



It's Your Life

with





# MyBatis

## 연습 & 구조 이해

# DB 부터 만듭시다!





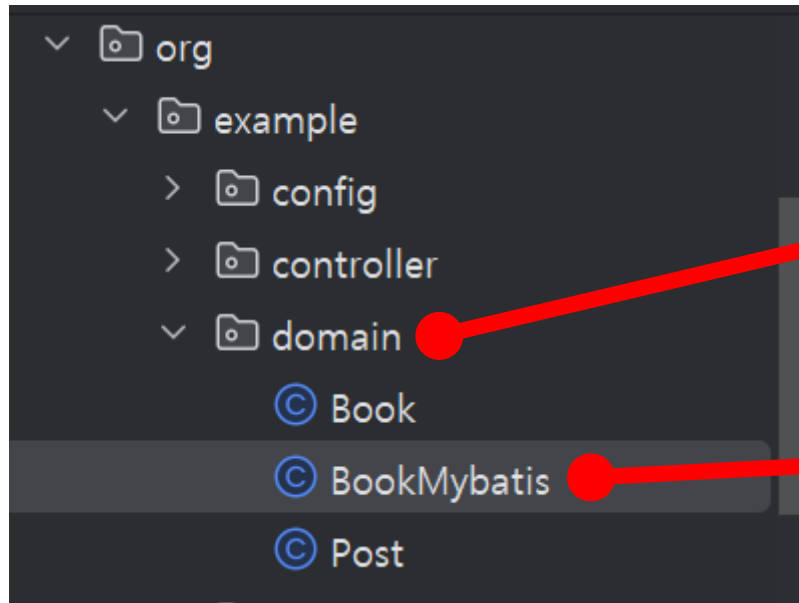
```
CREATE TABLE books
(
    id      INT AUTO_INCREMENT PRIMARY KEY,
    title   VARCHAR(50) NOT NULL,
    author  VARCHAR(50) NOT NULL
);

INSERT INTO books (title, author)
VALUES ("데미안", "헤르만 헤세"),
       ("인간의 굴레에서", "서머싯 몸"),
       ("참을 수 없는 존재의 가벼움", "밀란 쿤데라"),
       ("삶의 한가운데", "루이제 린저"),
       ("월든", "헨리 데이빗 소로우");

SELECT *
FROM books;
```

# VO 객체 만들기





좀더 실무적인 폴더 구조를 연습하기 위해서  
domain 패키지를 만들어 주세요!

BookMybatis 라는 VO 객체를 만들어 봅시다!

@Data 17 usages Tetz

@AllArgsConstructor

@NoArgsConstructor

@Builder

```
public class BookMybatis {  
    private Long id;  
    private String title;  
    private String author;  
}
```

방금 만든 DB 에 대응되는 Entity 역할을 하는  
VO 객체를 만듭시다!

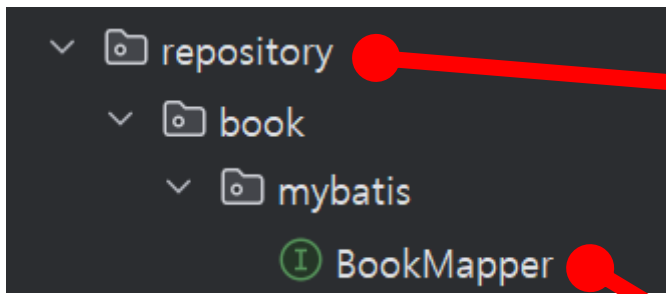




# Mapper

# 인터페이스 작업





데이터 관련 클래스는 보통  
repository 패키지에서 관리하므로  
repository 패키지를 생성!



domain 에 따라 관리를 나눠야 하므로  
book 패키지를 만들어 주세요  
  
그리고 BookMapper 인터페이스 생성!



@Mapper 어노테이션으로 Mapper Bean 으로 등록!

```
@Mapper  
public interface BookMapper {  
    public List<BookMybatis> findAll();  
    public BookMybatis findById(@Param("id") Long id);  
    public int save(BookMybatis newBookMybatis);  
    public int delete(Long id);  
}
```

우리가 사용할 기능인

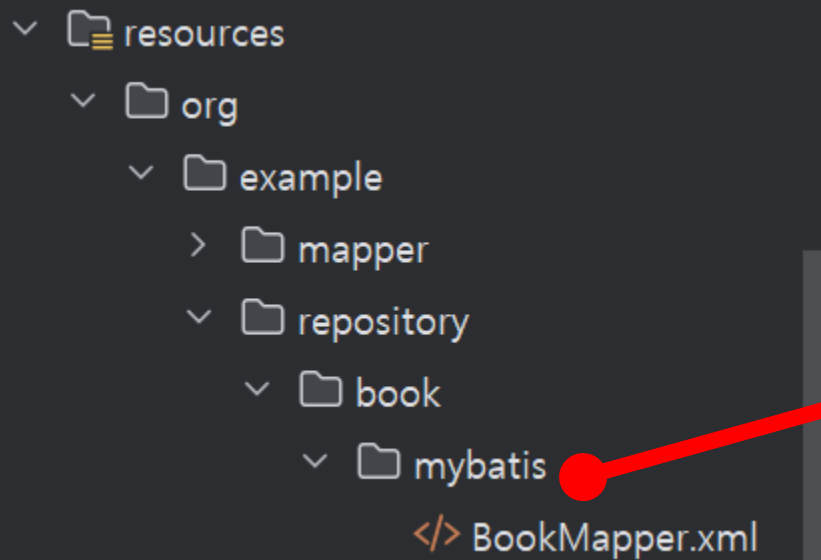
1. 전체 목록 조회
2. 특정 id 조회
3. 새로운 책 저장
4. 기존 책 삭제

기능의 추상 메서드를 선언!



# Mapper

## XML 파일 작업



```
resources
├── org
│   ├── example
│   │   ├── mapper
│   │   ├── repository
│   │   │   ├── book
│   │   │   │   └── mybatis
│   │   │   │       └── BookMapper.xml
```

Mapper XML 파일은  
자바의 패키지와 동일한 구조를 가지는 폴더에  
위치하는 것이 프로젝트의 관리 및 일관성 유지에  
좋으므로 동일한 구조의 폴더를 만들어 줍시다!

BookMapper 인터페이스가 위치하는 폴더와  
동일한 구조의 폴더를 만들어 줍니다!

+ BookMapper.xml 만들기

해당 XML 파일과 매핑 될 인터페이스를 정확히 지정 

```
<mapper namespace="org.example.repository.book.mybatis.BookMapper">
  <select id="findAll" resultType="BookMybatis">
    select * from books;
  </select>
```

인터페이스에서 선언한 추상 메서드의 이름을 정확하게 매칭

인터페이스에서 선언한 추상 메서드의 이름을 정확하게 매칭 

```
<mapper namespace="org.example.repository.book.mybatis.BookMapper">  
  <select id="findAll" resultType="BookMybatis">  
    select * from books;  
  </select>
```

해당 쿼리문에 의해 리턴 되는 데이터의 타입도 정확히 명시

findAll 메서드에서 수행하고자 하는 쿼리문을 적어주기!



```
<select id="findById" resultType="BookMybatis">  
    select * from books where id = #{id};  
</select>
```

id 값은 가변적이므로 매개변수로 받은 id 를 쓰기 위해  
#{ } 문법 사용!



그란데 말입니다

아래의 쿼리를 수행해서 새로운 책 데이터를 만들면  
우리가 알 수 있는 정보는 뭐뭐가 있을까요?

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">  
  insert into books (title, author)  
  values (#{title}, #{author})  
</insert>
```

그란데 말입니다



DB 가 AUTO\_INCREMENT 에 의해 자동으로 생성해주는 데이터를 가져오겠다는 설정을 useGeneratedKeys 로 해줍니다!

```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
  insert into books (title, author)
  values (#{title}, #{author})
</insert>
```

그렇게 DB 가 생성해준 데이터를 어느 필드에 넣을 것인지 지정하는 설정 → id 필드에 해당 값을 자동으로 넣어주는 설정

해당 설정을 해주면 insert 구문이 자신이 추가한 데이터 객체를 리턴 할 때 우리가 알 수 없던 id 의 값이 추가 되어서 리턴 됩니다!



```
<insert id="save" useGeneratedKeys="true" keyProperty="id">  
  insert into books (title, author)  
  values (#{title}, #{author})  
</insert>
```



title 과 author 는 이전의 id 와 마찬가지로  
매개변수로서 값을 전달 받아 사용하기 때문에 #{ } 문법을 사용합니다!



그런데 말입니다

그런데 말입니다 우리.. 저 매개변수 전달 했나요?

```
@Mapper 2 usages  Tetz +1
public interface BookMapper {
    public List<BookMybatis> findAll();  Tetz
    public BookMybatis findById(@Param("id") Long id);
    public int save(BookMybatis newBookMybatis);  Tetz
    public int delete(Long id);  Tetz
}
```

id 는 명시적으로 마이바티스에서 제공하는  
@Param 어노테이션으로 전달을 했습니다!

