

# 스프링부트 JPA

## <제목 차례>

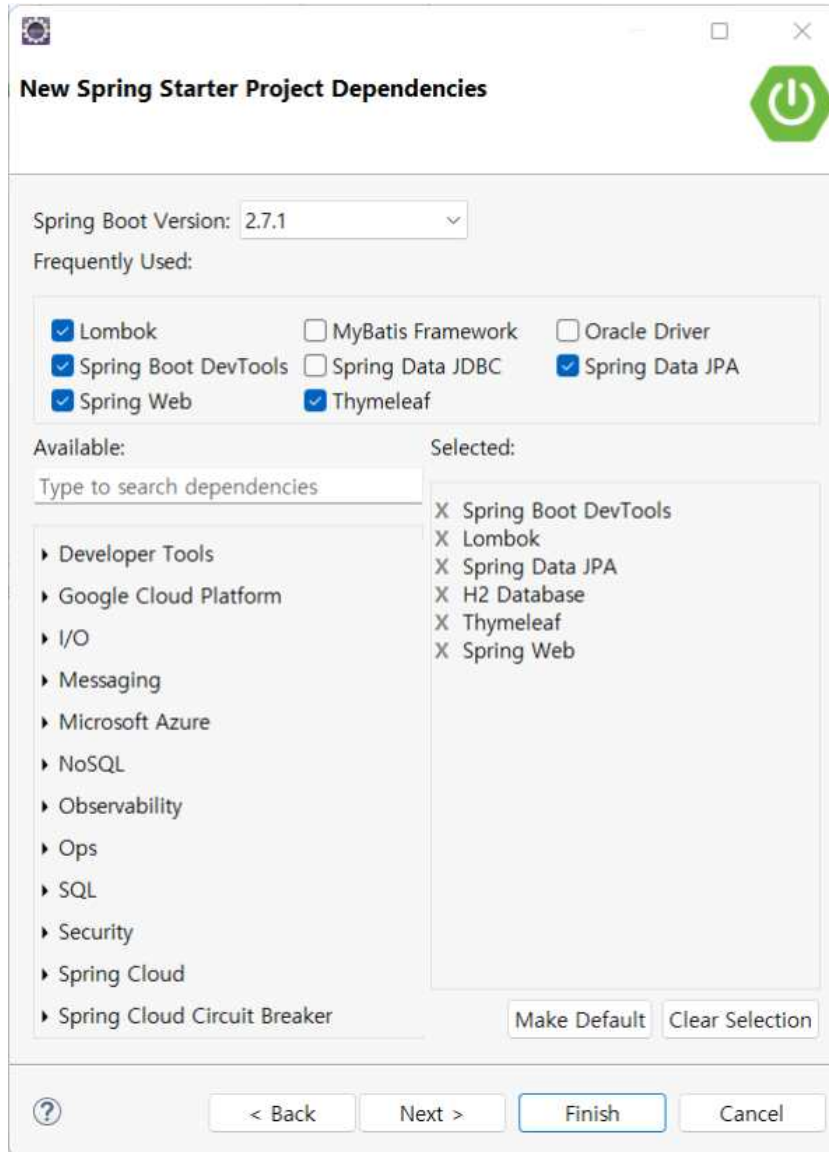
1. 스프링부트웹 JPA, H2 이용 .....	2
가. 프로젝트 생성 .....	2
나. 의존성 구성 확인 .....	2
다. 환경설정 .....	3
라. Entity 클래스 .....	3
마. Repository 인터페이스 .....	3
바. 컨트롤러 .....	4
사. 뷰페이지 .....	4
아. 테스트 .....	4
자. 데이터 입력 .....	4
차. 서버 시작할 때 샘플 데이터 입력 .....	7
2. 스프링부트웹 JPA, ORACLE 이용 .....	9
가. 프로젝트 생성 .....	9
나. 환경설정 .....	9
다. Entity 클래스 .....	10
라. Repository 인터페이스 .....	11
마. 일대일 .....	12
바. 일대다 .....	12
3. JPA .....	13
가. 개요 .....	13
나. jpa, hibernate 설정 .....	15
다. JPQL .....	18
4. 부록 .....	19
가. H2 Database .....	19

## 1. 스프링부트웹 JPA, H2 이용

참고사이트 : <https://spring.io/guides/gs/accessing-data-jpa/>

### 가. 프로젝트 생성

Devtools, Lombok, JPA, H2, Thymeleaf, Web 라이브러리 선택



### 나. 의존성 구성 확인

- build.gradle

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation('org.springframework.boot:spring-boot-starter-test') {
        exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
    }
}
```

## 다. 환경설정

```
#tomcat 설정
server.port=80

# jpa
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
#spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true
```

## 라. Entity 클래스

```
package com.example.demo;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import lombok.Data;

@Data
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(length = 20, nullable = false)
    private String name;

    @Column(length = 20, nullable = false, unique = true)
    private String phone;
}
```

Hibernate:

```
create table customer (
  id number(19,0) not null,
  name varchar2(20 char) not null,
  phone varchar2(20 char) not null,
  primary key (id)
)
```

Hibernate:

```
alter table customer
drop constraint UK_o3uty20c6csmx5y4uk2tc5r4m
```

Hibernate:

```
alter table customer
add constraint UK_o3uty20c6csmx5y4uk2tc5r4m unique (phone)
```

## 마. Repository 인터페이스

```
package com.example.demo;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long>{
    //비워있어도 잘 작동함.
}
```

## 바. 컨트롤러

```
@Controller
public class CustController {

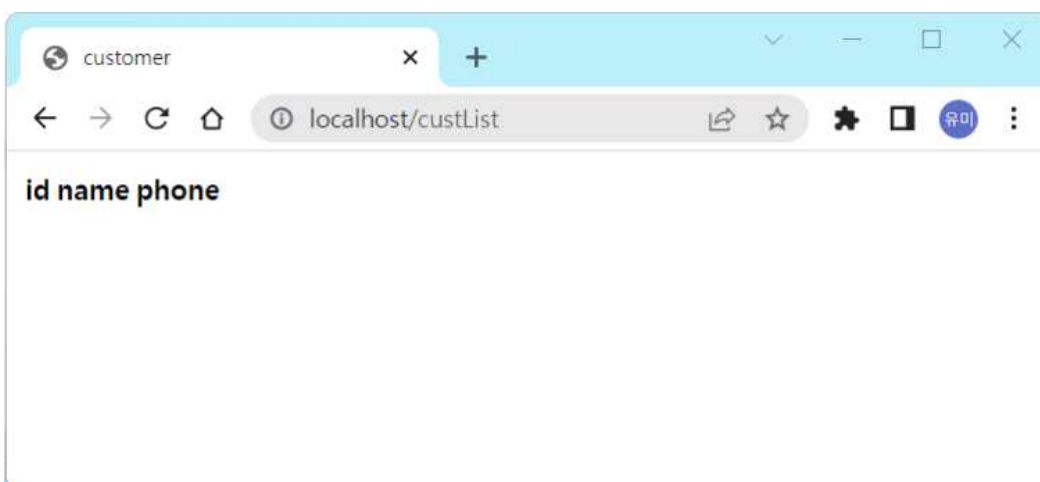
    @Autowired CustomerRepository dao;

    @RequestMapping("/custList");
    public String custList(Model model){
        model.addAttribute("custList", dao.findAll());
        return "custList";
    }
}
```

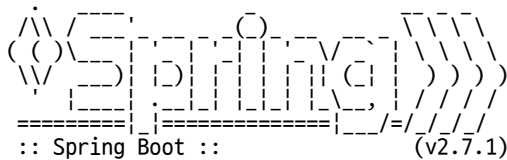
## 사. 뷰페이지

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>customer</title>
</head>
<body>
<table>
<tr>
<th>id</th> <th>name</th> <th>phone</th>
</tr>
<tr th:each="cust : ${custList}">
<td th:text="${cust.id}"></td>
<td th:text="${cust.name}"></td>
<td th:text="${cust.phone}"></td>
</tr>
</table>
</body>
</html>
```

