2021-09-08

<JPA>

-JPA : Java Persistence API

현재 자바 진형의 ORM 기술 표준으로 인터페이스의 모음

실제로 동작하는 것은 아니다.

JPA 인터페이스를 구현한 대표적인 오픈소스가 Hibernate라고 할 수 있다.

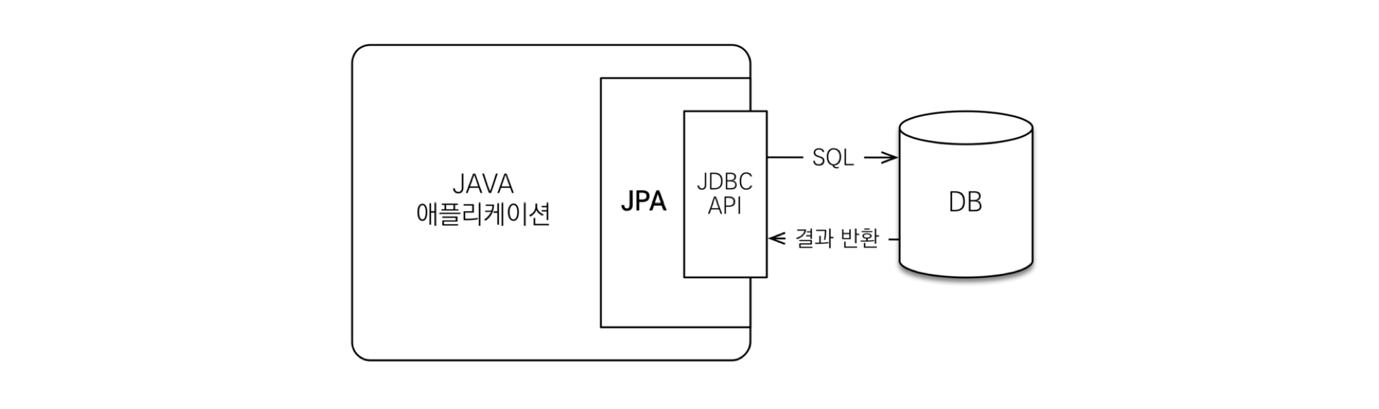
Ex) JPA 2.1 표준 명세를 구현한 3가지 구현체 : Hibernate, EclipseLink, DataNucleus

-JPA 동작 과정

JPA는 애플리케이션과 JDBC 사이에서 동작한다ㅏ.

개발자가 JPA를 사용하면, JPA 내부에서 JDBC API를 사용하여 SQL을 호출하여 DB와 통신한다.

개발자가 직접 JDBC API를 쓰는 것은 아니다.



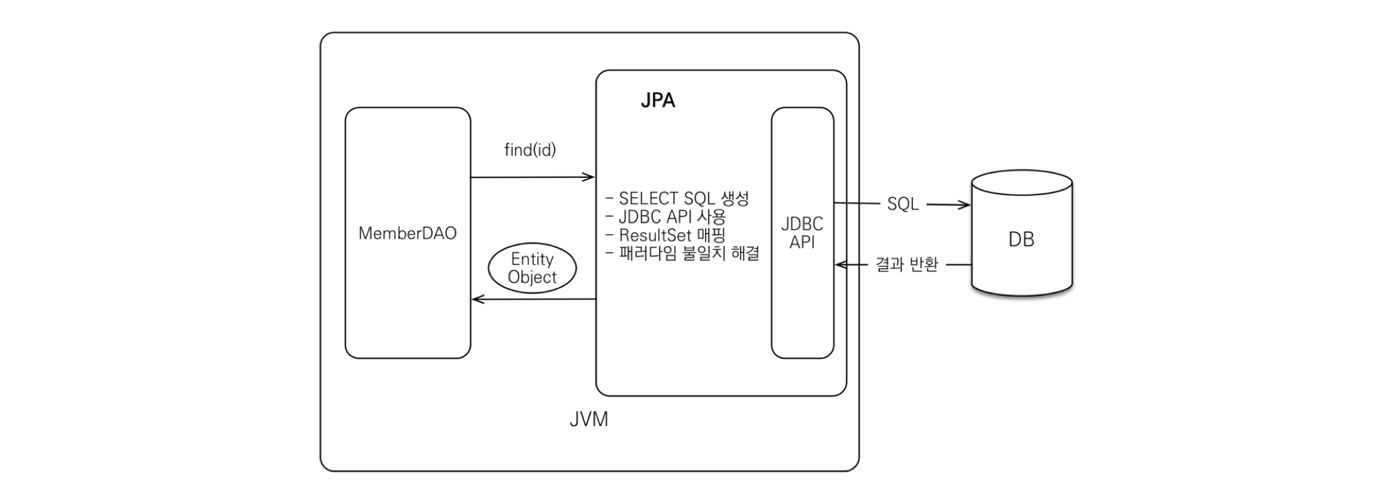
-JPA 저장 과정

Ex) MemberDAO에 객체를 저장하고 싶을 때 개발자는 JPA애 Member 객체를 넘긴다.