```
@Mapper 2 usages Tetz +1
public interface BookMapper {
    public List<BookMybatis> findAll(); Tetz
    public BookMybatis findById(@Param("id") Long id);
    public int save(BookMybatis newBookMybatis); Tetz
    public int delete(Long id); Tetz
}
```

그런데 save 는 그냥 VO 객체를 전달 했네요?

왜  
이러는  
걸까요?

VO 객체를 전달 했는데  
어떻게 title, author 매개변수를 받죠?



```
<insert id="save" useGeneratedKeys="true" keyProperty="id">
  insert into books (title, author)
  values (#{title}, #{author})
</insert>
```

```
@Data 17 usages Tetz
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class BookMybatis {
    private Long id;
    private String title;
    private String author;
}
```

객체를 전달하면 해당 매개변수의 이름과 동일한 필드의 값을  
Getter 를 사용해서 알아서 가져옵니다!

→ 이 방법이 더 보편적인 방법입니다!  
물론 상황에 따라서 적절히 사용해야 합니다!



```
<delete id="delete">  
    delete from books where id = #{id}  
</delete>
```

id 값은 가변적이므로 매개변수로 받은 id 를 쓰기 위해  
#{ } 문법 사용!



# 실제 MyBatis 가

# 하는 일!



왜  
이러는  
걸까요?

우리가 만든 Mapper 는 인터페이스라서  
구현 코드가 없는데 어떻게 작동하는 걸까요?

```
@Mapper 2 usages  Tetz +1
public interface BookMapper {
    public List<BookMybatis> findAll();  Tetz
    public BookMybatis findById(@Param("id") Long id);
    public int save(BookMybatis newBookMybatis);  Tetz
    public int delete(Long id);  Tetz
}
```



사실은 Mybatis 가 인터페이스를 상속 받아서  
Proxy 형태로 구현 클래스를 만들어서  
적용하기 때문입니다!





@Mapper 2 usages Tetz +1

```
public interface BookMapper {  
    public List<BookMybatis> findAll(); Tetz  
    public BookMybatis findById(@Param("id") Long id);  
    public int save(BookMybatis newBookMybatis); Tetz  
    public int delete(Long id); Tetz  
}
```

```
<mapper namespace="org.example.repository.book.mybatis.BookMapper">  
    <select id="findAll" resultType="BookMybatis">  
        select * from books;  
    </select>  
  
    <select id="findById" resultType="BookMybatis">  
        select * from books where id = #{id};  
    </select>  
  
    <insert id="save" useGeneratedKeys="true" keyProperty="id">  
        insert into books (title, author)  
        values (#{title}, #{author})  
    </insert>  
  
    <delete id="delete">  
        delete from books where id = #{id}  
    </delete>  
</mapper>
```



# BookMybatis

객체 등록



resources  
└─ org  
    └─ example  
        └─ mapper  
        └─ repository  
            └─ book  
application.properties  
log4j.xml  
log4jdbc.log4j2.properties  
mybatis-config.xml

Mybatis 설정을 위해 설정 파일로 갑시다!



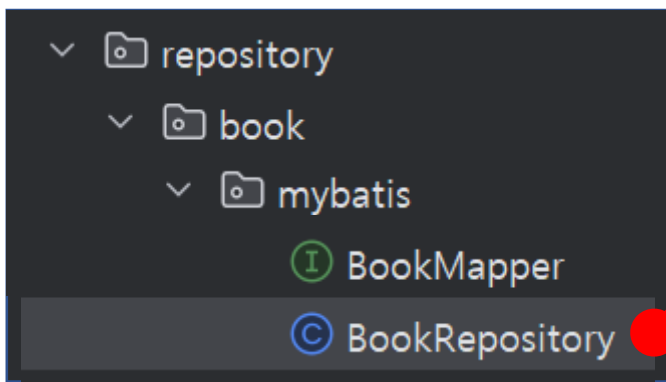
```
<configuration>
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>
  <typeAliases>
    <typeAlias alias="MemberDto" type="org.example.dto.member.MemberDto"/>
    <typeAlias alias="PostDto" type="org.example.dto.post.PostDto"/>
    <typeAlias alias="BoardVO" type="org.example.controller.board.BoardVO"/>
    <typeAlias alias="BookMybatis" type="org.example.domain.BookMybatis"/>
  </typeAliases>
</configuration>
```



도메인에 있는 BookMybatis VO 객체를 등록!

# Repository 만들기





실제 데이터를 처리하는 역할을 하는  
Repository 클래스를 만들어 봅시다!





```

@Repository  Tetz *
@RequiredArgsConstructor
public class BookRepository {
    private final BookMapper bookMapper;

    public List<BookMybatis> findAll() {
        return bookMapper.findAll();
    }

    public BookMybatis findById(Long id) { new *
        return bookMapper.findById(id);
    }

    public int save(BookMybatis newBookMybatis) {
        return bookMapper.save(newBookMybatis);
    }

    public int delete(Long id) { Tetz
        return bookMapper.delete(id);
    }
}

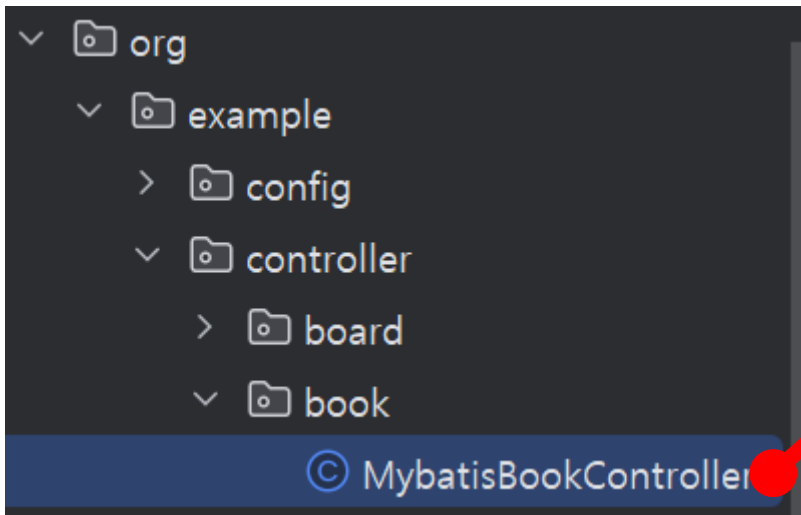
```

사실 Repository 는 역할을 명확하게 하기 위함이고  
실상 코드는 Mybatis 가 구현한 Mapper 가져와서  
그대로 사용하는 코드에 가깝습니다!

→ 물론 상황에 따라서는 별도의 데이터  
처리가 필요한 경우도 많습니다!!



# Controller 만들기



이제 클라이언트의 요청을 받아서 처리할  
MybatisBookController 를 만들어 봅시다!

```
@RestController
```

Tetz +1 \*

```
@Slf4j
```

```
@RequiredArgsConstructor
```

```
@CrossOrigin(origins = "*")
```

```
@RequestMapping(value = "⌚"/book/mybatis")
```

```
public class MybatisBookController {
```

```
    private final BookRepository bookRepository;
```

@RestController 로 작업 할 예정입니다!  
필요한 어노테이션 추가 + 주소는 /book/mybatis 로

데이터 처리를 담당할 Repository Bean 주입 받기



```
@GetMapping(🌐"/show")  👤 Tetz
public ResponseEntity<List<BookMybatis>> findAll() {
    List<BookMybatis> bookMybatis = bookRepository.findAll();
    return ResponseEntity.ok(bookMybatis);
}
```



Book 전체 목록을 받아서  
ResponseEntity 로 응답을 만들어 전달하는 컨트롤러

좀 더 현대적인 형태인 PathVariable 를 사용합시다!

```
@GetMapping("/find/{id}") kdtTetz *  
public ResponseEntity<BookMybatis> findById(@PathVariable Long id) {  
    BookMybatis findBook = bookRepository.findById(id);  
  
    if (findBook == null) return ResponseEntity.notFound().build();  
  
    return ResponseEntity.ok(findBook);  
}
```

전달 받은 id 를 가지는 Book 데이터를 찾고  
해당 데이터가 없으면 404 응답으로 얼리 리턴!

데이터를 찾았으면 얼리 리턴 수행이 안되므로  
찾은 Book 데이터 리턴!



그란데 말입니다

그란데 말입니다  
Query Parameter 와  
Path Variable 의 차이는 뭡까요?

Query Parameter  
/show?id=1  
(주로 필터링, 정렬 등에 사용)

Path Variable  
/show/1  
(주로 특정 리소스를 식별할 때 사용)

타이틀과 저자를 파라미터로 받아서 전달

```
@PostMapping("/save")
```

```
public ResponseEntity<BookMybatis> saveBook(
```

```
    @RequestParam("title") String title,
```

```
    @RequestParam("author") String author
```

```
) {
```

```
    BookMybatis newBook = new BookMybatis(id: null, title, author);
```

```
    int affectedRows = bookRepository.save(newBook);
```

```
    if (affectedRows == 0) {
```

```
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

```
    } else {
```

```
        return ResponseEntity.status(HttpStatus.CREATED).body(newBook);
```

```
    }
```

```
}
```

id 값은 DB 가 자동 생성하므로 null 부여



@PostMapping(🌐✓"/save") 👤 Tetz +1 \*

```
public ResponseEntity<BookMybatis> saveBook(  
    @RequestParam("title") String title,  
    @RequestParam("author") String author
```

```
) {
```

```
    BookMybatis newBook = new BookMybatis(id: null, title, author);
```

```
    int affectedRows = bookRepository.save(newBook);
```

```
    if (affectedRows == 0) {
```

```
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
```

```
    } else {
```

```
        return ResponseEntity.status(HttpStatus.CREATED).body(newBook);
```

```
    }
```

```
}
```

새롭게 생성한 Book 인스턴스를  
전달하여 새로운 데이터 생성

insert 구문도 해당 구문 수행으로 인해  
변화된 데이터의 수(rows)를 리턴하기 때문에  
해당 수치를 기반으로 응답을 변화



@PostMapping(🌐"/save") 👤 Tetz +1 \*

```
public ResponseEntity<BookMybatis> saveBook(
    @RequestParam("title") String title,
    @RequestParam("author") String author
) {
    BookMybatis newBook = new BookMybatis(id: null, title,
    int affectedRows = bookRepository.save(newBook);
    if (affectedRows == 0) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
    } else {
        return ResponseEntity.status(HttpStatus.CREATED).body(newBook);
    }
}
```

생성에 실패했으면 서버 에러 응답 처리

생성에 성공했으면 생성 응답을 만들고  
새롭게 생성된 책 데이터를 Body 에 담아서 전달



그런데 말입니다

저기 NewBook 은 id 가 null 이 아닌가요?  
저걸 돌려주는게 의미가 있나요!?

