ORM

Find similar titles

2017-03-02 16:43 (rev. 4)

green

Category

Programming

Table of Contents

ORM이란?

- ORM(Object-relational mapping)을 단순하게 표현하면 객체와 관계와의 설정이라 할 수 있다. ORM에서 말하는 객체(Object)의 의미는 우리가 흔히 알고 있는 OOP(Object_Oriented Programming)의 그 객체를 의미한다는 것을 쉽게 유추할 수 있을 것이다. 그렇다면 과연 관계라는 것이 의미하는 것은 무엇일까? 지극히 기초적인 이야기지만 개발자가 흔히 사용하고 있는 관계형 데이터베이스를 의미한다.

- 그렇다면 도대체 무엇이 문제여서 객체와 관계형 데이터베이스 간의 매핑을 지원해주는 Framework나 Tool들이 나오는 것일까? ORM framework나 도구가 없던 시절에도 이미 우리는 OOP를 하면서 객체와 관계형 데이터베이스를 모두 잘 사용하고 있었음에도 불구하고, 굳이 이런 새로운 개념들이 나오게 되는 이유는 "Back to basics(기본에 충실하자)" 을 지키기 위해서라고 볼 수 있다. 즉, 보다 OOP다운 프로그래밍을 하자는데부터 출발한 것이다.

- 그럼 과연 무엇이 문제였던 것일까? 우리가 어떤 어플리케이션을 만든다고 하면 관련된 정보들을 객체에 담아 보관하게 된다. 프로그래밍 실습 예제의 단골 손님격인 주소록을 만든다고 가정했을 때 주소록의 주체가 될 사람이라는 객체에는 주민등록번호, 이름, 키, 몸무게 등이 저장될 것이고 주소나 전화번호 같은 추가로 저장될 객체들이 연결 될 것이다. 이렇게 생성한 사람 객체를 영구적으로 저장하기 위해 파일이나 데이터베이스에 입력한다는 것은 객체와 그와 연결된 객체들을 데이터베이스의 테이블에 저장 한다는 것을 의미한게 된다. 즉, 테이블(Table)에 객체가 가지고 있던 정보를 입력하고, 이 테이블들을 "join"과 같은 SQL 질의어를 통해 관계 설정을 해 주게 된다. 여기서 문제는 이 테이블과 객체간의 이질성이 발생 하게 된다는 것이다.

사용예시

- 보통 ORM Framwork들은 이러한 이질성을 해결하기 위해서 객체와 테이블간의 관계를 설정하여 자동으로 처리하게 되는데 예시를 통해 확인하면 다음과 같다.

```
public class Person{
    private String name;
    private String height;
    private String weight;
    private String ssn;
    //implement getter & setter methods
}
```

- iBatis의 경우에는 다음과 같이 mapping file내에서 해당 query의 결과를 받을 객체를 지정해 줄 수 있다

```
<select id="getPerson" resultClass="net.agilejava.person.domain.Person">
    SELECT name, height, weight, ssn FROM USER WHERE name = #name#;
</select>
```

- 즉, getPerson 이라고 정의된 질의어 결과는 net.agilejava.person.domain의 Person객체에 자동으로 mapping 되는 것이다. Hibernate의 경우에는 mapping 파일에서 다음과 같이 표현을 해준다.

```
<hibernate-mapping>
    <class name="net.agilejava.person.domain.Person" table="person">
        <id name="name" column="name"/>
        <property name="height" column="height"/>
        <property name="weight" column="weight"/>
        <property name="ssn" column="ssn"/>
    <class>
</hibernate-mapping>
```

- 위 두개의 Framework의 예시를 보면 알 수 있듯이 setter 메소드가 있으면 객체에 결과를 set하는 작업을 따로 해줄 필요 없이 자동으로 해당 값이 할당되게 된다. 물론 여기에 1:m 이나 m:1등의 관계들이 형성되면 추가적인 작업이 필요하긴 하지만 일단 눈에 보이는 간단한 부분은 처리가 되는 것을 볼 수 있다. 물론 반대의 경우에도 객체를 던져주면 ORM Framework에서 알아서 get을 수행해 해당하는 column에 넣어주게 된다.

