

# 스프링 부트 웹 프로젝트

chapter03

## MariaDB와 Spring Data JPA

제공된 자료는 훈련생의 수업을 돕기 위한 것으로, 타인과 공유하시면 안됩니다.

# Contents

part.1

프로젝트 준비

part.2

Spring Data JPA

part.3

엔티티 클래스와 JpaRepository

part.4

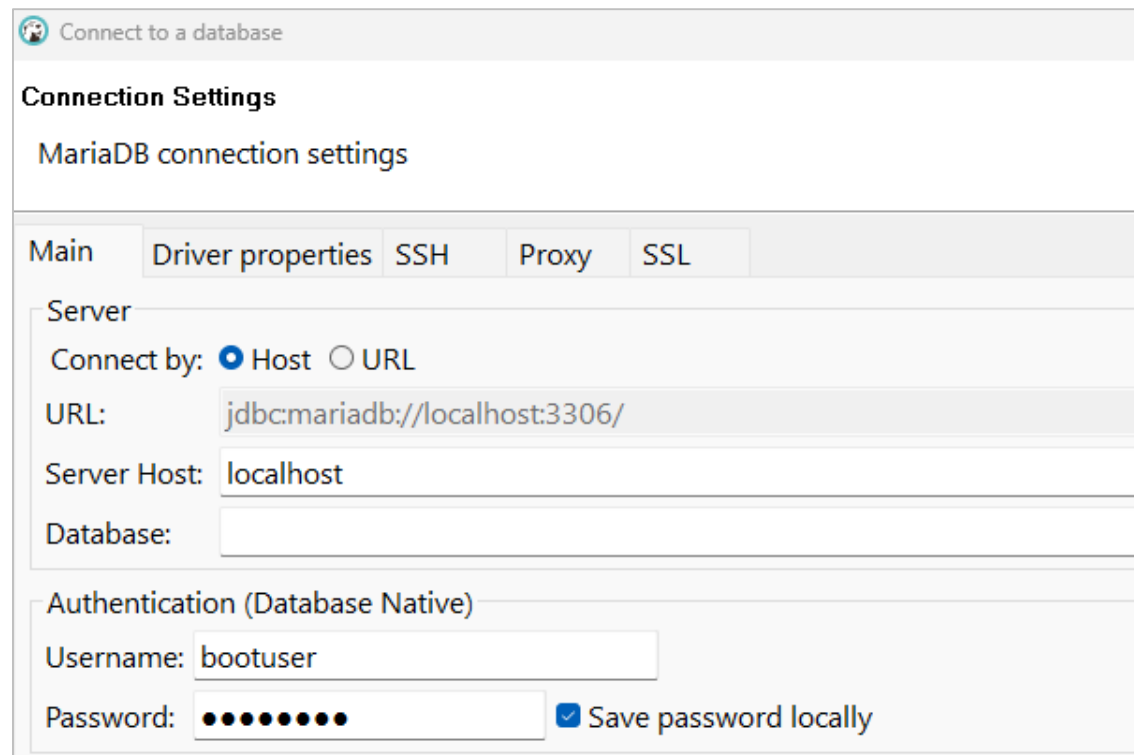
페이징 처리하기

part.5

쿼리메소드와 @Query

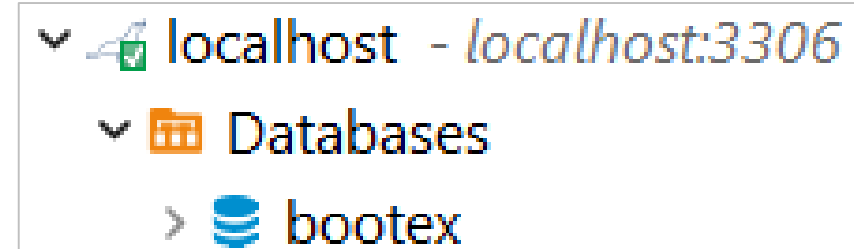
프로젝트를 생성하기 전에 데이터베이스에 문제가 없는지 확인한다.

1. MariaDB에 접속한다.
  2. "bootuesr" 사용자 계정이 있는지 확인한다.
  3. "bootex" 데이터베이스가 있는지 확인한다.
- > 계정 또는 데이터베이스가 없다면 새롭게 생성한다.



The screenshot shows the 'Connect to a database' dialog box with the following settings:

- Connection Settings**
  - MariaDB connection settings
- Main** tab is selected.
- Server** section:
  - Connect by: ☒ Host ☐ URL
  - URL: jdbc:mariadb://localhost:3306/
  - Server Host: localhost
  - Database: (empty)
- Authentication (Database Native)** section:
  - Username: bootuser
  - Password: (masked with dots)
  - ☒ Save password locally