해당 id 의 값은

Mybatis 의 useGeneratedKeys 옵션에 따라서  
Mybatis 가 DB 에서 자동으로 생성된 id 값을  
해당 객체에 알아서 넣어 줍니다!

따라서, 새롭게 생성된 데이터와 동일한 값을 가지는  
객체를 얻을 수 있습니다!

그런데 말입니다

PathVariable 과 문자열 응답을 위한  
produces 적용



```
@DeleteMapping(value = "/delete/{id}", produces = "text/plain;charset=UTF-8")
public ResponseEntity<String> deleteBook(@PathVariable("id") Long id) {
    int result = bookRepository.delete(id);
    if (result > 0) {
        return ResponseEntity.ok(body: "게시글 삭제 성공");
    } else {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("게시글 삭제 실패");
    }
}
```

삭제 쿼리 결과에 따라  
다른 응답을 전달!



# POSTMAN 으로

# 테스트!

GET



http://localhost:8080/book/mybatis/show

[

```
{
  "id": 1,
  "title": "데미안",
  "author": "헤르만 헤세"
},
{
  "id": 2,
  "title": "인간의 굴레에서",
  "author": "서머셋 모음"
},
{
  "id": 3,
  "title": "참을 수 없는 존재의 가벼움",
  "author": "밀란 쿤데라"
},
{
  "id": 4,
  "title": "삶의 한가운데",
  "author": "루이제 린저"
},
{
  "id": 5,
  "title": "월든",
  "author": "헨리 데이빗 소로우"
},
]
```



Status: 200 OK



GET



http://localhost:8080/book/mybatis/show/1

Body

Cookies

Headers (5)

Test Results



Status: 200 OK



Pretty

Raw

Preview

Visualize

1 {  
2  
3  
4  
5 }

```
"id": 1,  
"title": "데미안",  
"author": "헤르만 헤세"
```



**POST** ⌵ <http://localhost:8080/book/mybatis/save?title=인간 실격&author=다자이 오사무>

**Params** • Authorization Headers (8) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	title	인간 실격
<input checked="" type="checkbox"/>	author	다자이 오사무

**Body** Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize JSON ⌵

```
1 {  
2   "id": 11,  
3   "title": "인간 실격",  
4   "author": "다자이 오사무"  
5 }
```

useGeneratedKeys 에 의해 DB에 의해 자동으로  
생성된 id 값이 입력되어 리턴되는 것을 확인 가능!



Status: 200 OK





DELETE



http://localhost:8080/book/mybatis/delete/11

Body Cookies (1) Headers (8) Test Results



Status: 200 OK

Pretty

Raw

Preview

Visualize

Text



1 게시물 삭제 성공

Body Cookies (1) Headers (8) Test Results



Status: 404 Not Found

Pretty

Raw

Preview

Visualize

Text



1 게시물 삭제 실패



# Spring 과 데이터베이스



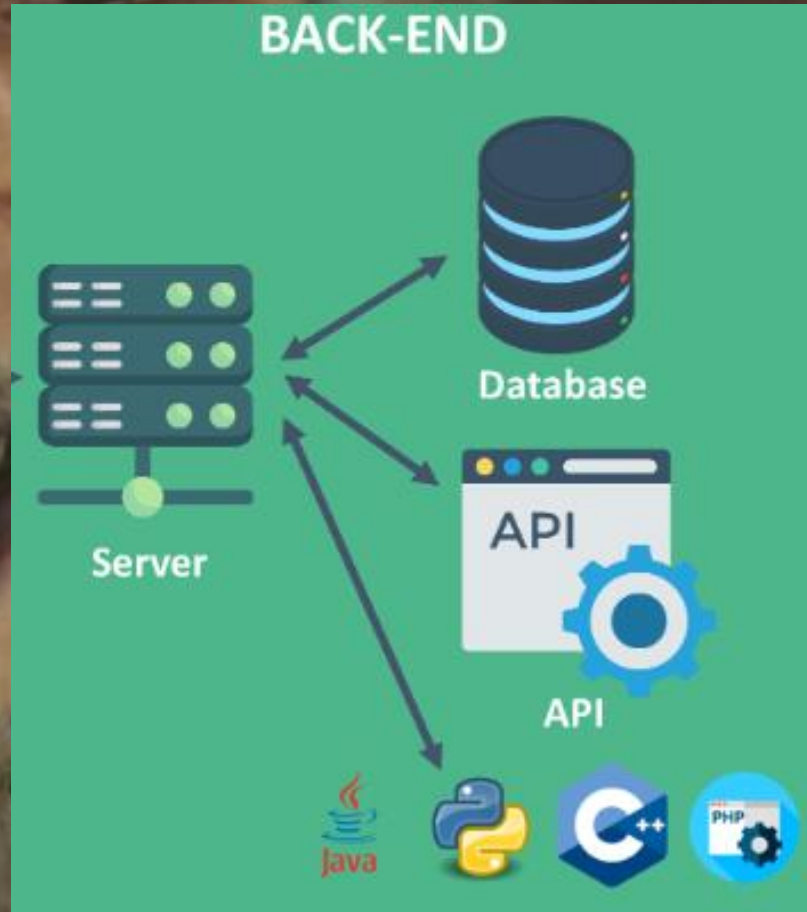
# 백엔드

## 개발자가 하는 일



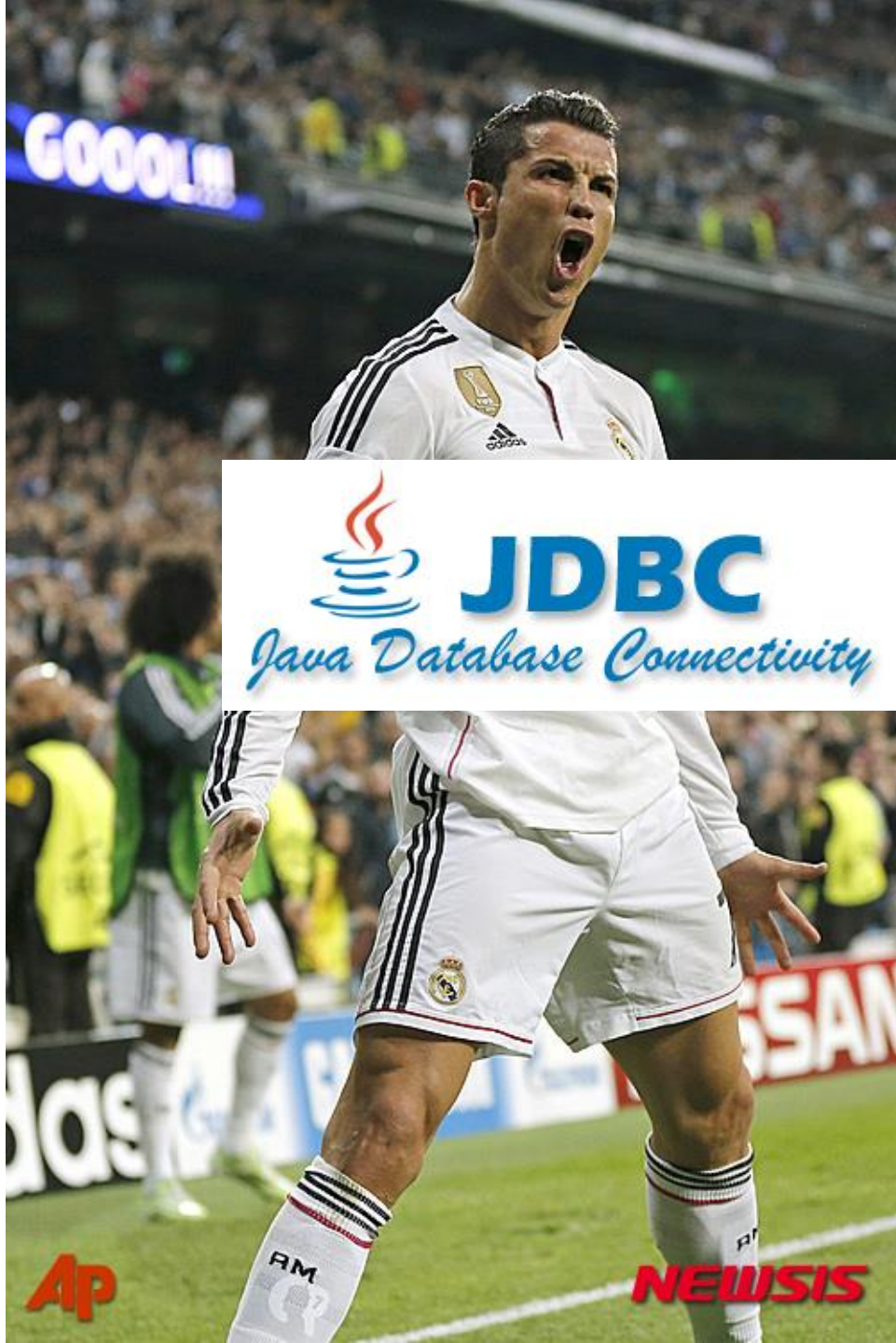


보아라 결국





A long time ago in a galaxy far,  
far away....