## 아. 테스트



## 자. 데이터 입력



```

2022-07-13 13:31:17.898 INFO 15052 --- [ restartedMain] com.example.demo.BjpaH2Application :
Starting BjpaH2Application using Java 11.0.13 on chichi with PID 15052
(D:\dev\workspace\bjpah2\target\classes started by user in D:\dev\workspace\bjpah2)
2022-07-13 13:31:17.898 INFO 15052 --- [ restartedMain] com.example.demo.BjpaH2Application : No
active profile set, falling back to 1 default profile: "default"
2022-07-13 13:31:17.937 INFO 15052 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor :
Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
2022-07-13 13:31:17.937 INFO 15052 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For
additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
2022-07-13 13:31:18.273 INFO 15052 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate :
Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-07-13 13:31:18.302 INFO 15052 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate :
Finished Spring Data repository scanning in 23 ms. Found 1 JPA repository interfaces.
2022-07-13 13:31:19.229 INFO 15052 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer :
Tomcat initialized with port(s): 80 (http)
2022-07-13 13:31:19.236 INFO 15052 --- [ restartedMain] o.apache.catalina.core.StandardService :
Starting service [Tomcat]
2022-07-13 13:31:19.236 INFO 15052 --- [ restartedMain] org.apache.catalina.core.StandardEngine :
Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-07-13 13:31:19.306 INFO 15052 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] :
Initializing Spring embedded WebApplicationContext
2022-07-13 13:31:19.307 INFO 15052 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root
WebApplicationContext: initialization completed in 1370 ms
2022-07-13 13:31:19.324 INFO 15052 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource :
HikariPool-1 - Starting...
2022-07-13 13:31:19.462 INFO 15052 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource :
HikariPool-1 - Start completed.
2022-07-13 13:31:19.471 INFO 15052 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2
console available at '/h2-console'. Database available at