JPA는1)Member엔티티를분석한다.2)INSERTSQL을생성한다.

3)JDBC API를사용하여SQL을DB에날린다.

-JPA 조회 과정



-ORM : Object – relational mapping

객체 관계 매핑 => 객체는 객체대로 생성하고, 관계형 데이터베이스는 관계형 데이터베이스대로 설계한다.

ORM 프레임워크가 중간에서 매핑해준다.

ORM은 객체와 RDB 두 기둥 위에 있는 기술

-EJB

과거의 자바 표준 (Entity Bean)

과거의 ORM이었다.

문제는 코드가 매우 지저분하고 API의 복잡성이 높고( 인터페이스는 많이 구현 해야 해서 ), 속도가 느리다.

-Hibernate

ORM 프레임워크, Open Source SW이다.

<Java의 정석 공부>

JVM : Java Virtual Machine

Java는 JVM을 한번 거치기 때문에 OS와 하드웨어에 독립적이다.

Java 애플리케이션은 일반 애플리케이션과 달리 JVM을 한번 더 거치고 하드웨어에 맞게 컴파일 된 것이 아니라 실행 시에 해석되기 때문에 속도가 느리다. => JIT 컴파일러와 향상된 기술 때문에 속도 격차 줄임

-JIT 컴파일러란? : Just-in-Time 컴파일러는 바이트코드를 컴퓨터 프로세서(CPU)로 직접 보낼 수 있는 명령어로 바꾸는 프로그램이다.  
-JIT 컴파일 : JIT 컴파일 또는 동적 번역은 프로그램을 실제 실행하는 시점에서 기계어로 번역하는 컴파일 기법

-JDK : Java Development Kit 자바 개발도구

-소스파일 이름은 public class의 이름과 일치해야 한다. 만일 소스파일 내 public class가 없다면, 소스파일의 이름은 소스파일 내의 어떤 클래스 이름으로 해도 상관없다.

public class Hello2 {}

Class Hello3 {}

=>public class가 있는 경우(Hello2로), 소스파일의 이름은 반드시 public class의 이름과 일치해야 한다.

-하나의 소스파일에 둘 이상의 public class가 존재하면 안된다. 각 클래스를 별도의 소스파일에 나눠서 저장하거나 둘 중의 한 클래스에 public을 붙이면 안된다.

-명명 규칙

1.클래스 이름의 첫 글자는 항상 대문자

2.여러 단어로 이루어진 이름의 단어의 첫 글자는 대문자

3.상수의 이름은 모두 대문자, 여러 단어로 이루어진 경우 ‘\_’로 구분한다.

변수의 타입

-자료형은 크게 기본형과 참조형으로 나눌 수 있다.

-기본형 변수는 실제 값(data)을 저장

-참조형 변수는 어떤 값이 저장되어 있는 주소를 값으로 갖게 됨.

-참조형 변수(참조변수)를 선언할 때는 변수의 타입으로 클래스의 이름을 사용 -> 클래스 이름이 참조변수의 타입이 된다.

-새로운 클래스를 작성한다는 것은 => 새로운 참조형을 추가하는 것.

-클래스 이름 변수이름 ; // 변수의 타입이 기본형이 아닌 것들은 모두 참조변수

<데이터베이스 설계>

UI로 화면을 짠 다음 어떤 기능이 들어갈지 테이블로 전부 나열해보는 것이 편하다.

벤치마킹 할 사이트를 찾아라.

눈에 보이는 테이블도 있겠지만 안보이는 기능들까지 어떤 것이 있을지 고민해보자.

테이블이(가) 표시된 사진

자동 생성된 설명

Sequel Pro를 다운받고 MySQL을 이용해서 위에 데이터 베이스 설계를 직접 작성해보고 쿼리까지도 해보기로 했다.

Sequeel Pro에 MySQL을 불러왔다. 127.0.0.1 이며 root

정상적으로 DB 접속

텍스트이(가) 표시된 사진

자동 생성된 설명

t\_movie라는 테이블을 생성하면 pri(primary Key)와 함께 extra에 auto\_increment라는 것이 생긴다. 이것은 내가 아이디 값을 입력해주지 않아도 t\_movie에 데이터 레코드가 들어가는 순간 id 필드는 자동으로 채번이 되는 것을 의미한다.