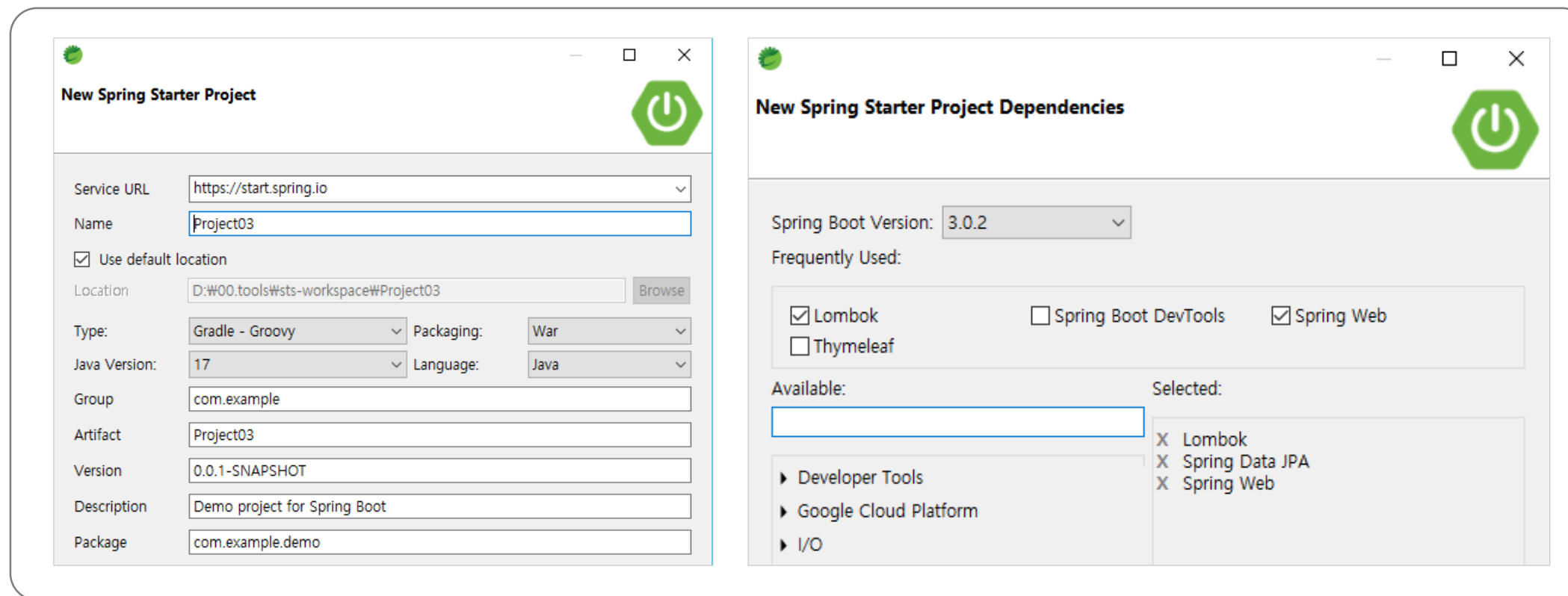


The screenshot shows a database explorer interface with the following structure:

- localhost - localhost:3306
  - Databases
    - bootex

실습을 위해 프로젝트를 생성한다.

라이브러리는 'Lombok, Spring Web, Spring Data JPA'를 선택한다.



프로젝트 생성 화면

# 프로젝트 준비

## 데이터베이스 연결

스프링 부트에서 JPA 라이브러리를 추가한 후 프로젝트를 실행하면 에러가 발생한다!

스프링 부트는 특정 라이브러리를 추가하면 관련된 설정을 자동으로 추가한다.

JPA와 관련된 설정을 추가되었지만, 구체적인 값이 없어서 발생하는 문제이다.

JPA 라이브러리를 사용하려면, 데이터베이스 연결 정보를 설정해야 한다.

- MariaDb를 연결할 드라이버
- 데이터베이스의 접속 정보

```
Error starting ApplicationContext. To display the condition evaluation report re-run y
2023-02-08T19:52:51.729+09:00 ERROR 13944 --- [           main] o.s.b.d.LoggingFailure

*****
APPLICATION FAILED TO START
*****

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded dat

Reason: Failed to determine a suitable driver class
```

프로젝트 실행 화면

# 프로젝트 준비

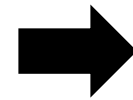
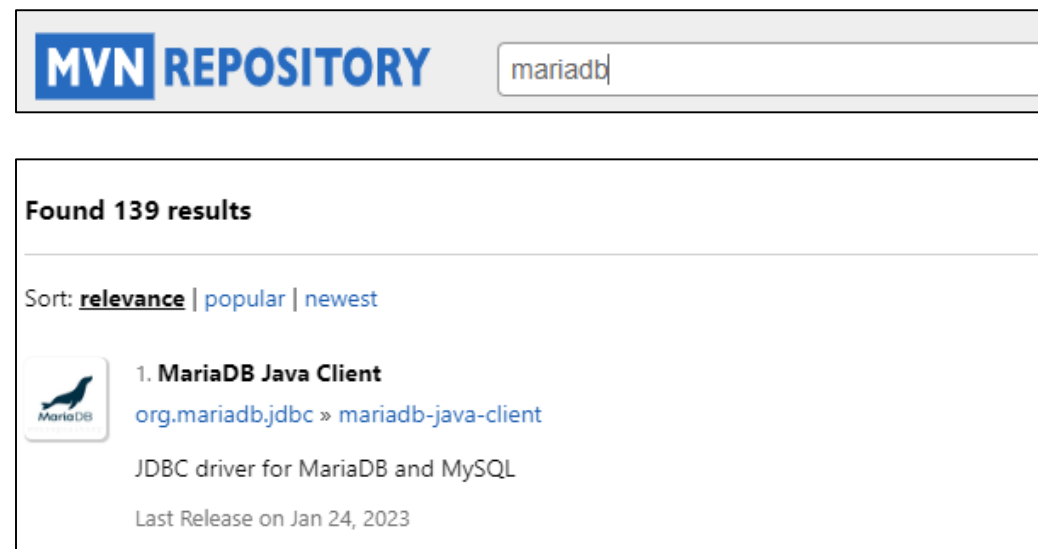
## 데이터베이스 연결

MariaDB 드라이버를 추가한다.

Maven 저장소에 접속하여 'mariadb'를 검색한 후, 첫번째 라이브러리를 선택한다.

'gradle' 탭을 클릭하고, 라이브러리 코드를 복사한다.

<https://mvnrepository.com>



패널을 클릭하면  
자동으로 복사된다

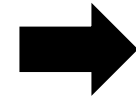
# 프로젝트 준비

## 데이터베이스 연결

복사한 내용을 'build.gradle' 파일의 'dependencies' 부분에 추가한다.  
파일을 수정한 후, 'Refresh Gradle Project' 메뉴를 실행한다.  
그러면 추가된 라이브러리가 자동으로 다운로드 된다.

```
15 }  
16 }  
17  
18 repositories {  
19     mavenCentral()  
20 }  
21  
22 dependencies {  
23     implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
24     implementation 'org.springframework.boot:spring-boot-starter-web'  
25     compileOnly 'org.projectlombok:lombok'  
26     annotationProcessor 'org.projectlombok:lombok'  
27     providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'  
28     testImplementation 'org.springframework.boot:spring-boot-starter-test'  
29     implementation group: 'org.mariadb.jdbc', name: 'mariadb-java-client', version: '3.1.2'  
30 }
```

붙여넣기



Project03 [boot]  
> src/main/java  
> src/main/resources  
> src/test/java  
> JRE System Library [JavaSE-17]  
> Project and External Dependencies  
bin  
> gradle  
> src  
build.gradle  
gradlew

Debug As  
Profile As  
Gradle  
Team

Refresh Gradle Project

Project and External Dependencies  
> mariadb-java-client-3.1.2.jar - C:₩

application.properties 파일은 Spring Boot 레퍼런스 문서를 참고하여 작성할 수 있다.  
데이터베이스 접속 정보를 문서에서 확인한 후, 이를 application.properties 파일에 추가한다.