'jdbc:h2:mem:0e41f7eb-d5ff-42cb-bdd8-4bc2a98c2bf6'
2022-07-13 13:31:19.590 INFO 15052 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper :
HHH000204: Processing PersistenceUnitInfo [name: default]
2022-07-13 13:31:19.624 INFO 15052 --- [ restartedMain] org.hibernate.Version :
HHH0000412: Hibernate ORM core version 5.6.9.Final
2022-07-13 13:31:19.735 INFO 15052 --- [ restartedMain] o.hibernate.annotations.common.Version :
HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-07-13 13:31:19.803 INFO 15052 --- [ restartedMain] org.hibernate.dialect.Dialect :
HHH0000400: Using dialect: org.hibernate.dialect.H2Dialect
Hibernate:

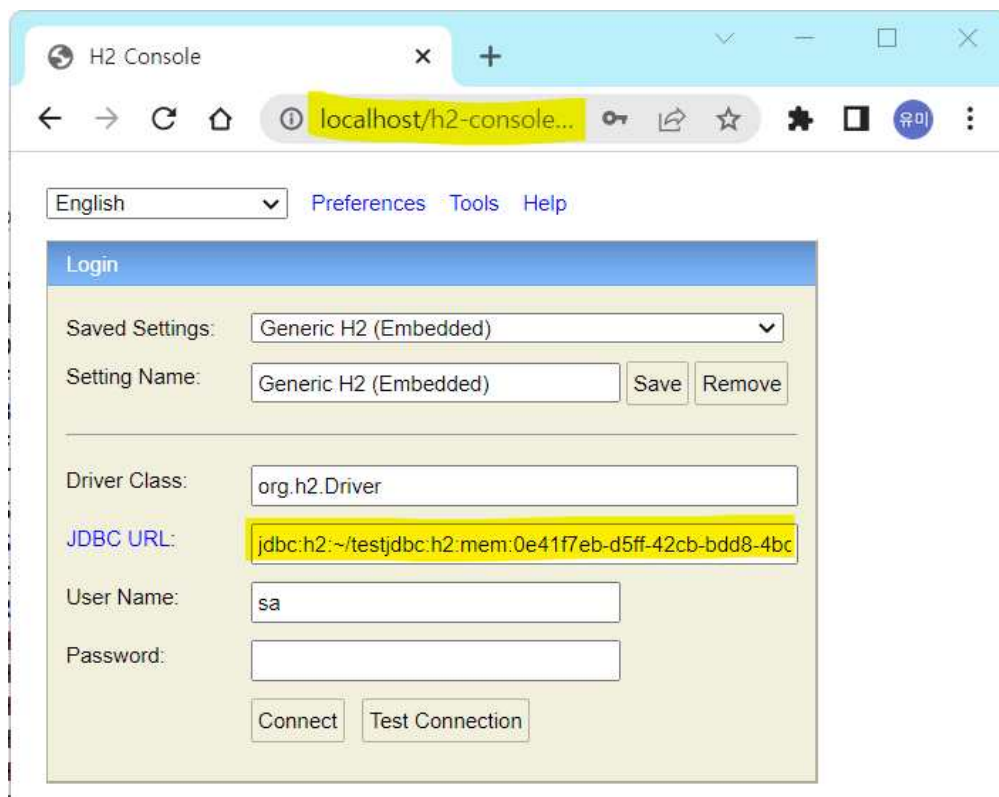
    drop table if exists customer CASCADE
Hibernate:

    drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

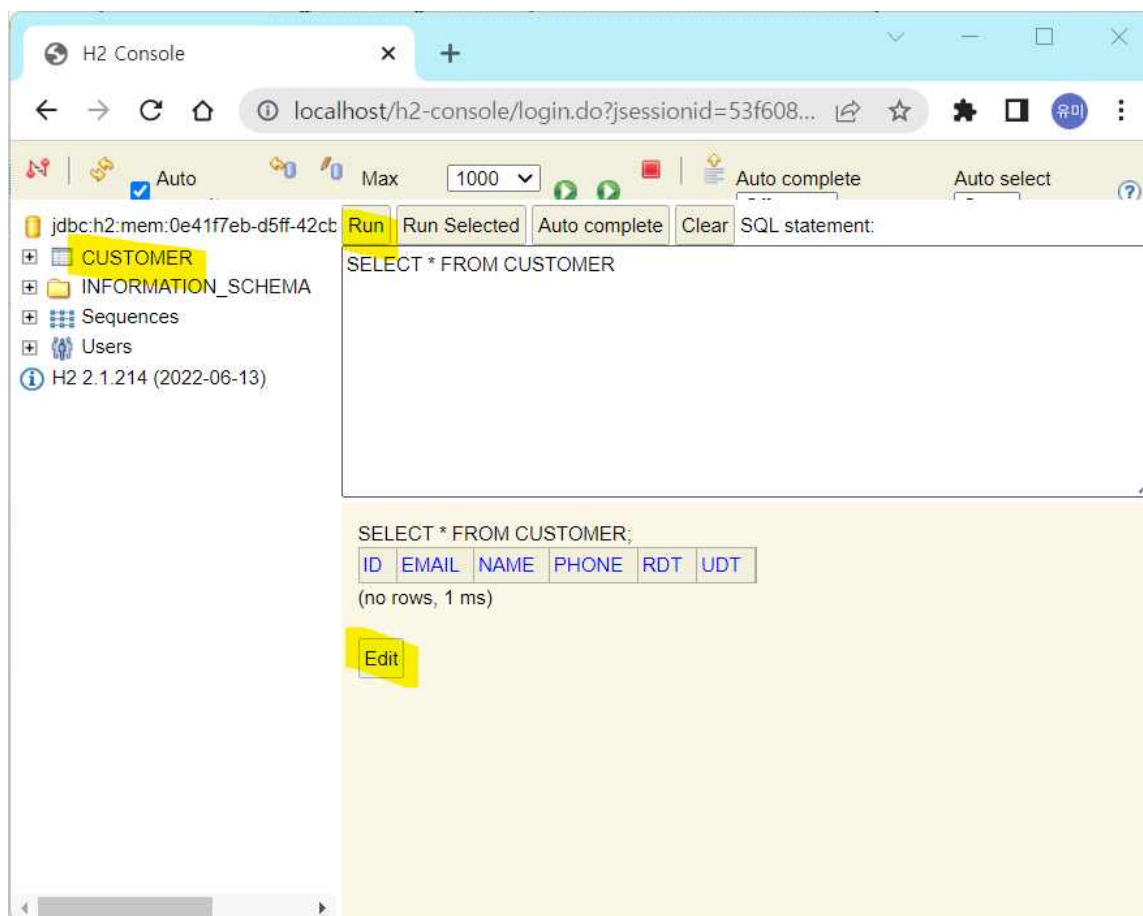
    create table customer (
        id bigint not null,
        email varchar(255),
        name varchar(20) not null,
        phone varchar(20) not null,
        rdt timestamp,
        udt timestamp,
        primary key (id)
    )
Hibernate:
    alter table customer
        add constraint UK_o3uty20c6csmx5y4uk2tc5r4m unique (phone)
2022-07-13 13:31:20.111 INFO 15052 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator :
HHH000490: Using JtaPlatform implementation:
[org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-07-13 13:31:20.116 INFO 15052 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean :
Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-07-13 13:31:20.303 WARN 15052 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration :
spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view
rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2022-07-13 13:31:20.650 INFO 15052 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer :
LiveReload server is running on port 35729
2022-07-13 13:31:20.686 INFO 15052 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer :
Tomcat started on port(s): 80 (http) with context path ''
2022-07-13 13:31:20.697 INFO 15052 --- [ restartedMain] com.example.demo.BjpaH2Application :
Started BjpaH2Application in 3.042 seconds (JVM running for 3.381)

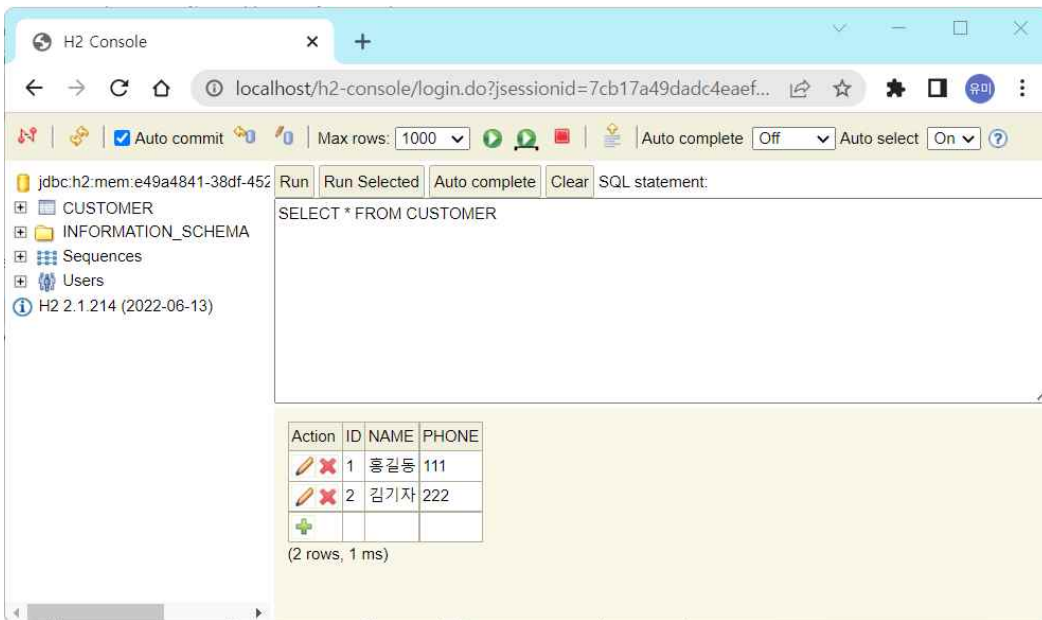
```