자동이요





그란데 말입니다

그래서 탄생한  
ORM 이 뭔지 아시는 분!?

ORM 은  
Object Relational Mapping 의 약자이며

관계형 데이터 베이스와 자바의 객체를  
자동으로 매핑시켜서  
사용자가 직접 매핑 시키지 않고  
관계형 데이터 베이스를 객체로  
사용할 수 있는 기술 입니다!



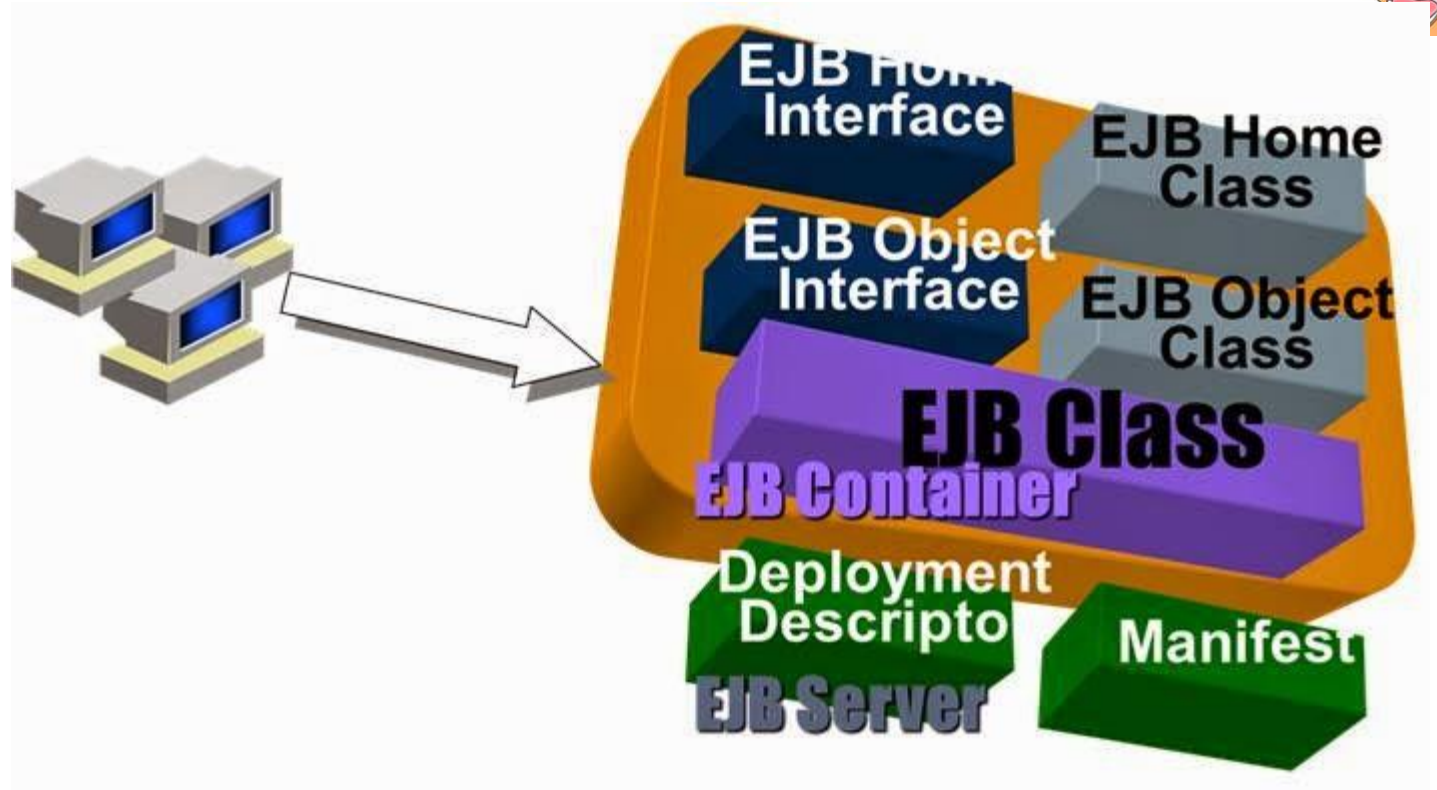
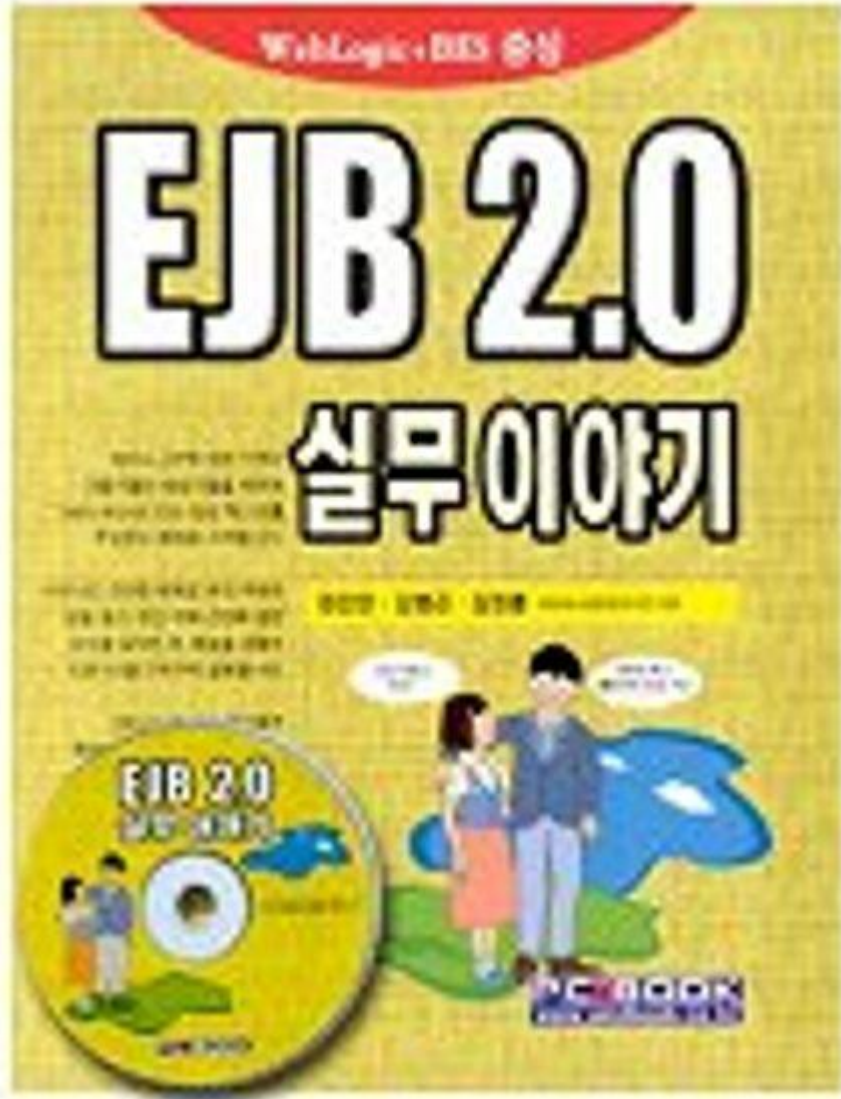
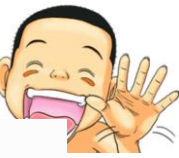


# Object Relational Mapping



-Sabarish Rajamohan

# Enterprise Java Beans 2.0







Garvin King

아놔... EJB2 못써먹겠네  
일단 사용법도 구리고 느리고...  
아오 -\_-+







답답하면 니들이 뛰던지

를 실현한

Garvin King!

답답하면 니들이 뛰던지

어떻게 이름도 000지?





# HIBERNATE







혹시 Hibernate 뜻 아시는 분!?



그란데 말입니다







909 June 24, 2019

early summer



909 September 17, 2019

early autumn

Image source: Bears of Brooks River 2019: A Guide to Their Identification, Lives, and Habits (e-book); <https://www.nps.gov/katm/learn/photosmultimedia/ebook>









그래서  
2006 년!





