JPA

이영석 이준열 임수민

index

1. SQL 중심 개발의 문제점

> 코드 양이 많고 SQL 작성 시 오타가 많다.

> 진정한 의미의 계층 분할이 어렵다.

> 패러다임의 불일치

2. ORM과 JPA

> ORM

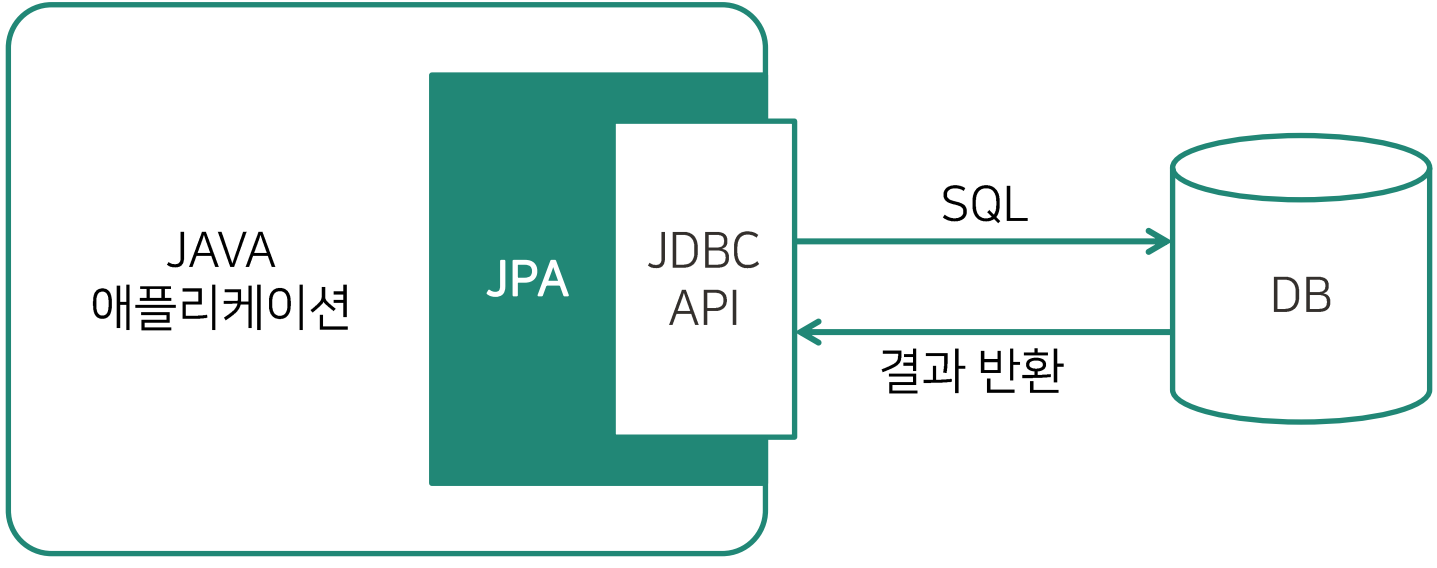
- Object-relational Mapping
- 객체와 관계형 데이터베이스를 매핑

> JPA

- Java Persistence API
- Java 진영의 ORM 표준

2. ORM과 JPA

> JPA 데이터 흐름도



3. Spring Data JPA

> 지루하게 반복되는 CRUD 문제를 세련된 방법으로 해결

> 개발자는 인터페이스만 작성

> Spring Data JPA가 구현 객체를 동적으로 생성해서 주입

4. 구현 (Maven 설정)

```
<!-- jpa -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.2.10.Final</version>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>1.11.6.RELEASE</version>
</dependency>
```

4. 구현 (Java config 설정)

```java
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
    emf.setDataSource(dataSource());
    emf.setPackagesToScan("com.yg.reservation.domain");

    HibernateJpaVendorAdapter jpaVendorAdapterendorAdapter = new HibernateJpaVendorAdapter();
    jpaVendorAdapterendorAdapter.setShowSql(true);
    emf.setJpaVendorAdapter(jpaVendorAdapterendorAdapter);
    emf.setJpaProperties(additionalProperties());

    return emf;
}

Properties additionalProperties() {
    Properties properties = new Properties();
    properties.setProperty("hibernate.dialect",
            "org.hibernate.dialect.MySQL5Dialect");
    return properties;
}
```