스프링 부트 문서

|  |   |
|--|---|
| <code>spring.datasource.driver-class-name</code> | Fully qualified name of the JDBC driver. Auto-detected based on the URL by default. |
| <code>spring.datasource.url</code>               | JDBC URL of the database.   |
| <code>spring.datasource.username</code>          | Login username of the database.   |
| <code>spring.datasource.password</code>          | Login password of the database.   |

<https://docs.spring.io/spring-boot/appendix/application-properties/index.html>

application.properties 파일

```
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.datasource.url=jdbc:mariadb://localhost:3306/bootex
spring.datasource.username=bootuser
spring.datasource.password=bootuser
```



프로젝트를 다시 실행하면 정상적으로 실행된다.

```
ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1085 ms
r.hikari.HikariDataSource           : HikariPool-1 - Starting...
r.hikari.pool.HikariPool            : HikariPool-1 - Added connection org.mariadb.jdbc.Connection@68ddd415
r.hikari.HikariDataSource           : HikariPool-1 - Start completed.
te.jpa.internal.util.LogHelper      : HHH000204: Processing PersistenceUnitInfo [name: default]
nate.Version                        : HHH000412: Hibernate ORM core version 6.1.6.Final
```

프로젝트 실행 화면

### ORM(Object-Relational Mapping)이란?

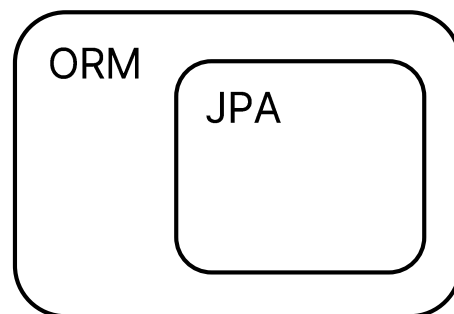
ORM은 객체지향 프로그램에서 작성된 객체를 관계형 데이터베이스에 매핑하고 저장하는 기술이다. 객체지향 프로그래밍과 관계형 데이터베이스는 데이터 구조를 설계하는 방식이 유사하여 ORM을 사용하여 둘을 쉽게 연결할 수 있다.

### JPA란?

JPA는 자바 언어를 위한 ORM 기술이다.

JPA는 자바 클래스와 데이터베이스 테이블을 매핑하여 객체와 데이터를 연결하는 기능을 제공한다.

이를 통해 개발자는 SQL을 직접 작성하지 않고 데이터베이스 연동을 쉽게 처리할 수 있다.



ORM과 JPA의 관계

```
public class Book {  
    int bookNo;  
    String title;  
    String publisher;  
    Date pubDate;  
    int Price;  
}
```

자바 클래스

| # | 이름        | 데이터 유형  |
|---|-----------|---------|
| 1 | BOOK_NO   | INT     |
| 2 | TITLE     | VARCHAR |
| 3 | PUBLISHER | VARCHAR |
| 4 | PUB_DATE  | DATE    |
| 5 | PRICE     | INT     |

DB 테이블

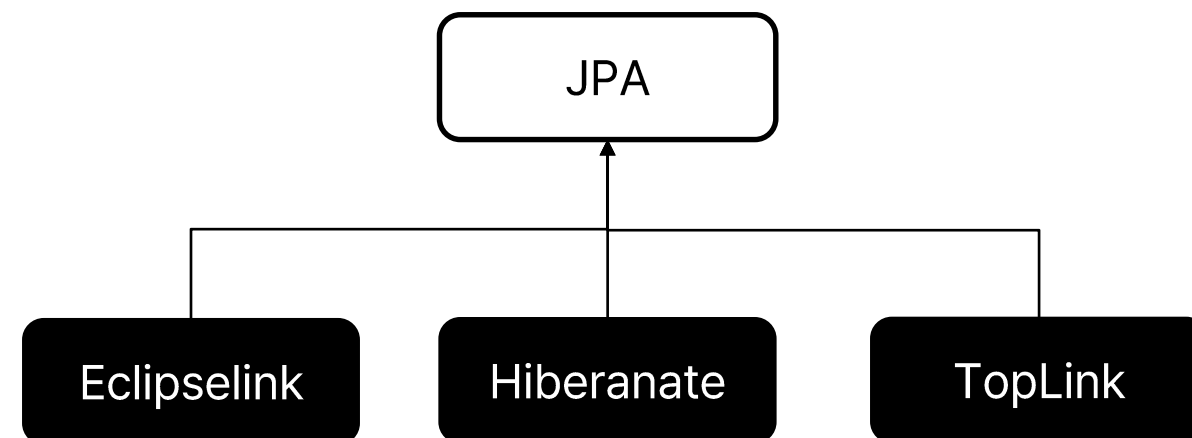
### JPA의 기능

- Create, Read, Update, Delete 작업을 수행하는 SQL을 자동으로 생성한다.
- 엔티티(Entity)를 기반으로 필요한 테이블을 자동으로 생성한다.

### JPA의 구현체

- JPA는 객체와 데이터베이스 간의 매핑을 위한 인터페이스이다.
- Spring Boot는 JPA의 구현체 중에서 'Hibernate'를 사용한다.

JPA와 구현체의 관계

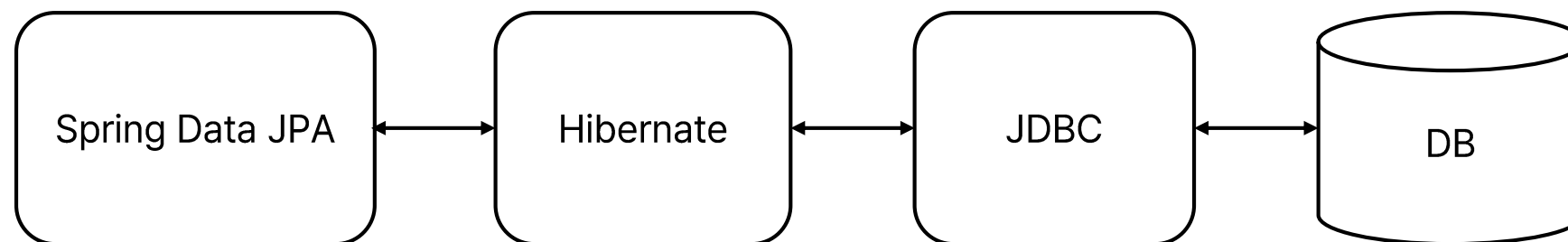


### Spring Data JPA란?

Spring Data JPA는 Hibernate를 쉽게 사용할 수 있도록 도와주는 라이브러리이다.  
JpaRepository 인터페이스를 제공하여 CRUD 작업을 쉽게 처리할 수 있다.