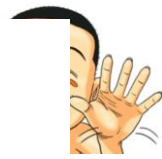


# JPA

Java Persistence API



HIBERNATE



HIBERNATE

JAVA  
애플리케이션

JPA

JDBC  
API

SQL →

DB

← 결과 반환



자동이요



어! 이게 진짜 되네? 우와..





# JPA의 구조



일단 JPA 는 DB 테이블을 읽어서  
자신만의 Map 컬렉션을 만들어서 등록 합니다!

JAVA  
애플리케이션

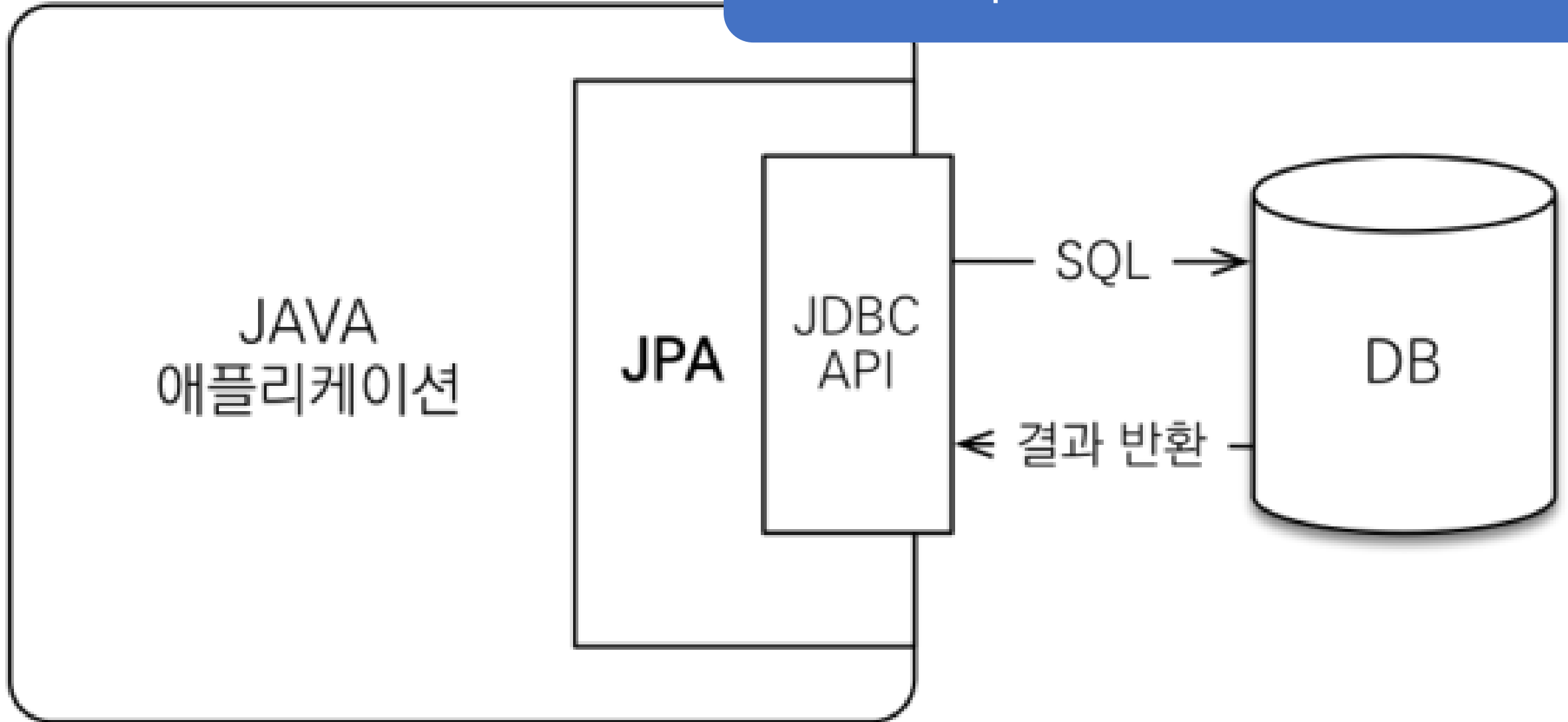
JPA

JDBC  
API

SQL →

DB

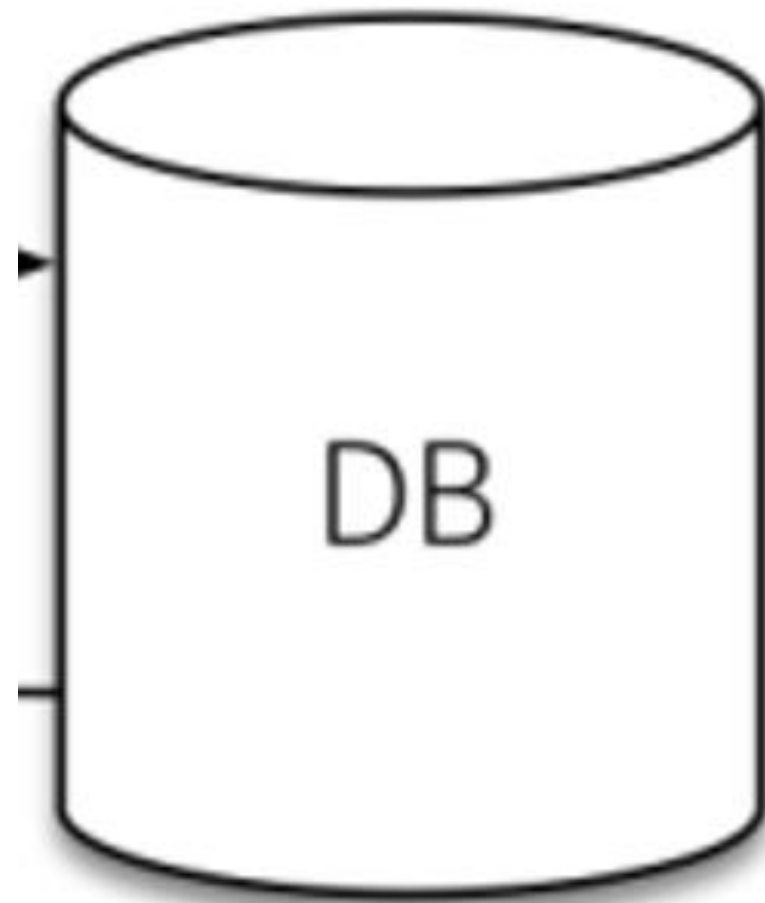
← 결과 반환



# JPA



일단 JPA 는 DB 테이블을 읽어서  
자신만의 Map 컬렉션을 만들어서 등록 합니다!



# JPA

HIBERNATE



## Entity Map



### Posts Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### Books Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### User Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### Todos Table

1. 데이터1
2. 데이터2
3. 데이터 3...



그란데 말입니다

아까 Garvin King 이 EJB2 의 성능이  
구리다고 했었는데요!

JPA 는 성능을 위해서 Entity Map 을  
어디에 위치 시킬까요!?

성능을 위해 Memory 에 위치 시킵니다!





**그때 그때 달라요**



# JPA의 구조

- Read



JPA는 DB 와 동일한 엔티티 맵을 가지고 있는데요!  
그럼 데이터를 조회할 때 다시 DB와 통신할 필요가 있을까요!?



## Entity Map

### Posts Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### Books Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### User Table

1. 데이터1
2. 데이터2
3. 데이터 3...

### Todos Table

1. 데이터1
2. 데이터2
3. 데이터 3...