4. 구현 (테이블 & Domain 클래스 – 적용 전)

id	INT(11)
user_id	INT(11)
file_name	VARCHAR(255)
save_file_name	VARCHAR(4000)
file_length	INT(11)
content_type	VARCHAR(255)
delete_flag	INT(1)
create_date	DATETIME
modify_date	DATETIME

<DB 테이블>

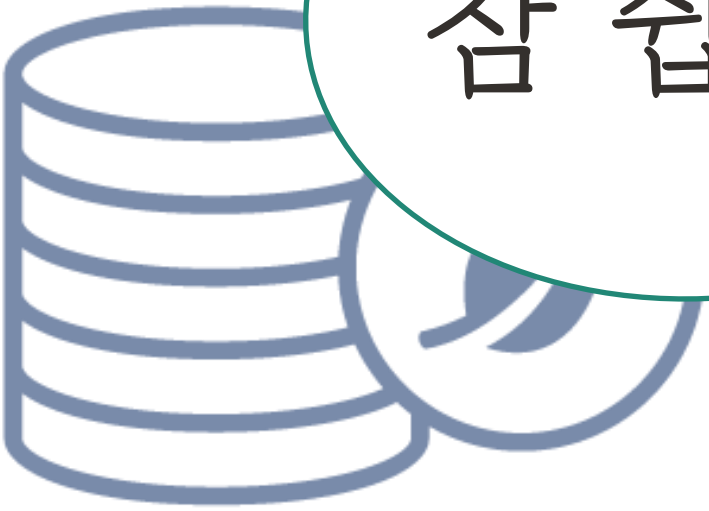
```
public class Image {
    private int id;
    private int userId;
    private String fileName;
    private String saveFileName;
    private long fileLength;
    private String contentType;
    private int deleteFlag;
}
```

<클래스>

4. 구현 (Entity class @ - 적용 후)

```java
@Entity
@Table(name = "files", indexes = {
        @Index(name = "IMAGE_IDX_0", columnList = "user_id", unique = false) })
public class Image {
    @Id
    @GeneratedValue(generator = "native")
    @GenericGenerator(name = "native", strategy = "native")
    private int id;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id")
    @JsonIgnore
    private User user;
    @Column(name = "file_name")
    private String fileName;
    @Column(name = "save_file_name", length = 4000)
    private String saveFileName;
    @Column(name = "file_length")
    private long fileLength;
    @Column(name = "content_type")
    private String contentType;
    @Column(name = "delete_flag")
    private int deleteFlag;
    @Column(name = "create_date", nullable = false, updatable = false, insertable = false,
            columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
    @Temporal(TemporalType.TIMESTAMP)
    private Date createDate;
    @Column(name = "modify_date", nullable = false, updatable = false, insertable = false,
            columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP")
    @Temporal(TemporalType.TIMESTAMP)
    private Date modifyDate;
}
```

4. 구현 (Entit



4. 구현 (Entity class @ - Table)

```
@Entity
@Table(name = "files", indexes = {
        @Index(name = "IMAGE_IDX_0", columnList = "user_id", unique = false) })
public class Image {
```

4. 구현 (Entity class @ - 기본 키 매핑)

```
@Id
@GeneratedValue(generator = "native")
@GenericGenerator(name = "native", strategy = "native")
private int id;
```

4. 구현 (Entity class @ - 필드 매핑)

```
@Column(name = "file_name")
private String fileName;
@Column(name = "save_file_name", length = 4000)
private String saveFileName;
@Column(name = "file_length")
private long fileLength;
@Column(name = "content_type")
private String contentType;
@Column(name = "delete_flag")
private int deleteFlag;
```