**프로젝트에서 데이터베이스까지 연결되는 과정은 다음과 같다.**

1. 개발자는 Spring Data JPA가 생성한 리파지토리를 사용하여 데이터 작업을 처리한다
2. JPA(Hibernate)는 리파지토리의 메소드를 해석하여 필요한 SQL을 자동으로 생성한다.
3. JDBC는 자바 프로그램과 데이터베이스를 연결하여 생성된 SQL을 전달한다.
4. JDBC를 통해 데이터베이스에 처리된 결과를 다시 받아온다.



# 엔티티와 리파지토리    엔티티 클래스와 JpaRepository

## Spring Data JPA를 사용할 때 필요한 두 가지 클래스

1. 엔티티 클래스(Entity): 데이터베이스의 테이블 구조를 나타내는 클래스
2. 리파지토리 클래스(Repository): 엔티티 객체를 통해 데이터 작업을 처리하는 기능을 제공하는 클래스

```
▼ com.example.demo
  ▼ entity
    > Memo.java
  ▼ repository
    > MemoRepository.java
```

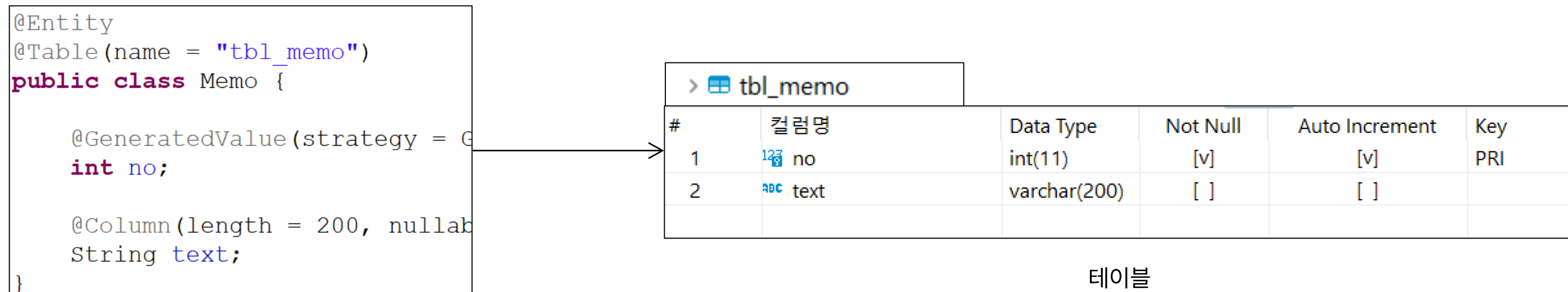
엔티티와 리파지토리

엔티티 클래스는 데이터베이스의 테이블과 동일한 구조로 작성해야 한다.

아래와 같이 엔티티를 작성한 후 프로젝트를 실행하면, 'tbl\_memo' 테이블이 자동으로 생성된다.

### 엔티티 만드는 방법

1. 엔티티 표시
2. 테이블 이름 설정: 클래스와 테이블 이름이 같을 경우에는 생략
3. 기본키(PK) 설정: 키 생성 방식
4. 일반 컬럼 설정: 컬럼의 크기, NULL 허용 여부



엔티티 클래스

테이블

# 엔티티와 리파지토리      엔티티 클래스에서 사용하는 어노테이션

| 어노테이션           | 설명  | 위치  |
|-----------------|---|-----|
| @Entity         | 해당 클래스가 JPA가 관리하는 객체임을 나타낸다.<br>데이터베이스에 동일한 이름의 테이블이 있으면 해당 클래스와 연결되고, 없으면 자동으로 클래스에 맞는 테이블을 생성한다 | 클래스 |
| @Table          | 테이블 이름을 설정한다  | 클래스 |
| @Id             | 엔티티 클래스의 필드를 Primary key 로 지정할 때 사용한다<br>엔티티 클래스는 PK컬럼을 반드시 가져야 한다                                | 필드  |
| @GeneratedValue | 키 값을 자동으로 생성할 때 사용한다  | 필드  |
| @Column         | 컬럼의 속성을 지정한다 (자료형, 크기, 제약사항)  | 필드  |

| 키 생성 방식  | 설명   |
|----------|--|
| AUTO     | JPA 알아서 생성방식을 선택한다                                 |
| IDENTITY | 데이터베이스에 따라 생성방식을 선택한다 (MariaDB의 경우 auto increment) |
| SEQUENCE | 데이터베이스의 시퀀스를 사용해서 키를 생성한다                          |
| TABLE    | 키 전용 테이블을 사용하여 키를 생성한다                             |

# 엔티티와 리파지토리 JPA를 위한 스프링 설정

application.properties 파일에 JPA 관련 설정을 추가한다.

- spring.jpa.hibernate.ddl-auto: "alter"는 테이블에 변경이 있을 때 자동으로 수정하고, 테이블이 없을 경우에는 "create"로 새로 생성한다.
- spring.jpa.properties.hibernate.format\_sql: SQL을 보기 좋게 포맷하여 출력한다.
- spring.jpa.show-sql: JPA가 실행할 때 생성되는 SQL을 콘솔에 출력한다.

PPT 내용을 복사해서  
넣어주세요!

```
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
```

application.properties 파일

```
Problems Javadoc Declaration Console x Gradle Tasks Gradle Executions
<terminated> MemoRepositoryTest 데이터등록 [JUnit] D:\#0.tool\spring-tool-suite-4-4.19.0.RELEASE-e4.28.0-win32.win3
2023-07-09T14:55:23.518+09:00 INFO 58788 --- [
Hibernate:

    create table tbl_memo (
        no integer not null auto_increment,
        text varchar(200),
        primary key (no)
    ) engine=InnoDB
```