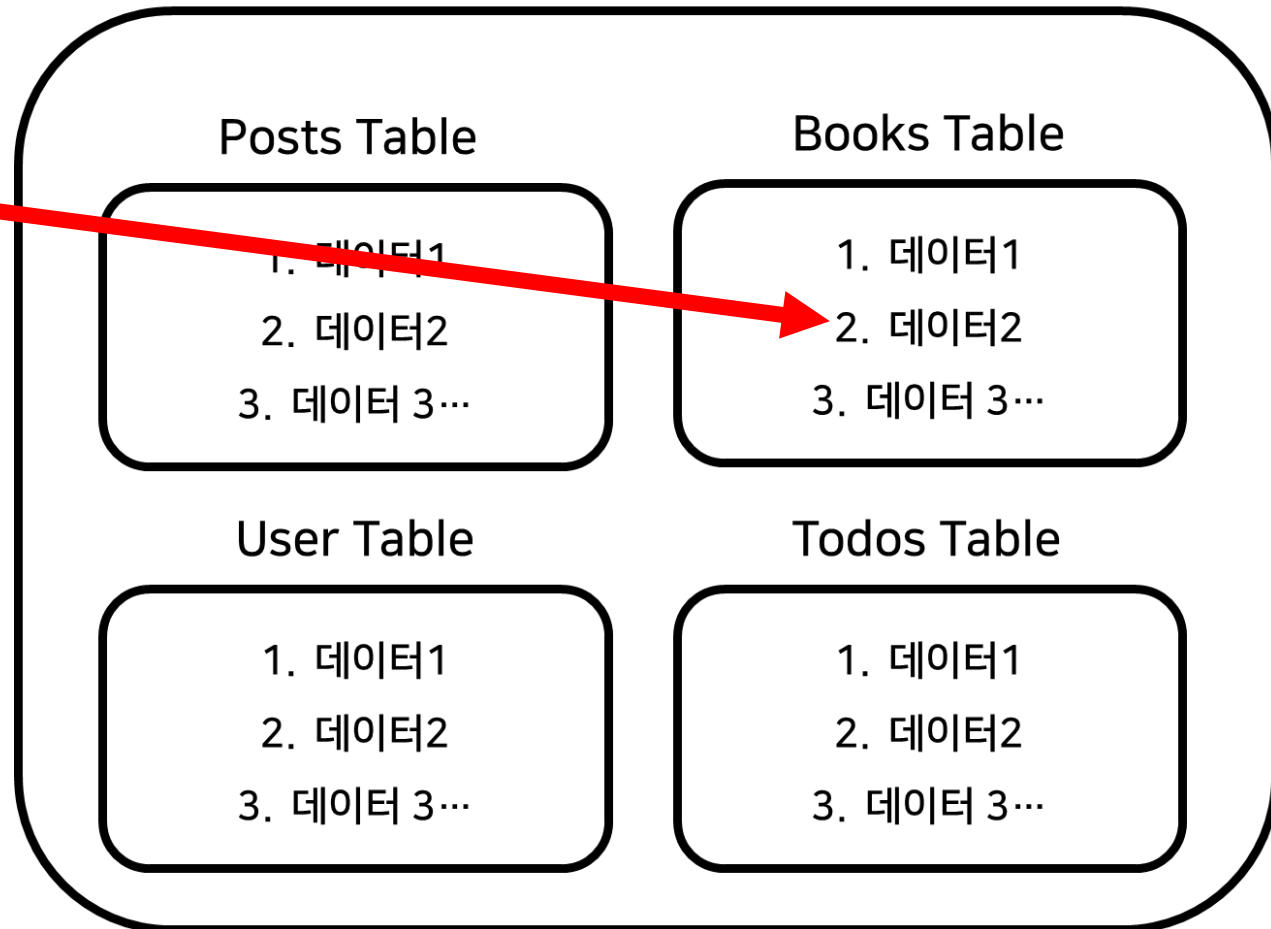




필요가 없습니다!  
바로 자신이 가진 Entity Map 에서 데이터를 가져오면 더 효율적이죠!!



## Entity Map





# JPA의 구조

- Create, Update,  
Delete



그런데 말입니다

JPA는 DB 와 동일한 엔티티 맵을 가지고 있네요!?

그럼 데이터를 추가, 삭제, 수정 할 때마다  
DB 와 통신할 필요가 있을까요!?

하나의 동작에서 데이터를 동시에 추가, 삭제, 수정을  
할 수도 있기 때문에 JPA 는 하나의 동작인  
Transaction 을 기준으로 모든 동작은  
JPA 의 Entity Map 에서만 일어나고  
Transaction 이 끝나면 그 때  
Entity Map 의 변경 내용을 DB에 반영 합니다!



JPA  
 HIBERNATE

참... 좋다!

# JPA 세팅하기!





# build.gradle

## 설정



```
// JPA
implementation 'org.springframework:spring-orm:5.3.29' // Spring ORM 의존성
implementation 'javax.persistence:javax.persistence-api:2.2' // JPA API
implementation 'org.springframework.data:spring-data-jpa:2.5.4'
implementation 'org.hibernate:hibernate-core:5.5.6.Final'
implementation 'mysql:mysql-connector-java:8.0.33'
```

아까 들었었던 Hibernate 라던지, persistence 라던지  
등등의 라이브러리가 추가 됩니다!

# JpaConfig 적용







Jpa 설정을 위해 JpaConfig 를 추가해 봅시다!

org  
example  
config  
© JpaConfig

© RootConfig



© ServletConfig

© WebConfig

스프링에 설정 Bean 으로 등록



스프링에서 @Transactional 어노테이션을 활성화  
→ sql 구문을 2개 이상 실행할 때 사용합니다!

```
@Configuration  xenosign +1  
@EnableTransactionManagement   
@PropertySource("classpath:application.properties")  
@EnableJpaRepositories(basePackages = "org.example.domain")  
public class JpaConfig {
```

설정 파일을 어디서 읽어오는지 설정!



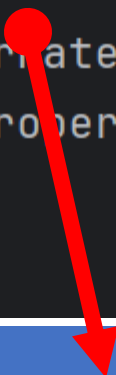
```
@Configuration  @ xenosign +1
@EnableTransactionManagement
@PropertySource("classpath:application.properties")
@EnableJpaRepositories(basePackages = "org.example.domain")
public class JpaConfig {
```

중요!!!!!!

JPA 가 자동으로 매핑 시킬 객체를 어느 패키지에서  
찾을지 지정하는 옵션입니다!!

여러분의 패키지 이름.domain 으로 반드시 지정 필요!

```
public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {  
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();  
    emf.setDataSource(dataSource);  
    emf.setPackagesToScan("org.example.domain");  
    emf.setJpaVendorAdapter(new HibernateJpaVendorAdapter());  
    emf.setJpaProperties(additionalProperties());  
    return emf;  
}
```



로컬에 있는 데이터 베이스를 자동으로 스캔한 다음  
지정한 패키지 하위에 존재하는 객체와 매핑 시키는  
옵션에 대한 내용입니다!



```
@Bean  👤 kdtTetz +1  
public PlatformTransactionManager transactionManager(EntityManagerFactory emf) {  
    return new JpaTransactionManager(emf);  
}
```



JPA 추상화를 담당하는 PlatformTransactionManager  
관련 설정입니다!

```
@Value("org.hibernate.dialect.MySQL8Dialect")
private String dialect;
@Value("update")
private String hbm2ddlAuto;
@Value("true")
private String hibernateShowSql;
@Value("true")
private String formatSql;
```

JPA가 어떤 데이터 베이스를 읽을지  
+ 읽을 때 필요한 라이브러리 설정 입니다!

JPA 수행 시, 관련 로그를 보기 위한 설정

```
private Properties additionalProperties() { 1 usage  xenosign
    Properties properties = new Properties();
    properties.setProperty("hibernate.dialect", dialect);
    properties.setProperty("hibernate.hbm2ddl.auto", hbm2ddlAuto);
    properties.setProperty("hibernate.show_sql", hibernateShowSql);
    properties.setProperty("hibernate.format_sql", formatSql);
    return properties;
}
```

# application.properties



## 적용



<https://github.com/xenosign/spring-code-repo/blob/main/jpa.properties>



```
# JPA 설정  
hibernate.dialect=org.hibernate.dialect.MySQL8Dialect  
hibernate.hbm2ddl.auto=update  
hibernate.show_sql=true  
hibernate.format_sql=true
```



그런데 말입니다

스프링 Boot 에서 JPA 설정은 어떻게 하면 될까요?

gradle 에 starter 팩(라이브러리 모아 둔 것)  
한 줄을 추가하고

application.properties 에 가서  
자신이 원하는 옵션만 켜주면  
Spring boot 가 알아서 적용 시켜 줍니다!

그런데 말입니다

# build.gradle 설정



```
// 스프링 부트 일때  
implementation 'org.springframework.boot:spring-boot-starter:2.2.0'
```

## application.properties 설정

```
# DB ??  
jdbc.driver=com.mysql.cj.jdbc.Driver  
jdbc.url=jdbc:mysql://localhost:3306/mybatis  
jdbc.username=root  
jdbc.password=1234  
  
# JPA 설정  
hibernate.dialect=org.hibernate.dialect.MySQL8Dialect  
hibernate.hbm2ddl.auto=update  
hibernate.show_sql=true  
hibernate.format_sql=true
```