- 어떻게 보면 더 복잡해 보일 수도 있는 ORM이지만 막상 사용해 보면 그 편리함에 몸을 떨게 된다. 단순하게 get/set만 해주는게 목적이 아니라 객체지향적인 시스템을 위해서 관계형 데이터베이스의 설계부터 변화를 주고, 설계된 데이터베이스와 객체와의 관계에 대한 설정 등을 포함하여 보다 객체지향적인 시스템의 완성을 위한 도구라고 말할 수 있겠다. 물론, ORM 이라는 것이 흔히 말하는 silver bullet은 절대 아니다. 많은 사람들이 ORM에 대해 우려하고 있는 부분은 객체지향적으로 설계되지 않은 데이터베이스에서의 사용에 따른 폐해라고 생각한다. 이미 데이터베이스 중심적인 사고를 통해 만들어 놓은 데이터베이스에 ORM을 도입 해도 분명 이점이 있긴 하겠지만, 그에 비해 개발자들의 학습곡선이라던지, 기존에 존재하는 코드나 시스템들과의 연계 또는 유지보수 적인 측면, 그리고 성능 등에서 생각해보면 부정적으로 볼 수 밖에 없다. 즉, 전체 적인 시스템의 분석, 설계 단계에서부터 객체와 데이터베이스를 따로 생각하는 것이 아니라 하나의 덩어리로 인지하고 양쪽 모두를 고려한 설계를 해나갈 수 있을 때, ORM은 보다 좋은 모습을 보여주고 각광을 받을 수 있을 것이다.

장점

- 객체 지향적인 코드로 인해 더 직관적이고 비즈니스 로직에 더 집중할 수 있게 도와준다.
  - 선언문, 할당, 종료 같은 부수적인 코드가 없거나 급격히 줄어든다.
  - 각종 객체에 대한 코드를 별도로 작성하기 때문에 코드의 가독성을 올려준다.
  - SQL의 절차적이고 순차적인 접근이 아닌 객체 지향적인 접근으로 인해 생산성이 증가한다.

- 재사용 및 유지보수의 편리성이 증가한다.
  - ORM은 독립적으로 작성되어있고, 해당 객체들을 재활용 할 수 있다.
  - 때문에 모델에서 가공된 데이터를 컨트롤러에 의해 뷰와 합쳐지는 형태로 디자인 패턴을 견고하게 다지는데 유리하다.
  - 매핑정보가 명확하여, ERD를 보는 것에 대한 의존도를 낮출 수 있다.

- DBMS에 대한 종속성이 줄어든다.
  - 대부분 ORM 솔루션은 DB에 종속적이지 않다.
  - 종속적이지 않다는것은 구현 방법 뿐만아니라 많은 솔루션에서 자료형 타입까지 유효하다.
  - 프로그래머는 Object에 집중함으로 극단적으로 DBMS를 교체하는 거대한 작업에도 비교적 적은 리스크와 시간이 소요된다.
  - 또한 자바에서 가공할경우 equals, hashCode의 오버라이드 같은 자바의 기능을 이용할 수 있고, 간결하고 빠른 가공이 가능하다.

단점

- 완벽한 ORM 으로만 서비스를 구현하기가 어렵다.
  - 사용하기는 편하지만 설계는 매우 신중하게 해야한다.
  - 프로젝트의 복잡성이 커질경우 난이도 또한 올라갈 수 있다.
  - 잘못 구현된 경우에 속도 저하 및 심각할 경우 일관성이 무너지는 문제점이 생길 수 있다.
  - 일부 자주 사용되는 대형 쿼리는 속도를 위해 SP를 쓰는등 별도의 튜닝이 필요한 경우가 있다.
  - DBMS의 고유 기능을 이용하기 어렵다. (하지만 이건 단점으로만 볼 수 없다 : 특정 DBMS의 고유기능을 이용하면 이식성이 저하된다.)

- 프로시저가 많은 시스템에선 ORM의 객체 지향적인 장점을 활용하기 어렵다.
  - 이미 프로시저가 많은 시스템에선 다시 객체로 바꿔야하며, 그 과정에서 생산성 저하나 리스크가 많이 발생할 수 있다.

Incoming Links

Related Data Sciences

- Celery with Django