## - H2 db 콘솔 시작

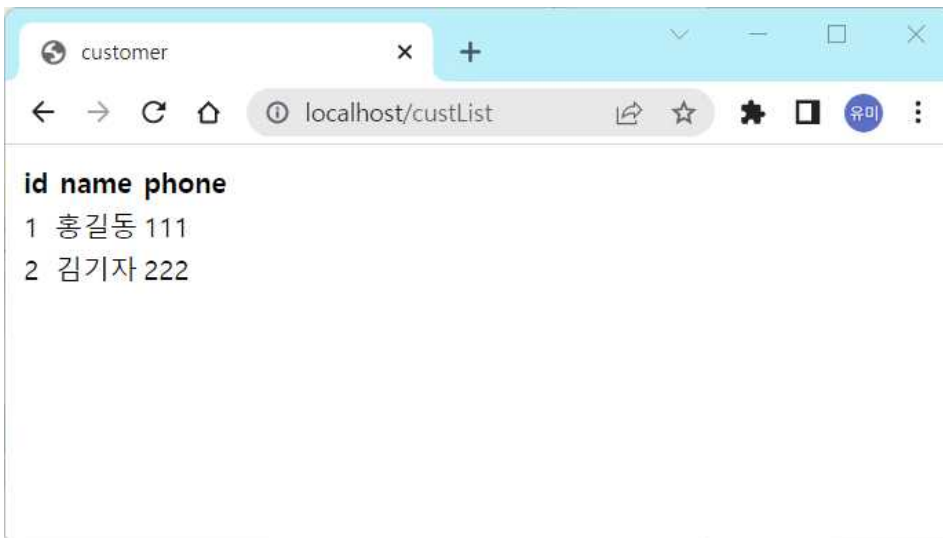


## - 데이터 입력





- 브라우저에서 테스트



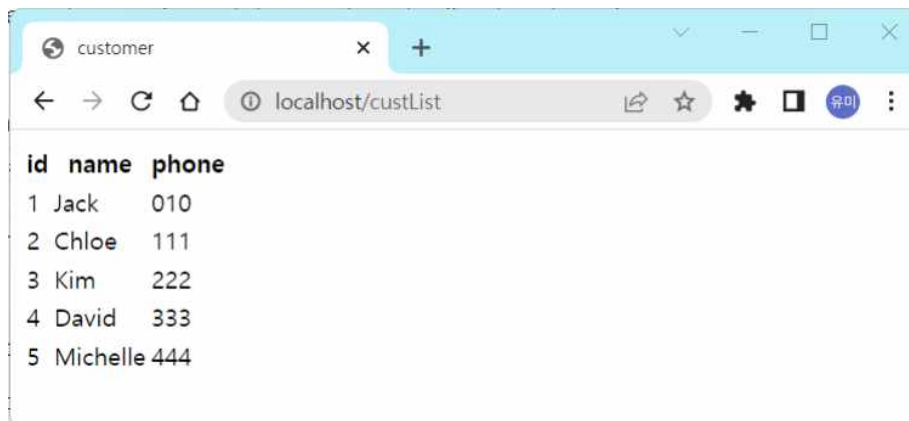
#### 4. 서버 시작할 때 샘플 데이터 입력

```

private static final Logger log = LoggerFactory.getLogger(BjpaH2Application.class);

@Bean
public CommandLineRunner demo(CustomerRepository repository) {
    return (args) -> {
        // save a few customers
        repository.save(new Customer("Jack", "010"));
        repository.save(new Customer("Chloe", "111"));
        repository.save(new Customer("Kim", "222"));
        repository.save(new Customer("David", "333"));
        repository.save(new Customer("Michelle", "444"));

        // fetch all customers
        log.info("Customers found with findAll():");
        log.info("-----");
        for (Customer customer : repository.findAll()) {
            log.info(customer.toString());
        }
        log.info("");
    }
}
  
```



A screenshot of a web browser window. The tab is labeled 'customer'. The address bar shows 'localhost/custList'. The page content displays a table with three columns: 'id', 'name', and 'phone'. The table contains five rows of data.

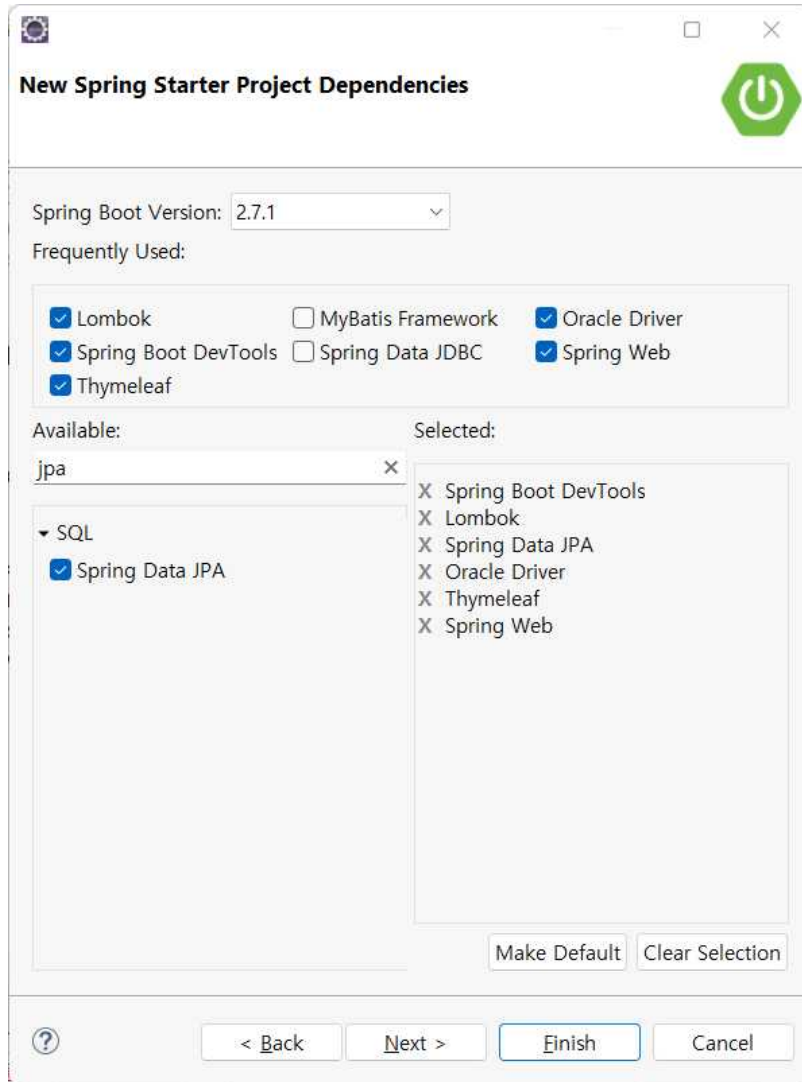
id	name	phone
1	Jack	010
2	Chloe	111
3	Kim	222
4	David	333
5	Michelle	444