이제야한다



spring



ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ

ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ

ㅋㅋ



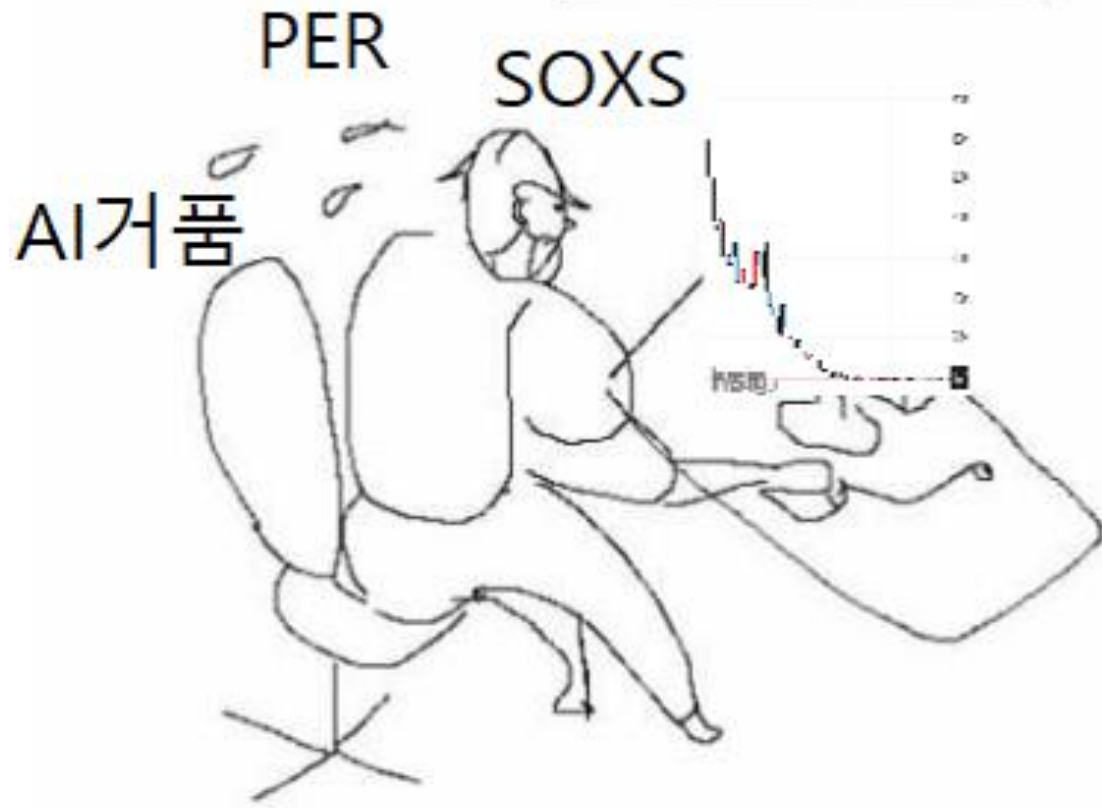
spring  
boot



떨림



벌어야한다



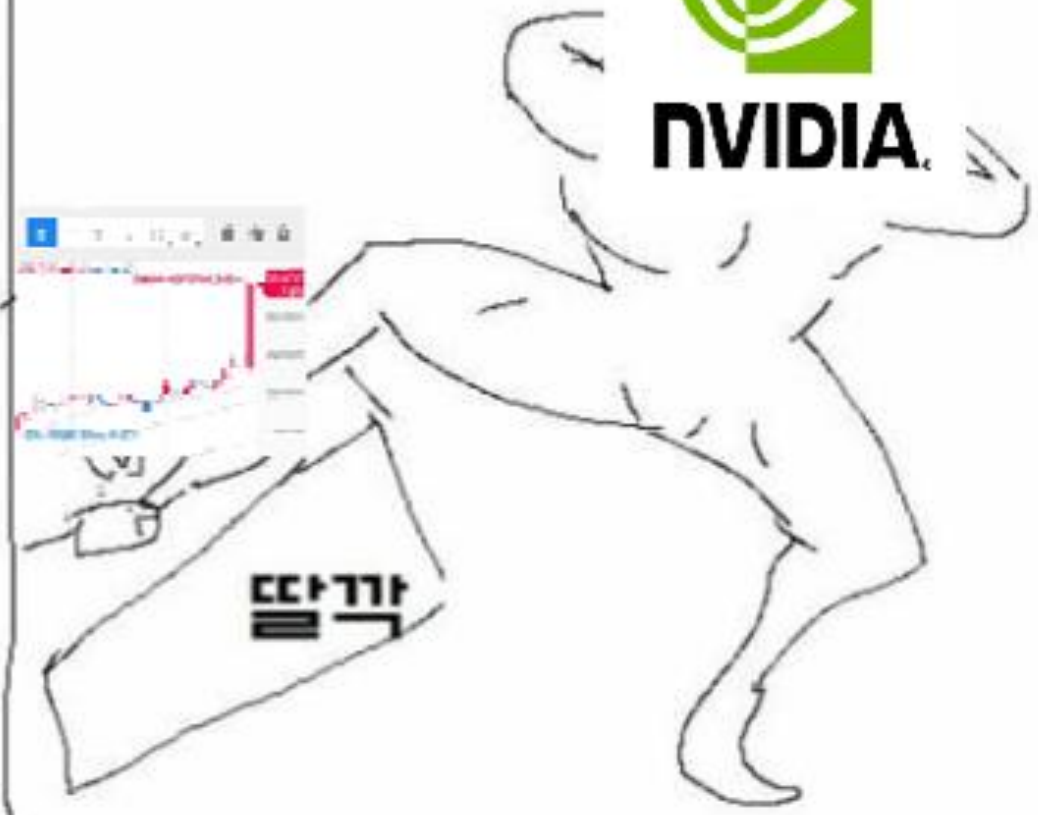
ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ  
ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ  
ㅋㅋ




NVIDIA

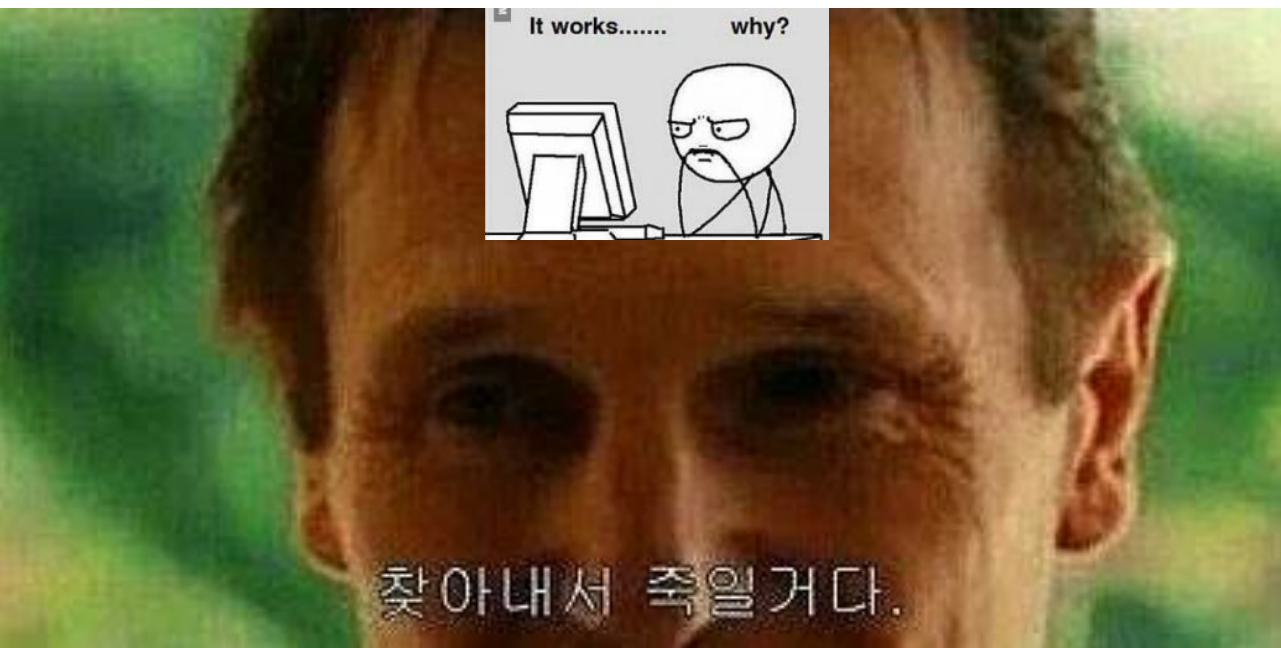


떨깽





인간의 욕심은 끝이없고



찾아내서 죽일거다.





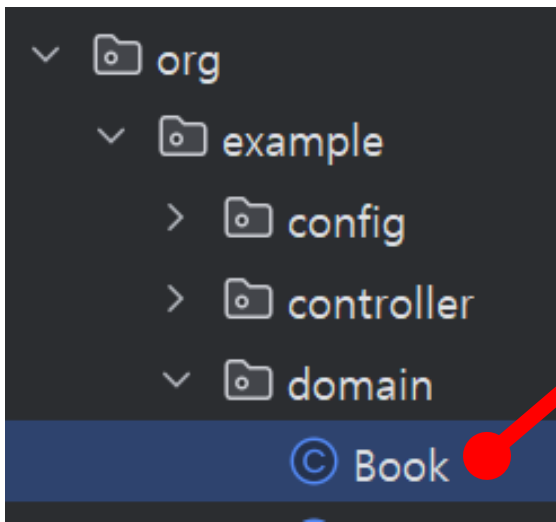
# JPA

# Entity 만들기





도메인 패키지에 Book Entity 를 추가합니다!





그란데 말입니다

Entity 의 의미가 뭐죠!?

데이터 베이스 테이블과  
정확하게 매핑되는 JAVA 객체 입니다!

그란데 말입니다





	id	title	author
▶	1	데미안	헤르만 헤세
	2	인간의 굴레에서	서머싯 몸
	3	참을 수 없는 존재의 가벼움	밀란 쿤데라
	4	삶의 한가운데	루이제 린저
	5	열든	헨리 데이빗 소로우
	NULL	NULL	NULL

우리가 가지고 있는 요 테이블과  
정확하게 매핑되는 객체를 만들어야 합니다!

@Data 22 usages Tetz \*

@Entity

@Table(name="books")

public class Book {

private Long id;

private String title;

private String author;

JPA 가 클래스 이름과 동일한 테이블을  
자동으로 매핑 시켜 주지만 이름이 다른 경우라면  
요렇게 직접 지정할 수 있습니다!

클래스 명이 Books 라면 자동으로 매핑 됩니다!

컬럼과 동일한 멤버 필드를 선언

@Id 어노테이션으로 해당 필드가  
PK 역할을 하는 것을 알려 줍니다!