SQL 출력 화면



# 엔티티와 리파지토리 JpaRepository 인터페이스

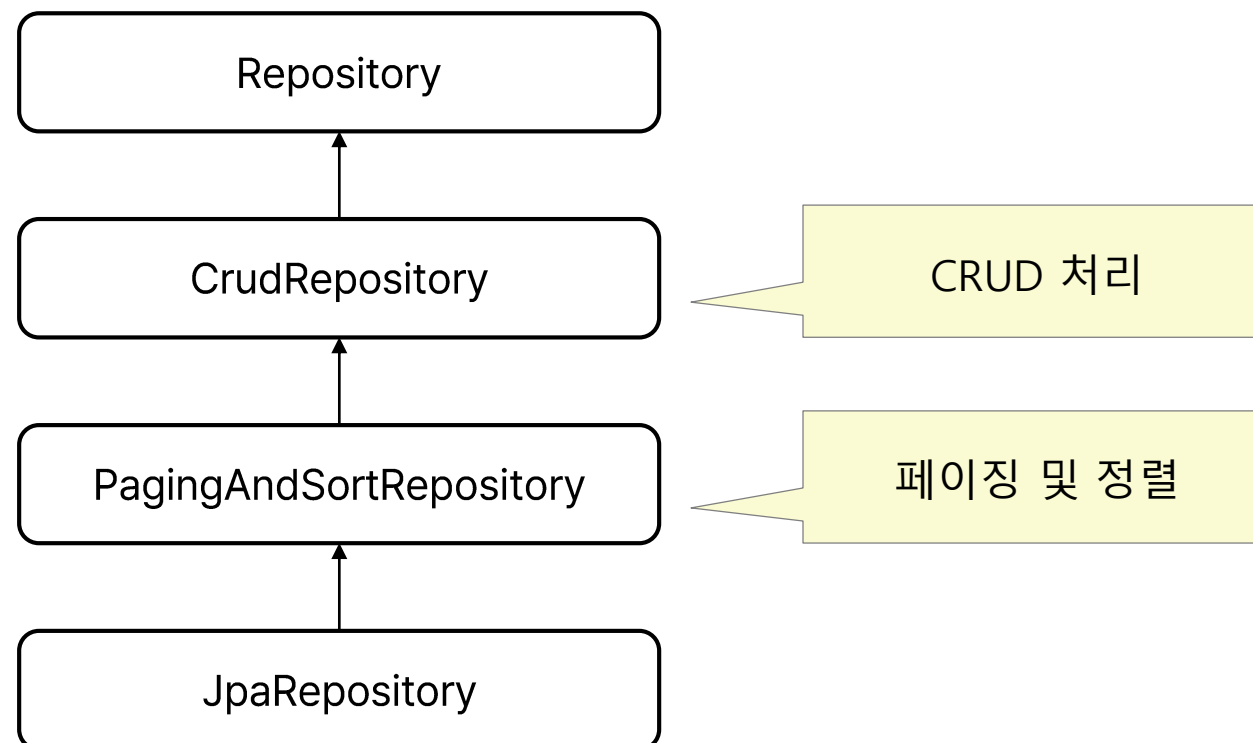
## Repository란?

Repository는 엔티티를 처리하는 인터페이스로, 데이터베이스 작업을 처리하는 기능을 가지고 있다.

다양한 Repository 인터페이스가 있으며 CRUD 기능, 페이징, 정렬 등 다양한 기능을 지원한다.

특별한 이유가 없다면, 가장 기능이 많은 JpaRepository를 사용하는 것이 좋다.

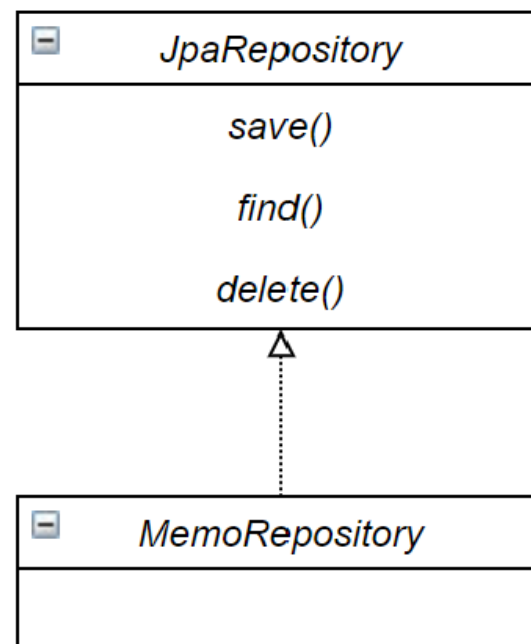
JpaRepository는 상위 인터페이스들을 상속받아 모든 기능이 추가되었다.



# 엔티티와 리파지토리 JpaRepository 인터페이스

## JpaRepository의 기능

- CRUD기능: 데이터 추가, 조회, 수정, 삭제
- 페이징: 데이터를 페이지 단위로 조회
- 정렬: 오름차순 또는 내림차순으로 데이터 정렬
- 복잡한 쿼리: JPQL, 네이티브 쿼리, @Query 어노테이션을 통해 복잡한 조건 검색을 지원



```
memoRepository.save(memo);

Optional<Memo> result = memoRepository.findById(1);

Memo memo = new Memo(1, "글이수정되었습니다");
memoRepository.save(memo);

memoRepository.deleteById(1);
```

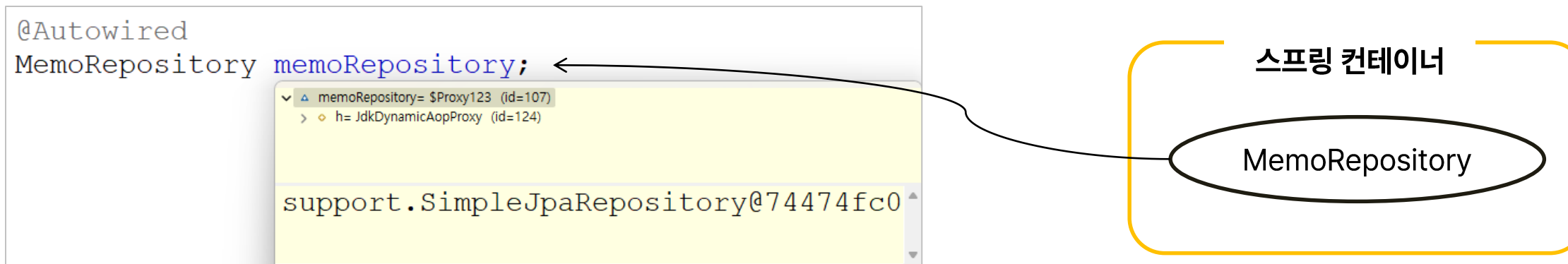
리파지토리의 기능들

# 엔티티와 리파지토리 JpaRepository 인터페이스

## 리파지토리 생성하기

1. JpaRepository를 상속받는 리파지토리 인터페이스를 생성한다.  
인터페이스에 엔티티 클래스와 엔티티의 PK 타입을 지정한다.
2. Spring Data JPA가 이 인터페이스를 구현한 객체를 생성하고, 이를 스프링 컨테이너에 빈으로 등록한다.
3. 등록된 리파지토리 빈은 @Autowired를 통해 주입받아 사용할 수 있다.

```
interface MemoRepository extends JpaRepository<Memo, Integer> {  
  
}
```



### 리파지토리가 생성되는 과정

인터페이스를 정의하면, Spring이 해당 인터페이스의 실제 구현체를 자동으로 생성한다.

이 과정에서 우리가 직접 구현하지 않은 클래스가 만들어진다.