## 2. 스프링부트웹 JPA, ORACLE 이용

### 가. 프로젝트 생성

Devtools, Lombok, JPA, Oracle, Thymeleaf, Web 라이브러리 선택



### 나. 환경설정

```
#tomcat 설정
server.port=80

# jpa
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true
#spring.jpa.hibernate.ddl-auto=create
spring.jpa.generate-ddl=true
spring.jpa.show-sql=true

#jpa에서 oracle 사용
spring.jpa.database=oracle
spring.jpa.database-platform=org.hibernate.dialect.Oracle10gDialect
#logging.level.org.hibernate=info

#datasource (oracle)
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521/xe
spring.datasource.username=hr
spring.datasource.password=hr
```

## 다. Entity 클래스

```
package com.example.demo;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import lombok.Data;

@Data
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(length = 20, nullable = false)
    private String name;

    @Column(length = 20, nullable = false, unique = true)
    private String phone;
}
```

- spring.jpa.generate-ddl 설정이 "true"이면 서버 시작 시 엔티티 클래스에 해당하는 테이블이 없으면 생성됨.

Hibernate:

```
create table customer (
  id number(19,0) not null,
  name varchar2(20 char) not null,
  phone varchar2(20 char) not null,
  primary key (id)
)
```

Hibernate:

```
alter table customer
  drop constraint UK_o3uty20c6csmx5y4uk2tc5r4m
```

Hibernate:

```
alter table customer
  add constraint UK_o3uty20c6csmx5y4uk2tc5r4m unique (phone)
```

- vo에 필드 추가

```
public class Customer {
    ....

    private String email;

    @CreationTimestamp
    private Timestamp rdt;

    @UpdateTimestamp
    private Timestamp udt;
}
```

- spring.jpa.generate-ddl 설정이 true 이면 엔티티 클래스에 필드가 변경되면 update 구문이 실행됨

Hibernate:

```
alter table customer
  add email varchar2(255 char)
```

Hibernate:

```
alter table customer
  add rdt timestamp
```

Hibernate:

```
alter table customer
  add udt timestamp
```

- 테이블 생성 확인

```
SQL> desc customer
Name                                     Null?   Type
-----
ID                                       NOT NULL NUMBER(19)
NAME                                   NOT NULL VARCHAR2(20 CHAR)
PHONE                                 NOT NULL VARCHAR2(20 CHAR)
EMAIL                                VARCHAR2(255 CHAR)
RDT                                  TIMESTAMP(6)
UDT                                  TIMESTAMP(6)
```

- spring.jpa.generate-ddl 설정을 "false" 변경

## 라. Repository 인터페이스

```
package com.example.demo;

import java.util.List;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

@Repository
public interface CustomerRepository extends JpaRepository<Customer, Long>{ }
```

### (1) 등록 테스트

```
package com.example.demo;
import static org.hamcrest.CoreMatchers.is;
import static org.hamcrest.MatcherAssert.assertThat;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import com.example.demo.customer.Customer;
import com.example.demo.customer.CustomerRepository;

@SpringBootTest
public class CustomerRepositoryClient {

    @Autowired CustomerRepository custRepo;

    @Test
    public void insert() {
        Customer cust = new Customer();
        cust.setName("홍길동");
        cust.setPhone("0101111");
        Customer result = custRepo.save(cust);
        assertThat( result.getName(), is("홍길동") );
    }
}
```

```
Hibernate:
select
  hibernate_sequence.nextval
from
  dual
Hibernate:
/* insert com.example.demo.customer.Customer
*/ insert
into
  customer
(email, name, phone, rdt, udt, id)
values
  (?, ?, ?, ?, ?, ?)
```

## (2) 수정 테스트

```

@Test
public void update() {
    Optional<Customer> opcust = custRepo.findById(1L);
    if( ! opcust.isEmpty() ) {
        Customer cust = opcust.get();
        cust.setPhone("0102222");
        Customer result = custRepo.save(cust);
        System.out.println(result);
    }
}

```

```

Hibernate:
/* load com.example.demo.customer.Customer */ select
customer0_id as id1_0_0_,
customer0_email as email2_0_0_,
customer0_name as name3_0_0_,
customer0_phone as phone4_0_0_,
customer0_rdt as rdt5_0_0_,
customer0_udt as udt6_0_0_
from
customer customer0_
where
customer0_id=?
Hibernate:
/* update
com.example.demo.customer.Customer */ update
customer
set
email=?,
name=?,
phone=?,
rdt=?,
udt=?
where
id=?

