTeam(String memberId) {}

추가

SQL에 의존적인 개발

연관 객체를 불러오는지는
SQL문을 봐야 안다.

SQL에 의존적인 개발

아무리 DAO 계층을 숨겨도
어쩔 수 없이 들춰야한다.

요약

진정한 의미의 계층 분할이 어렵다.
엔티티를 신뢰할 수 없다.
SQL 의존적 개발을 피할 수 없다.

JPA 등장

자바 ORM 프레임워크
객체 모델링과 관계형 DB 연결

JPA는 어떻게 객체를 저장할까

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

JPA는 어떻게 객체를 조회할까

```
Member meber = jpa.find(Member.class, memberId);
// 조회
```

JPA는 어떻게 객체를 수정할까

```
Member meber = jpa.find(Member.class, memberId);
member.setName( “ 이름변경 ” ); // 수정
```

JPA는 어떻게 연관 객체를 조회할까

```
Member meber = jpa.find(Member.class, memberId);
Team team = member.getTeam(); // 연관 객체 조회
```

패러다임 불일치

객체지향 vs 관계형 DB

패러다임 불일치

[객체지향]

추상화

상속

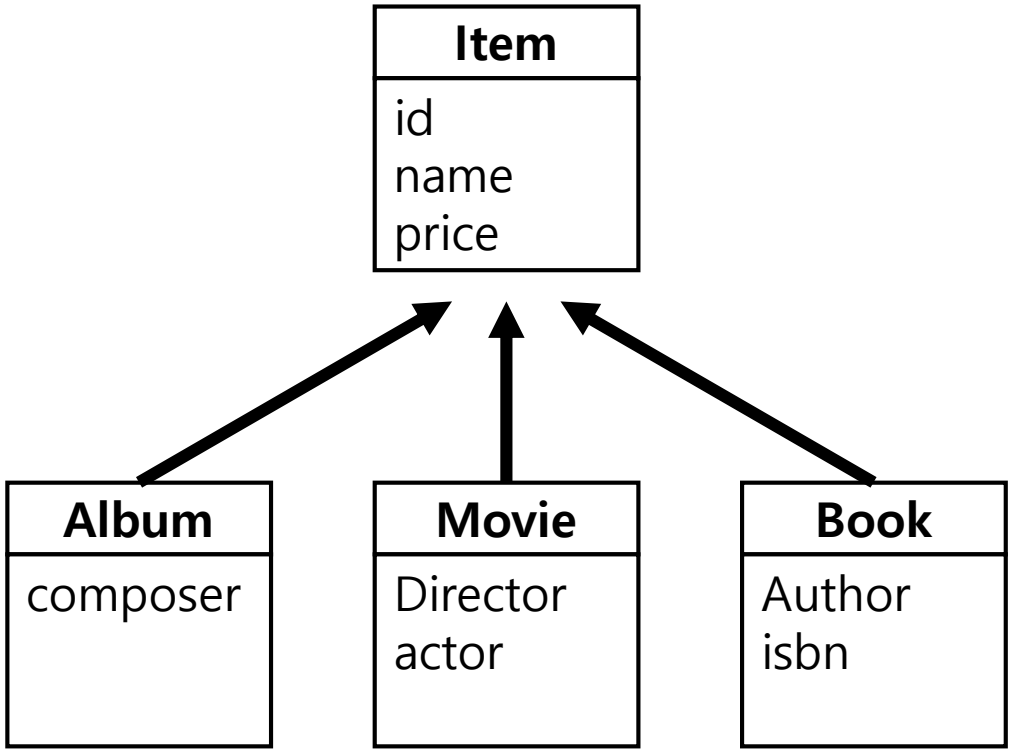
다형성

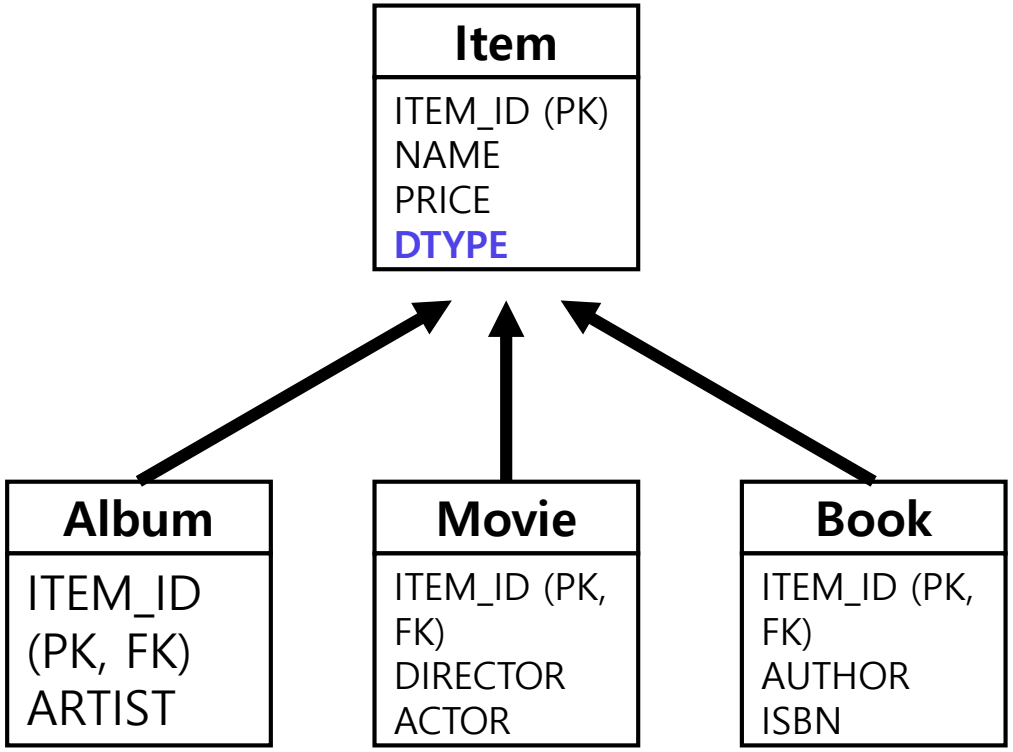
[관계형 DB]