예를 들어, 다음과 같이 MemoRepository 라는 인터페이스를 정의하면 Spring은 이 인터페이스를 기반으로 구현체를 생성한다.

리파지토리 인터페이스

```
public interface MemoRepository extends JpaRepository<Memo, Integer>{  
}
```

리파지토리를 상속받은 구현 클래스 (예상)

```
public class MemoRepositoryImpl implements MemoRepository {  
    @Override  
    public Optional<Memo> findById(Integer id) {  
        return optional; //DB에서 조회한 데이터 반환  
    }  
    @Override  
    public List<Memo> findAll() {  
        return list; //DB에서 조회한 데이터 반환  
    }  
    ...  
}
```





| 메소드           | 설명  |
|---------------|---|
| save(엔티티)     | 엔티티를 데이터베이스에 저장하거나 업데이트한다<br>데이터베이스에 이미 존재하는 경우에는 해당 엔티티를 업데이트하고,<br>존재하지 않는 경우에는 새로운 레코드로 저장한다 |
| deleteAll()   | 데이터베이스에 있는 모든 엔티티를 삭제한다   |
| deleteById(키) | 주어진 아이디에 해당하는 엔티티를 삭제한다   |
| findAll()     | 데이터베이스에 있는 모든 엔티티를 조회한다   |
| findById(키)   | 주어진 아이디에 해당하는 엔티티를 조회한다   |
| count()       | 데이터베이스에 저장된 엔티티의 총 개수를 반환한다   |

# 엔티티와 리파지토리 연습문제





**Q1.** Book 엔티티 클래스를 만들고, 해당 엔티티에 대한 테이블을 생성하세요.

BookRepository 인터페이스를 만들고, 이 리파지토리를 사용하여 데이터 조회, 추가, 수정, 삭제 기능을 테스트하세요.

tbl\_book 테이블

| # | 컬럼명  | Data Type    | Not Null | Auto Increment | Key |
|---|--|--------------|----------|----------------|-----|
| 1 |  book_no    | int(11)      | [v]      | [v]            | PRI |
| 2 |  price      | int(11)      | [v]      | [ ]            |     |
| 3 |  publisher | varchar(100) | [v]      | [ ]            |     |
| 4 |  title    | varchar(30)  | [v]      | [ ]            |     |

tbl\_book 테이블에 저장된 데이터





|   |  book_no |  price |  publisher |  title |
|---|---|---|---|---|
| 1 | 1   | 20,000  | 한빛출판사   | 자바프로그래밍입문   |
| 2 | 2   | 25,000  | 남가람북스   | 스프링부트프로젝트   |
| 3 | 3   | 40,000  | 남가람북스   | 실무로 끝내는 PHP   |
| 4 | 4   | 35,000  | 이지스퍼블리싱   | 알고리즘코딩테스트   |

# 엔티티와 리파지토리 연습문제

**Q2.** Gift 엔티티 클래스를 만들고, 해당 엔티티에 대한 테이블을 생성하세요.

GiftRepository 인터페이스를 만들고, 이 리파지토리를 사용하여 데이터 조회, 추가, 수정, 삭제 기능을 테스트하세요.

tbl\_gift 테이블

| # | 컬럼명  | Data Type   | Not Null | Auto Increment | Key |
|---|--|-------------|----------|----------------|-----|
| 1 |  no     | int(11)     | [v]      | [v]            | PRI |
| 2 |  name   | varchar(20) | [v]      | [ ]            |     |
| 3 |  price  | int(11)     | [v]      | [ ]            |     |
| 4 |  type | varchar(20) | [v]      | [ ]            |     |

tbl\_gift 테이블에 저장된 데이터





|   |  no |  name |  price |  type |
|---|--|--|---|--|
| 1 | 1  | 참치세트   | 10,000  | 식품   |
| 2 | 2  | 햄세트  | 20,000  | 식품   |
| 3 | 3  | 샴푸세트   | 30,000  | 생활용품   |
| 4 | 4  | 세차용품   | 40,000  | 생활용품   |
| 5 | 5  | 주방용품   | 50,000  | 생활용품   |
| 6 | 6  | 노트북  | 60,000  | 가전제품   |
| 7 | 7  | 벽걸이TV  | 70,000  | 가전제품   |

# 엔티티와 리파지토리 연습문제





**Q3.** Order 엔티티 클래스를 만들고, 해당 엔티티에 대한 테이블을 생성하세요.

OrderRepository 인터페이스를 만들고, 이 리파지토리를 사용하여 데이터 조회, 추가, 수정, 삭제 기능을 테스트하세요.

tbl\_order 테이블

| # | 컬럼명  | Data Type    | Not Null | Auto Increment | Key |
|---|--|--------------|----------|----------------|-----|
| 1 |  order_no       | int(11)      | [v]      | [v]            | PRI |
| 2 |  customer_name  | varchar(30)  | [v]      | [ ]            |     |
| 3 |  order_date     | date         | [v]      | [ ]            |     |
| 4 |  ship_address | varchar(100) | [ ]      | [ ]            |     |

tbl\_order 테이블에 저장된 데이터

|   |  order_no |  customer_name |  order_date |  ship_address |
|---|--|---|--|--|
| 1 | 1  | 둘리  | 2023-07-01   | 인천 구월동   |
| 2 | 2  | 또치  | 2023-07-02   | 인천 연수동   |
| 3 | 3  | 도우너   | 2023-07-03   | 부산 동래동   |
| 4 | 4  | 마이콜   | 2023-07-01   | [NULL]   |
| 5 | 5  | 고길동   | 2023-07-02   | [NULL]   |

[Hint] LocalDate 객체를 생성하는 방법

```
LocalDate localDate = LocalDate.of(2023, 7, 1);
```







# 엔티티와 리파지토리 연습문제





**Q4.** Member 엔티티 클래스를 만들고, 해당 엔티티에 대한 테이블을 생성하세요.

MemberRepository 인터페이스를 만들고, 이 리파지토리를 사용하여 데이터 조회, 추가, 수정, 삭제 기능을 테스트하세요.

tbl\_member 테이블

| # | 컬럼명   | Data Type    | Not Null | Auto Increment | Key |
|---|---|--------------|----------|----------------|-----|
| 1 |  user_id         | varchar(255) | [v]      | [ ]            | PRI |
| 2 |  grade           | varchar(255) | [v]      | [ ]            |     |
| 3 |  password       | varchar(255) | [v]      | [ ]            |     |
| 4 |  register_date | date         | [v]      | [ ]            |     |

tbl\_member 테이블에 저장된 데이터

|   |  user_id |  grade |  password |  register_date |
|---|---|---|--|---|
| 1 | admin   | 관리자   | 1234   | 2024-09-11  |
| 2 | user1   | 사용자   | 1234   | 2024-09-11  |
| 3 | user2   | 사용자   | 1234   | 2024-09-11  |
| 4 | yoyt22  | 관리자   | 1234   | 2024-09-11  |

등록일은 현재시간으로 저장하세요!