```

## (3) 리포지토리 메서드 추가

- repository에서 메서드 추가하면 실행시 자동으로 쿼리를 생성함.

```

// long 이 아니라 Long으로 작성. ex) int => Integer 같이 primitive형식 사용못함
public List<Customer> findByName(String name);
public List<Customer> findByPhone(String phone);
//like검색도 가능
public List<Customer> findByNameLike(String keyword);

@Query("select u from Customer u where u.name like ?1%")
List<Customer> findByNameAndSort(String name);

@Transactional
@Modifying
@Query("update Customer u set u.name = ?1 where u.id = ?2")
int setFixedNameFor(String name, Long id);

```

참고사이트 :

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

마. 페이징 조회

바. 일대일

사. 일대다

### 3. JPA

#### 가. 개요

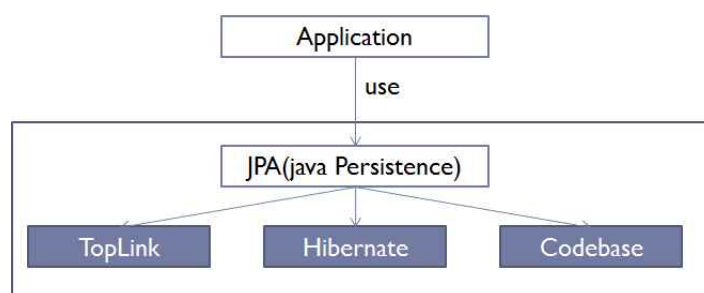
##### (1) JPA(Java Persistence API)

JPA는 기술명세이다. JPA는 자바 ORM 기술에 대한 API 표준 명세를 의미한다

Java Persistence API의 약자로 자바 어플리케이션에서 관계형 데이터베이스를 사용하는 방식을 정의한 인터페이스이다. 자바 어플리케이션에서 관계형 데이터베이스를 어떻게 사용해야 하는지를 정의하는 한 방법일 뿐이다.

JPA를 사용하기 위해서는 JPA를 구현한 Hibernate 및 EclipseLink, DataNucleus와 같은 ORM 프레임워크를 사용해야 한다.

JDBC가 특정 DBMS에 종속되지 않는 DB 연동 구현을 지원하는 것처럼 JPA API를 이용하면 소스 수정 없이 ORM 프레임워크를 교체할 수 있다.



##### (2) JPA 등장 배경

- 기존 SQL에 의존적이고 중심적인 개발 시 불편( 유사한 CURD SQL 반복 작업)
- 쿼리가 변경되면 프로그램 소스 DTO 객체도 변경(DAO와 테이블의 강한 의존성)
- 데이터를 가져와서 객체지향적인 관계를 Mapping하는 일이 빈번( 객체를 단순히 데이터 전달 목적으로 사용할 뿐, 객체 지향적이지 못함 ( 페러다임 불일치 )

##### (3) 장점

- 객체지향적으로 데이터를 관리할 수 있기 때문에 비즈니스 로직에 집중
- 테이블 생성, 변경, 관리가 쉽다. 로직을 쿼리에 집중하기 보다는 객체 자체에 집중할 수 있다.
- query를 직접 작성하지 않고 메서드 호출만으로 query가 수행되다 보니, ORM을 사용하면 생산성이 매우 높다.
- 특정 벤더에 종속적이지 않다.(JPA는 추상화된 접근 계층을 제공하기 때문에 특정 벤더에 종속적이지 않다. DB 변경이 수월)

##### (4) 단점

- 그러나 query가 복잡해지면 ORM으로 표현하는데 한계가 있고, 성능이 raw query에 비해 느리다.
- 복잡한 통계 분석 쿼리를 메서드만으로 해결하는 것은 힘들다. 이것을 보완하기 위해 SQL과 유사한 기술인 JPQL을 지원한다.

##### (5) Hibernate

참고사이트: <https://www.slideshare.net/visualkhh/hibernate-start>

Hibernate는 JPA라는 명세의 구현체이다. 즉, 위에서 언급한 javax.persistence.EntityManager와 같은 인터페이스를 직접 구현한 라이브러리이다. JPA와 Hibernate는 마치 자바의 interface와 해당 interface를 구현한 class와 같은 관계이다.