PK 는 자동 생성이 되므로  
@GeneratedValue 어노테이션으로  
DB 에서 해당 값을 받아오기로 설정

▼ @Data 22 usages Tetz

@Entity

@Table(name="books")

public class Book {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    @Column(name = "title")

    private String title;

    @Column(name = "author")

    private String author;



▼ @Data 22 usages Tetz

@Entity

@Table(name="books")

public class Book {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "title")

private String title;

@Column(name = "author")

private String author;

역시나 JPA 가 자동으로 매핑을 시켜 주지만  
@Column 어노테이션으로  
직접 매핑도 가능합니다!

JPA 에 의해 매핑이 되고 있다는 아이콘들!



```
public Book() {  👤 Tetz  
}
```

```
public Book(Long id, String title, String author) {  
    this.id = id;  
    this.title = title;  
    this.author = author;  
}
```

JPA 에서 필요하기 때문에

빈 생성자와, 모든 필드 생성자를  
반드시 생성해야 합니다!





그란데 말입니다

그란데 말입니다

JPA 를 쓰면 Mybatis 에서  
반드시 해야할 일이 없어집니다!

어떤 일이 없어질까요?

객체와 테이블 매핑 작업을 안해도 됩니다!  
+ 기본적인 SQL 쿼리는 자동으로 제공 됩니다!





짱이다..너무 짱인데..  
진짜 짱이다..오..  
완전 캡짱이다..





# JPA

# Repository 만들기

repository

book

jpa

© JpaBookRepository

mybatis

매핑이 필요 없기 때문에  
바로 Repository 를 만듭니다!!



@Slf4j    kdtTetz

@RequiredArgsConstructor

@Transactional

@Repository

```
public class JpaBookRepository {  
    private final EntityManager em;
```


필요한 어노테이션을 추가해 줍니다!

@Transactional 은 해당 클래스에서  
2번 이상의 SQL 쿼리가 수행 될 수 있도록  
설정하는 어노테이션 입니다!

실제적으로 DB 통신 및 쿼리를 수행하는  
EntityManager 을 주입 받습니다!

JPA 는 자신만의 쿼리문인 JPQL(Java Persistence Query Language) 을 사용!

JPA 는 다양한 Entity를 관리하므로 어떤 Entity 를 사용할지 명시하고 쿼리를 작성해야 합니다!



```
public List<Book> findAll() {  
    String jpql = "select b from Book b";  
    List<Book> bookList = em.createQuery(jpql, Book.class).getResultList();  
    return bookList;  
}
```



```
public List<Book> findAll() {  👤 kdtTetz  
    String jpql = "select b from Book b";  
    List<Book> bookList = em.createQuery(jpql, Book.class).getResultList();  
    return bookList;  
}
```

작성한 jpql 구문과, Book 엔티티의 클래스를  
EntityManager 에게 전달하여 쿼리를 수행하고  
빌더 패턴을 사용하여 해당 쿼리의 결과를 받아옵니다!

쿼리 수행 결과 리스트를 리턴





```
public Book save(Book book) {  
    em.persist(book);  
    return book;  
}
```

저장은 정말 간단합니다!

Entity 가 테이블과 정확하게 매핑되어 있기 때문에  
해당 Entity 에 맞는 객체를 생성해서 저장 시키면 끝입니다!

persist(= 영속 시키다) → JPA 가 관리하는 객체를  
영속 시키다 → 데이터 베이스에 저장하다

저장한 데이터를 EntityManager 가 알아서 가져오기 때문에  
바로 리턴 시키면 됩니다!

Like, 마이바티스의 useGeneratedKeys

```
public void delete(Long id) {  👤 kdtTetz *  
    Book book = em.find(Book.class, id);  
    if (book != null) em.remove(book);  
}
```

삭제는 데이터 베이스에서 직접 삭제한다는  
개념이라기 보다는

JPA 에 관리하는 데이터베이스 Map 에서  
해당 데이터를 찾아서 먼저 삭제를 한 다음  
JPA 의 Map 과 데이터베이스의 테이블을  
나중에 일치시키는 방향으로 진행이 됩니다

따라서 데이터 삭제를 할 때는 id 가 아닌  
해당 객체를 직접 보내줘야 합니다



# JPA

# Controller 만들기!

controller

board

book

JpaBookController

Jpa 를 사용하는 JpaBookController 를 만들어 봅시다!

@Controller kdtTetz

@Slf4j

@RequiredArgsConstructor

@Transactional

@RequestMapping("/book/jpa")

public class JpaBookController {

private final JpaBookRepository jpaBookRepository;

필요 어노테이션을 추가해 줍시다!

데이터 처리를 담당하는  
JpaBookRepository 주입



```
@GetMapping(🌐"/show")  👤 kdtTetz  
public ResponseEntity<List<Book>> findAll() {  
    List<Book> bookList = jpaBookRepository.findAll();  
    return ResponseEntity.ok(bookList);  
}
```

사실 컨트롤러는 코드가 거의 비슷합니다!

JpaRepository 에서 필요 데이터를 받아서  
해당 데이터를 ResponseEntity 로  
응답을 만들어서 돌려 줍니다!



```
@GetMapping(🌐"/show/{id}")  👤 kdtTetz
public ResponseEntity<Book> findById(@PathVariable Long id) {
    Book findBook = jpaBookRepository.findById(id);
    if(findBook == null) return ResponseEntity.notFound().build();
    return ResponseEntity.ok(findBook);
}
```

동일하게 id 를 PathVariable 로 전달 받아서  
해당 id 가 존재하는지 확인하고  
결과에 따라서 다른 응답을 내보내기

title 과 author 를 파라미터로 전달 받아서  
새로운 글을 등록합니다!



```
@PostMapping(🌐"/save")  👤 kdtTetz *  
public ResponseEntity<Book> save(  
    @RequestParam("title") String title,  
    @RequestParam("author") String author  
) {  
    Book newBook = new Book(id: null, title, author);  
    Book addedBook = jpaBookRepository.save(newBook);  
    if(addedBook == null) return ResponseEntity.internalServerError().build();  
    return ResponseEntity.ok(newBook);  
}
```

JPA 는 데이터 추가가 성공하면  
성공한 객체 자체를 돌려주기 때문에  
성공 여부는 객체가 null 인가 아닌가로 체크





```
@PostMapping(🌐"/save")  👤 kdtTetz *  
public ResponseEntity<Book> save(  
    @RequestParam("title") String title,  
    @RequestParam("author") String author  
) {  
    Book newBook = new Book(id: null, title, author);  
    Book addedBook = jpaBookRepository.save(newBook);  
    if(addedBook == null) return ResponseEntity.internalServerError().build();  
    return ResponseEntity.ok(newBook);  
}
```

성공 여부에 따라서 다른 응답을 전달하기!



```
@DeleteMapping("/{delete/{id}}") kdtTetz *
public ResponseEntity<Book> delete(@PathVariable Long id) {
    Book book = jpaBookRepository.findById(id);
    if(book == null) return ResponseEntity.notFound().build();

    jpaBookRepository.delete(id);
    return ResponseEntity.ok(book);
}
```

Repository 에 데이터를 못 찾는 경우에  
처리가 안되어 있으므로 직접 찾아서  
데이터가 없을 경우 Not\_Found 응답 처리

데이터가 있을 경우 삭제 처리 후  
삭제된 데이터를 리턴!



# POSTMAN

# 확인

GET

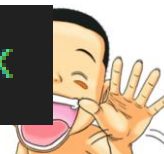


http://localhost:8080/book/jpa/show

```
[
  {
    "id": 1,
    "title": "데미안",
    "author": "헤르만 헤세"
  },
  {
    "id": 2,
    "title": "인간의 굴레에서",
    "author": "서머셋 모م"
  },
  {
    "id": 3,
    "title": "참을 수 없는 존재의 가벼움",
    "author": "밀란 쿤데라"
  },
  {
    "id": 4,
    "title": "삶의 한가운데",
    "author": "루이제 린저"
  },
  {
    "id": 5,
    "title": "월든",
    "author": "헨리 데이빗 소로우"
  },
]
```



Status: 200 OK



GET



http://localhost:8080/book/jpa/show/1

Body

Cookies

Headers (5)

Test Results



Status: 200 OK



Pretty

Raw

Preview

Visualize

1



{

2

"id": 1,

3

"title": "데미안",

4

"author": "헤르만 헤세 "

5

}



POST ▼ http://localhost:8080/book/jpa/save?title=그리스인 조르바&author=니코스 카잔차키스

Params ● Authorization Headers (8) Body Scripts Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value
<input checked="" type="checkbox"/>	title	그리스인 조르바
<input checked="" type="checkbox"/>	author	니코스 카잔차키스

Body Cookies (1) Headers (5) Test Results

Pretty Raw Preview Visualize JSON ▼

```
1 {  
2   "id": 7,  
3   "title": "그리스인 조르바",  
4   "author": "니코스 카잔차키스"  
5 }
```

JPA의 EntityManager 가 알아서 다 넣어 줍니다!



Status: 200 OK



DELETE



http://localhost:8080/book/jpa/delete/7

Body Cookies (1) Headers (5) Test Results



Status: 200 OK

Pretty

Raw

Preview

Visualize

JSON



1



2

"id": 7,

3

"title": "그리스인 조르바",

4

"author": "니코스 카잔차키스"

5



Body Cookies (1) Headers (4) Test Results



Status: 404 Not Found

Pretty

Raw

Preview

Visualize

Text



1