### 페이징(Paging)이란?

많은 양의 콘텐츠를 효율적으로 처리하기 위해 페이지 단위로 나누어 표시하는 기술이다.

주로 게시판 같은 데이터 목록을 다룰 때 사용된다.

사용자는 모든 콘텐츠를 한번에 보지 않고, 원하는 페이지로 이동하여 필요한 데이터만 선택적으로 볼 수 있다.

| #                   | 제목    |
|---------------------|-------|
| <a href="#">100</a> | 100번글 |
| <a href="#">99</a>  | 99번글  |
| <a href="#">98</a>  | 98번글  |
| <a href="#">97</a>  | 97번글  |
| <a href="#">96</a>  | 96번글  |
| <a href="#">95</a>  | 95번글  |
| <a href="#">94</a>  | 94번글  |
| <a href="#">93</a>  | 93번글  |
| <a href="#">92</a>  | 92번글  |
| <a href="#">91</a>  | 91번글  |
| <a href="#">90</a>  | 90번글  |
| <a href="#">89</a>  | 89번글  |
| <a href="#">88</a>  | 88번글  |
| <a href="#">87</a>  | 87번글  |
| <a href="#">86</a>  | 86번글  |
| <a href="#">85</a>  | 85번글  |

페이징 처리 없이 모든 데이터를 표시한 게시판

| #                   | 제목    | 작성자   |
|---------------------|-------|-------|
| <a href="#">100</a> | 100번글 | user1 |
| <a href="#">99</a>  | 99번글  | user1 |
| <a href="#">98</a>  | 98번글  | user1 |
| <a href="#">97</a>  | 97번글  | user1 |
| <a href="#">96</a>  | 96번글  | user1 |
| <a href="#">95</a>  | 95번글  | user1 |
| <a href="#">94</a>  | 94번글  | user1 |
| <a href="#">93</a>  | 93번글  | user1 |
| <a href="#">92</a>  | 92번글  | user1 |
| <a href="#">91</a>  | 91번글  | user1 |

|                   |                   |                   |                   |                   |                   |                   |                   |                   |                    |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|--------------------|
| <a href="#">1</a> | <a href="#">2</a> | <a href="#">3</a> | <a href="#">4</a> | <a href="#">5</a> | <a href="#">6</a> | <a href="#">7</a> | <a href="#">8</a> | <a href="#">9</a> | <a href="#">10</a> |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|--------------------|

페이징 처리를 통해 데이터를 나눈 게시판

## 페이징 처리


- Pageable는 많은 데이터를 페이지 단위로 나누어 처리할 수 있도록 도와주는 인터페이스이다.
- Pageable은 페이지 번호, 한 페이지에 표시할 데이터 개수, 정렬 방식 등 정보를 포함하여 페이지 요청 정보를 만든다.
- find 메소드와 함께 사용하여 특정 페이지의 데이터를 조회할 수 있다.

### 특정 페이지의 데이터 조회

```
Pageable pageable = PageRequest.of(0, 10);  
  
Page<Memo> page = repository.findAll(pageable);
```

### 첫번째 페이지를 조회하는 SQL

```
select  
    no,  
    text  
from  
    tbl_memo limit 10;
```



```
<terminated> MemoRepositoryTest1_2 페이징처리 [JUnit] D:\W00\tool\spring-tool-suite-4-4.19.0.RELEASE-e4.2  
Memo (no=1, text=Sample..1)  
Memo (no=2, text=Sample..2)  
Memo (no=3, text=Sample..3)  
Memo (no=4, text=Sample..4)  
Memo (no=5, text=Sample..5)  
Memo (no=6, text=Sample..6)  
Memo (no=7, text=Sample..7)  
Memo (no=8, text=Sample..8)  
Memo (no=9, text=Sample..9)  
Memo (no=10, text=Sample..10)
```

JpaRepository는 다양한 조건 검색을 처리하기 위한 여러 기능을 제공한다.

- 쿼리 메소드(Query Method): 메소드 이름을 기반으로 자동으로 쿼리를 생성하는 기능
- @Query: SQL과 유사한 JPQL을 사용하여 엔티티 클래스를 기반으로 쿼리를 생성하는 기능
- QueryDSL: 복잡한 조건 검색을 동적으로 처리하는 기능

QueryDSL은 4장에 배울 예정입니다

예시

```
// 쿼리 메소드
List<Memo> findByNoBetween(int from, int to);

// @Query
@Query("select m from Memo m where m.no between :from and :to")
List<Memo> get3(@Param("from") int from, @Param("to") int to);
```

# 쿼리메소드

# 쿼리메소드

## 쿼리메소드란?

메소드 이름을 기반으로 SQL 쿼리를 생성하는 기능이다.

메소드 이름을 규칙에 맞게 작성하면, JPA가 이를 해석하여 쿼리를 실행한다.

### 쿼리 메소드 예시

| Keyword  | Sample                             | JPQL snippet  |
|----------|------------------------------------|---|
| Distinct | findDistinctByLastnameAndFirstname | <pre>select distinct ... where x.lastname = ?1 and x.firstname = ?2</pre> |

### 스프링 공식 문서

| 3. Dependencies<br>4. Working with Spring Data Repositories<br>5. Reference Documentation<br>5.1. JPA Repositories<br>5.1.1. Introduction<br>5.1.2. Persisting Entities<br>5.1.3. Query Methods<br>Query Lookup Strategies<br><b>Query Creation</b><br>Using JPA Named Queries<br>Using @Query | <table><tr><th>Keyword</th><th>Sample</th></tr><tr><td>Distinct</td><td>findDistinctByLastnameAndFirstname</td></tr><tr><td>And</td><td>findByLastnameAndFirstname</td></tr><tr><td>Or</td><td>findByLastnameOrFirstname</td></tr><tr><td>Is, Equals</td><td>findByFirstname, findByFirstnameIs, findByFirstnameEquals</td></tr><tr><td>Between</td><td>findByStartDateBetween</td></tr></table> | Keyword | Sample | Distinct | findDistinctByLastnameAndFirstname | And | findByLastnameAndFirstname | Or | findByLastnameOrFirstname | Is, Equals | findByFirstname, findByFirstnameIs, findByFirstnameEquals | Between | findByStartDateBetween |
|--|--|---------|--------|----------|------------------------------------|-----|----------------------------|----|---------------------------|------------|---|---------|------------------------|
| Keyword  | Sample   |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |
| Distinct   | findDistinctByLastnameAndFirstname   |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |
| And  | findByLastnameAndFirstname   |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |
| Or   | findByLastnameOrFirstname  |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |
| Is, Equals   | findByFirstname, findByFirstnameIs, findByFirstnameEquals  |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |
| Between  | findByStartDateBetween   |         |        |          |                                    |     |                            |    |                           |            |   |         |                        |

<https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html>

### 메소드 이름 규칙

- 메소드 이름은 findBy 또는 getBy로 시작해야 한다.
- 엔티티의 필드와 Between, Like 등의 키워드를 조합하여 검색 조건을 만든다.
- 사용된 키워드에 따라 필요한 파라미터의 개수가 결정된다.