일반적으로 국가 코드는 2자리로 관리된다.

Country\_code VARCHAR 2

[VARCHAR 대신 캐릭터도 상관없다]

각 테이블에서 동일한 field값을 연결해주기 위해서는 index값을 하나 설정해주어야 연결된다. Index값 설정하지 않고 연결하려고 하면 계속 에러가 뜨게 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

두 field를 연결하면 이런식으로 된다.

연결하려는 field의 이름 값이 다른 경우는 Index를 추가하지 않아도 연결이 된다.

# ERD : Entity Relationship Model 요구분석사항을 그림으로 그려내어 그 관계를 도출하는 것.

개체-관계 모델

텍스트이(가) 표시된 사진

자동 생성된 설명

이런 식으로 데이터 베이스를 어떤 식으로 설계하고, 설계한 것에 맞는 테이블과 필드값을 만들고, 각 값들 사이에 연관된 것들을 연결 짓고

쿼리문으로 불러오는 과정을 하고 있다.

그 흐름을 잊지말자.

## 필수 입력을 하는 값이라면 allow NULL 체크를 해제해야 한다.

## unsigned 같은 경우 테이블을 보면 Type이 int 옆에 unsigned라고 적힌 것을 볼 수 있다.

모든 Integer 타입들은 속성으로 UNSIGNED를 가지고 있는데 이 타입은 칼럼 내에서 음수를 포함하지 않거나 혹은 수의 range를 양수쪽으로 더 넓게 가지고 싶을 때 사용한다.

언제사용하냐? => 해달 칼럼값이 음수가 될 일이 절대 없다면 쓰는게 좋다.

즉 type값이 VARCHAR인 경우는 unsigned를 체크 해제 하는데

Int 같은 경우는 음수값을 쓰냐 안쓰냐에 따라서 체크하거나 체크하지 않는 것 같다.

보통 id 값은 int로 받게 되는데 id값이 음수일리는 없기 때문에 unsigned에 체크를 한다.

<Git>

<https://git-scm.com/book/ko/v2> => 이거 보면서 git 공부하기

<https://www.yalco.kr/26_git_tutorial_2/> => git & github 명령어 정리

\*git config —global user.name “내 이름”

\*git config —global user.email “내 이메일”

=> 이거 2개 컴퓨터에서 한번 입력했으면 다시 입력할 필요가 없다.

Git 상태 확인 \*git status

\*git add -A : git에 기록 저장

\*git commit -m “ 내용” : 기록 저장할 것 설명

\*git log : git에 저장했던 정보들의 내용이 뜬다. -> :q 눌러서 빠져나올 수 있다.

Git에 저장한 내용들 중 어떤 특정한 상황으로 다시 돌아가고 싶다면

\*git log 하고 저장한 내용들 중 일련번호가 뜨게 되는데

\*git reset 일련번호 —hard 를 하면 그 시점으로 돌아간다. : 한번 돌아가면 다시 돌아가기 전 상태로 돌아오지 못한다.

복구가 가능하게 하려면 revert를 쓰면 된다.

\*git log를 누른 뒤 일련번호를 찾아보는데 돌아갈 시점이 아닌 취소할 시점을 찾는 것이다.

\*git revert 일련번호 : 새 커밋을 작성하는 화면이 뜬다. -> 그대로 저장하겠다는 vi 명령어 :wq를 입력해준다.

-> reset 때와 마찬가지로 파일들이 과거 상태로 돌아와있다.

-> git log 확인해보면 reset처럼 이 시점 이후의 파일들이 지워진 것이 아니라 그 시점 뒤에 새 캡슐이 묻어 있다.

-> reset으로 다시 미래로 돌아갈 수 있다.

평행우주 만들기 - branch

\*git branch my-idea 하면

현시점에서 2개의 branch로 나누겠다는 것이다.

\*git branch를 다시 해보면

기존의 branch인 master와 my-idea 2가지가 있다.

master에서 my-idea로 넘어가려면

\*git checkout my-idea를 입력하면 된다.

새로 만들어진 branch는 원래의 branch 상태를 그대로 가져간다.

### git add -A를 해서 내가 작업한 것을 추가하고

### git commit -m “ 설명 “ 을 써서 추가한 내용의 설명을 한다.

다른 우주(branch)에서 가져오기 : merge

다른 branch에서 바꾼 내용을 master내용으로 가져오고 싶을 때

Git master로 일단 이동한 다음 (git checkout master)

\*git merge 변화를가져올branch이름

->이렇게 되면 커밋 메세지를 작성할 수 있는 vi 화면으로 넘어오게 된다.