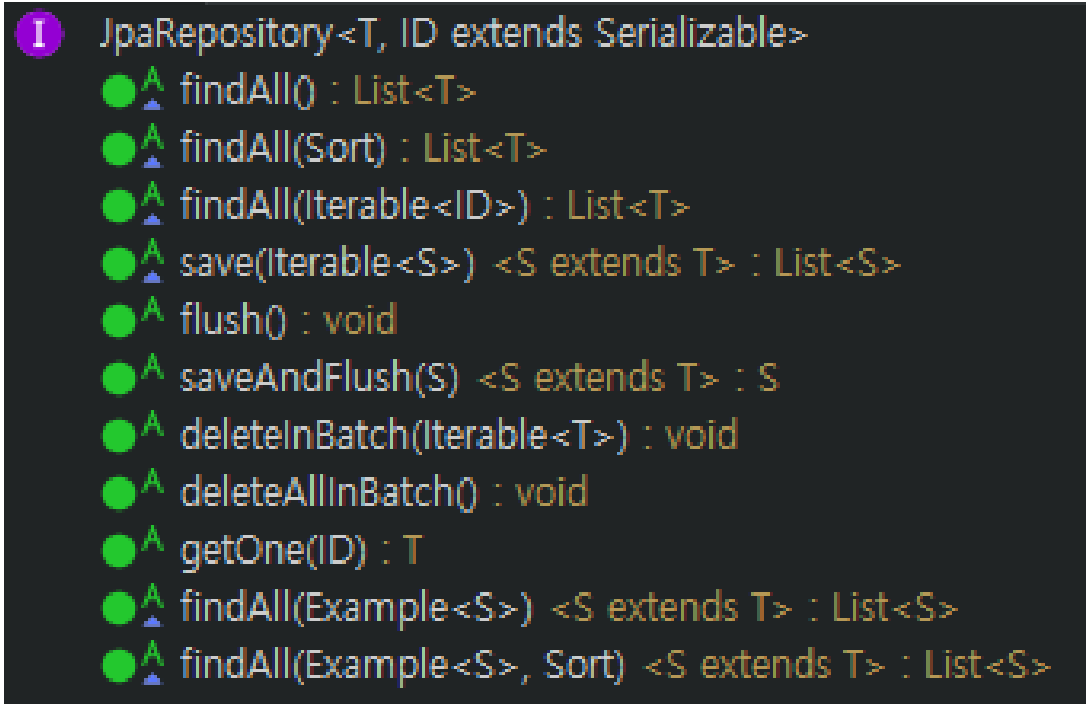
4. 구현 (Entity class @ - 필드 매핑)

```
@Column(name = "create_date", nullable = false, updatable = false, insertable = false,
        columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
@Temporal(TemporalType.TIMESTAMP)
private Date createDate;
@Column(name = "modify_date", nullable = false, updatable = false, insertable = false,
        columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP")
@Temporal(TemporalType.TIMESTAMP)
private Date modifyDate;
```

4. 구현 (Entity class @ - 외래 키 매핑)

```java
@ManyToOne(fetch=FetchType.LAZY)
@JoinColumn(name = "user_id")
private User user;
```

4. 구현 (Query Method - JPA Repository)



4. 구현 (Query Method - Keywords)

Find	By	In
TopN	OrderBy	Desc

4. 구현 (Query Method - Image Repository)

```java
@Repository
public interface ImageRepository extends JpaRepository<Image, Integer> {
    public List<Image> findByIdIn(List<Integer> ids);
}
```

4. 구현 (SELECT → findOne)

```
public static final String SELECT =
        "SELECT id, user_id, file_name, save_file_name, file_length, content_type, delete_flag "
        + "FROM files "
        + "WHERE id=:id";
```

↓

```
Image image = imageRepository.findOne(id);
```

4. 구현 (UPDATE → find & setValue)

```
public static final String UPDATE_DELETE_FLAG_TO_0 =
        "UPDATE files SET delete_flag=0 WHERE id IN(:ids)";
```