스프링 문서 일부

Between

findByStartDateBetween

... where x.startDate between ?1 and ?2

메모리파지토리 코드

```
List<Memo> findByNoBetween(int from, int to);
```

**@Query**는 메소드의 이름과 상관없이 직접 JPQL 또는 SQL 쿼리를 작성하는 기능이다. 쿼리 메소드는 조건이 많아지면 이름이 지나치게 길어질 수 있어 불편할 수 있다. 따라서 복잡한 쿼리는 쿼리 메소드 대신 @Query를 사용하는 것이 좋다.

### JPQL을 사용한 조회

- JPQL은 객체지향 쿼리로, SQL과 비슷하지만 작성 방법이 다르다.
- 테이블 대신 엔티티를 사용하고, 테이블의 컬럼 대신 엔티티의 필드를 사용한다.

### 파라미터 처리

- 파라미터는 ":파라미터이름"으로 처리한다. (예: ":xxx")

```
@Query("select m from Memo m where m.no < :mno")  
List<Memo> get(@Param("mno") int mno);
```

### 순수한 SQL을 사용한 조회

데이터베이스에서 사용하는 SQL을 그대로 작성하는 방식이다.

nativeQuery 속성을 true로 설정해야 SQL을 사용할 수 있다.

```
@Query(value = "select * from tbl_memo order by no desc", nativeQuery = true)  
List<Memo> get();
```



### 1. Query Method

간단한 검색에 적합.

조건이 많아지면 메소드 이름이 길어져 사용이 불편할 수 있음.

### 2. JPQL

객체지향 쿼리로 데이터베이스가 변경되어도 쿼리를 수정할 필요가 없음.

데이터베이스가 중간에 변경될 가능성이 있는 경우에 적합.

SQL 보다 작성이 어려움.

### 3. Native SQL

데이터베이스에서 사용하는 SQL을 그대로 작성하여 사용이 쉬움.

특정 데이터베이스에 종속적이므로, 중간에 데이터베이스가 변경되면 모든 쿼리를 수정해야함.

→ 결론: 우리 시스템은 중간에 데이터베이스 변경될 가능성이 없기 때문에, 가장 사용이 쉬운 SQL을 사용하시면 됩니다!

**Q5.** BookRepository 인터페이스에 다음과 같이 검색 메소드를 추가하세요.  
순수한 SQL을 사용하여 작성하세요.

먼저 테이블의 데이터를  
처음 상태로 초기화 해주세요!

1) 제목이 '자바프로그래밍입문'인 책을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 336ms

`title = ?`

`Book(bookNo=1, title=자바프로그래밍입문, publisher=한빛출판사, price=20000)`

2) 가격이 '3만원'이상이고 출판사가 '남가람북스'인 책을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 247ms

`Book(bookNo=3, title=실무로 끝내는 PHP, publisher=남가람북스, price=40000)`

3) 출판사가 '한빛출판사' 또는 '이지스퍼블리싱'인 책을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 318ms

`publisher IN (?, ?)`

`Book(bookNo=1, title=자바프로그래밍입문, publisher=한빛출판사, price=20000)`

`Book(bookNo=4, title=알고리즘코딩테스트, publisher=이지스퍼블리싱, price=35000)`

**Q6.** GiftRepository 인터페이스에 다음과 같이 검색 메소드를 추가하세요.  
순수한 SQL을 사용하여 메소드를 작성하세요.

1) 가격이 '50000원'이상인 선물을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 291ms

```
Gift(no=5, name=주방용품, type=생활용품, price=50000)
Gift(no=6, name=노트북, type=가전제품, price=60000)
Gift(no=7, name=벽걸이TV, type=가전제품, price=70000)
```

2) 이름이 '세트'로 끝나는 선물을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 227ms

```
Gift(no=1, name=참치세트, type=식품, price=10000)
Gift(no=2, name=햄세트, type=식품, price=20000)
Gift(no=3, name=샴푸세트, type=생활용품, price=30000)
```

3) 가격은 '40000원'이하고 품목은 '생활용품'인 선물을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 252ms

```
Gift(no=3, name=샴푸세트, type=생활용품, price=30000)
Gift(no=4, name=세차용품, type=생활용품, price=40000)
```

**Q7.** OrderRepository 인터페이스에 다음과 같이 검색 메소드를 추가하세요.

순수한 SQL을 사용하여 메소드를 작성하세요.

1) 주소지가 '인천'인 주문을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 299ms

```
Order(orderNo=1, customerName=둘리, shipAddress=인천 구월동, orderDate=2023-07-01T00:00)
Order(orderNo=2, customerName=또치, shipAddress=인천 연수동, orderDate=2023-07-02T00:00)
```

2) 주문일이 '7월3일'인 주문을 검색

✓ 테스트 통과: 1 / 1개 테스트 - 285ms

```
Order(orderNo=3, customerName=도우너, shipAddress=부산 동래동, orderDate=2023-07-03T00:00)
```

**Q8.** MemberRepository 인터페이스에 다음과 같이 검색 메소드를 추가하세요.  
순수한 SQL을 사용하여 메소드를 작성하세요.

1) 아이디가 'user'로 시작하는 회원 검색

✓ 테스트 통과: 1 / 1개 테스트 - 315ms

```
Member(userId=user1, password=1234, grade=사용자, registerDate=2024-09-11)  
Member(userId=user2, password=1234, grade=사용자, registerDate=2024-09-11)
```

2) 관리자 등급인 회원 검색

✓ 테스트 통과: 1 / 1개 테스트 - 406ms

```
Member(userId=admin, password=1234, grade=관리자, registerDate=2024-09-11)  
Member(userId=yoyt22, password=1234, grade=관리자, registerDate=2024-09-11)
```