-> :wq 하면 저장하고 나가게 된다.s

\*git log —graph —all —decorate : 시각화된 여러 분기에서의 작업 내용을 볼 수 있다.

\*\*\* 여러 branch에서 작업을 할 때 or 협업할 때 가능한한 다른 branch들이 같은 파일을 수정하는 일을 피해서 작업하는 것이 좋다.

#다른 branch에서 작업한 내용을 가져오는 방법 : rebase

# 모든 변경 내용들이 한 줄로 깔끔하게 정리되었으면 좋겠을 때 merge 대신 rebase를 쓴다.

#다 쓴 branch를 삭제할 때

\*git branch -D 브랜치명

# 원격에서 branch를 삭제할 때

\*git push -d 원격명 브랜치명

\*git branch 브랜치명 : 브랜치 생성

\*git checkout 브랜치명 : 브랜치 넘어가는거

=> 합친것이 git checkout -b 브랜치명 : 브랜치 생성하고 생성한 브랜치로 넘어가기

\*git branch 는 로컬에서의 branch만 볼 수 있지만

\*git branch -a는 로컬과 원격 모두를 보여준다.

——————————————————————————————————————

### GitHub 올리는 방법 정리

\*git status 눌러서 지금까지 저장되었는지 확인하고 빠진게 있으면 전부 add해서 commit한다.

\*git remote : 현 폴더에 원격 레퍼지토리를 확인하는 명령어

\*origin이 뜰텐데 원격 레퍼지토리 이름이다.

\*git push —u origin master : 폴더의 현 브랜치에 커밋된 내용들을 origin이라는 원격 레퍼지토리, master라는 branch에 올리겠다는 뜻.

\*git push origin master : GitHub에 올라가게 된다.

## 다루지 않을 파일 설정

.gitignore : git으로 관리하고 github에 올릴 필요가 없을 때 & 오히려 그러면 안될 때 쓴다.

프로젝트 폴더 바로 안쪽 최상위 공간에 .gitignore파일을 생성한다. => 숨김 파일로 지정이 된다.

Ex) secret이라는 파일을 생성한다. -> git status를 입력하면 secret이라는 파일이 뜨게 된다.

-> .gitignore이라는 파일에 secret이란 이름을 추가, 저장하면

Git status를 실행했을 때 secret은 없어지게 된다.

ADD & commit 하고 push 하면 secret이란 파일은 github에서 안보이게 된다.

——————————————————————————————————————

### Github 소스 내려받기

public이면 누구나 private이라면 나, 해당 레퍼지토리에 초대된 사람만 가능하다. 다른 사용자를 초대하려면 그 레퍼지토리에서 setting으로 들어간 뒤 collaborators 그 다음 유저네임이나 이메일로 초대해서 private 레퍼지토리를 보거나 내가 만든 레퍼지토리에 소스를 push할 수 있도록 설정할 수 있다.

Clone or download를 누르고 그 주소를 복사한 뒤

\*git clone 복사한주소 : 이렇게 하면 vscode에서 바깥 폴더를 연 것이다.

그 폴더에서 작업하기 위해서는 cd 폴더명 을 입력해서 그 폴더로 들어가야 한다.

그 다음 git log를 해보면 프로젝트 내역들을 볼 수 있게 된다.

——————————————————————————————————————

## git pull과 git fetch의 차이

\*git pull : 원격 저장소로부터 필요한 파일을 다운 + 병합

지역 브랜치와, 원격 저장소 origin/master 가 같은 위치를 가리킨다.

\*git fetch : 원격 저장소로부터 필요한 파일을 다운(병합은 따로 해야 한다.)

지역 브랜치는 원래 가지고 있던 지역 저장소의 최근 커밋 위치를 가리키고, 원격 저장소 origin/master는 가져온 최신 커밋을 가리킨다.

신중할 때 사용한다.

사용하는 이유? 원래 내용과 바뀐 내용과의 차이를 알 수 있다 (git diff HEAD origin/master) commit이 얼마나 됐는지 알 수 있다 (git log --decorate --all --oneline) 이런 세부 내용 확인 후 git merge origin/master 하면 git pull 상태와 같아진다. (병합까지 완료)

Fork 와 clone의 차이

Fork는 다른 사람의 github repositroy에서 내가 어떤 부분을 수정하거나 추가 기능을 넣고 싶을 때 해당 repo를 내 github repo로 그대로 복제하는 기능.

Fork한 저장소는 다른 사람의 github repo와 연결되어 있다. 여기서 연결 됐다는 것은 original repo에 어떤 변화가 생기면 이는 그대로 fork된 repo로 반영할 수 있다.

이때 fetch나 rebase과정이 필요하다.