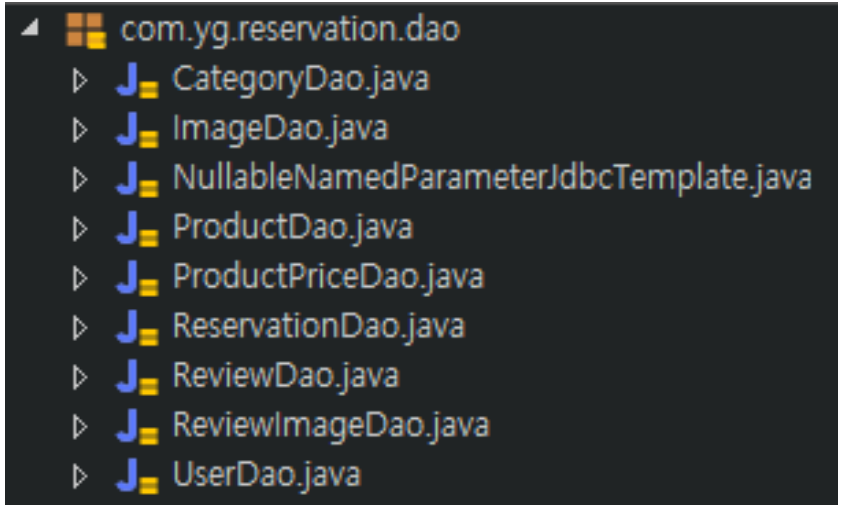
↓

```
List<Image> images = imageRepository.findByIdIn(imageIds);
images.forEach(i -> i.setDeleteFlag(0));
```

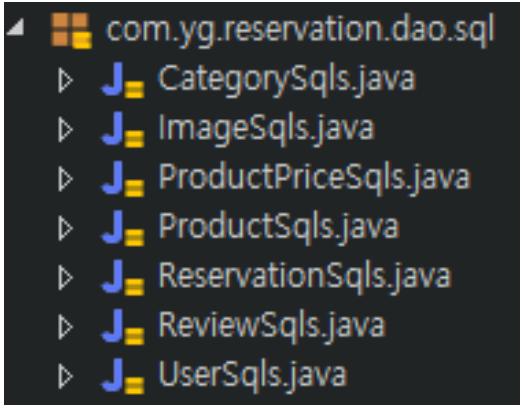
4. 구현 (Paging 처리)

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Integer> {
    public List<Product> findTop5ByOrderByIdDesc();
    public List<Product> findByCategoryId(int categoryId, Pageable pageable);
}
```

5. 비교 (클래스 – 적용 전)

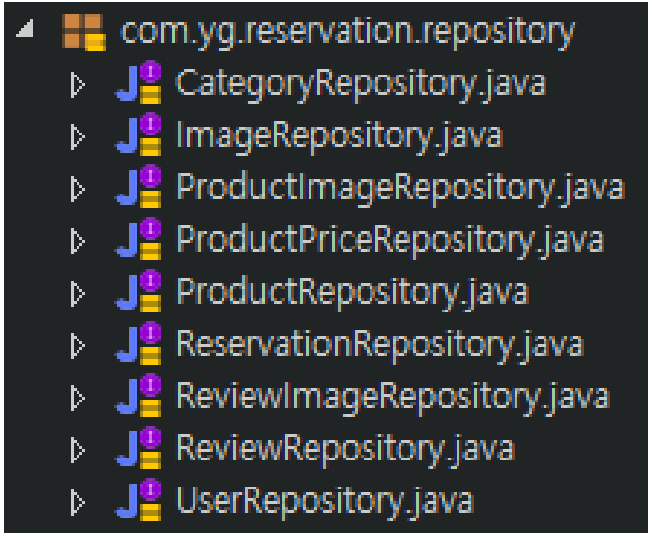


<DAO>



<SQL>

5. 비교 (클래스 – 적용 후)



<Repository>

5. 비교 (LoC)

540 80% ↓ ⇨ 108

6. 정리

> 생산성, 편의성 향상

> JPA에 대한 정확한 이해 및 설정 필수

> 효율적인 쿼리 사용을 위해서는 JPQL, QueryDSL 사용

END

감사합니다

Q&A