데이터 중심

집합적

JPA와 패러다임 불일치 - 상속





JPA와 패러다임 불일치 - 상속

Movie를 저장하면
Item 과 Movie SQL 분리

JPA와 패러다임 불일치 - 연관 객체

```
Class Member {
    String id;
    Team team;
    String username;

    Team getTeam() {
        return team;
    }
}
```

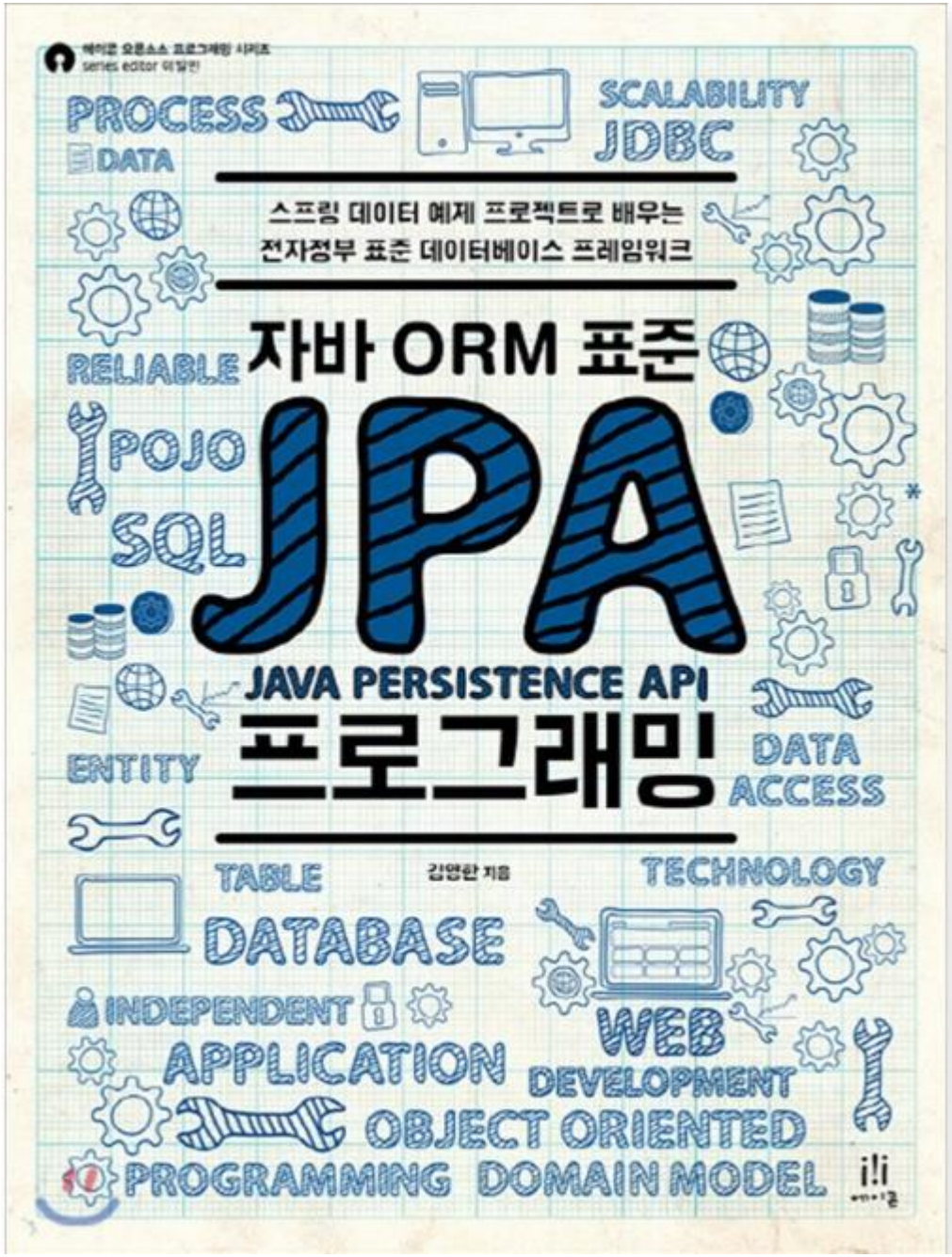
```
Class Team {
        Long id;
        String name;
}
```

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

JPA와 패러다임 불일치 - 연관 객체

연관 객체까지
저장된다.

참고 도서


PROCESS
DATA
SCALABILITY
JDBC
스프링 데이터 예제 프로젝트로 배우는
전자정부 표준 데이터베이스 프레임워크
RELIABLE
자바 ORM 표준
POJO
SQL
JPA
JAVA PERSISTENCE API
ENTITY
프로그래밍
DATA
ACCESS
TABLE
김영한 지음
TECHNOLOGY
DATABASE
INDEPENDENT
WEB
APPLICATION
DEVELOPMENT
OBJECT ORIENTED
PROGRAMMING
DOMAIN MODEL

감사합니다.