Boss에서 개발한 ORM 프레임워크이며 특정 클래스에 매핑되어야 하는 데이터베이스의 테이블에 대한 관계 정의가 되어 있는 XML 파일의 메타데이터 객체관계 매핑을 간단하게 수행시킨다. Hibernate를 사용하면 데이터베이스가 변경되더라도 SQL 스크립트를 수정하는 등의 작업을 할 필요가 없다. 애플리케이션에서 사용되는 데이터베이스를 변경시키고자 한다면 설정파일의 dialect 프로퍼티를 수정함으로써 쉽게 처리할 수 있다.

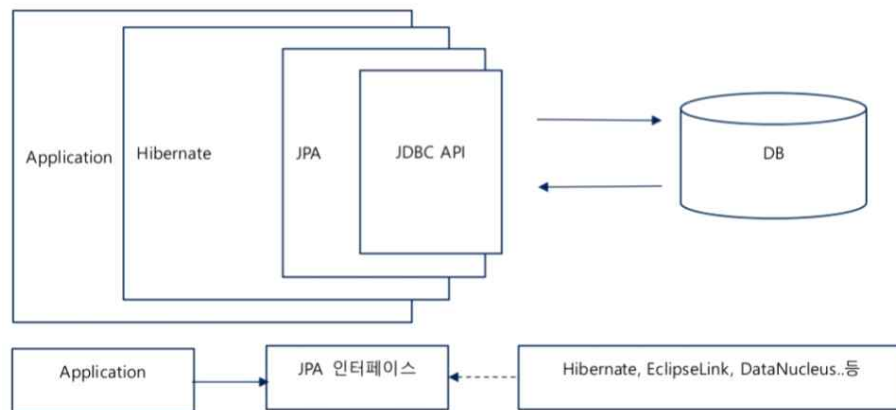
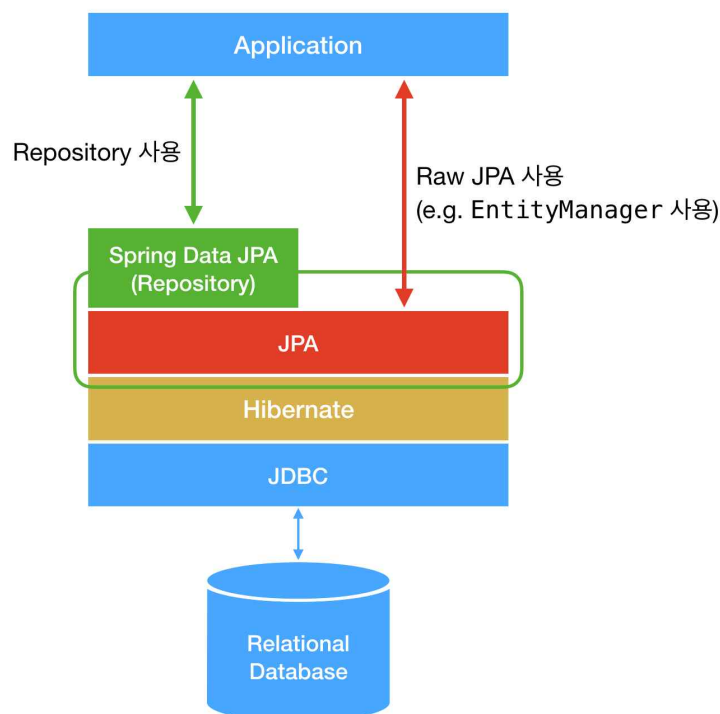


그림 12 JPA, Hibernate architecture

#### (6) Spring Data JPA

Spring Data JPA는 Spring에서 제공하는 모듈 중 하나로, 개발자가 JPA를 더 쉽고 편하게 사용할 수 있도록 도와준다. 이는 JPA를 한 단계 추상화시킨 Repository라는 인터페이스를 제공함으로써 이루어진다. 사용자가 Repository 인터페이스에 정해진 규칙대로 메소드를 입력하면, Spring이 알아서 해당 메소드 이름에 적합한 쿼리를 날리는 구현체를 만들어서 Bean으로 등록해준다.

참고사이트: <https://suhwan.dev/2019/02/24/jpa-vs-hibernate-vs-spring-data-jpa/>



Spring Data JPA가 JPA를 추상화했다는 말은, Spring Data JPA의 Repository의 구현에서 JPA를 사용하고 있다는 것이다. 예를 들어, Repository 인터페이스의 기본 구현체인 SimpleJpaRepository의 코드를 보면 내부적으로 EntityManager를 사용하고 있는 것을 볼 수 있다.

## 나. jpa, hibernate 설정

### (1) 속성

속성	설명
hibernate.dialect	사용할 데이터베이스 지정
hibernate.show_sql	생성된 SQL을 콘솔에 출력한다.
hibernate.format_sql	SQL을 출력할 때 일정한 포맷으로 보기 좋게 출력한다.
hibernate.use_sql_comments	SQL에 포함된 주석도 같이 출력한다.
hibernate.id.new_generator_mappings	새로운 키 생성 전략을 사용한다.
hibernate.hbm2ddl.auto	테이블 생성이나 수정 삭제 같은 DDL 구문을 자동으로 처리할 지를 지정한다.

#### (가) hibernate.dialect 속성

특정 DBMS에 최적화된 SQL을 제공하기 위해서 DBMS마다 다른 Dialect 클래스 지정한다. DBMS가 변경되는 경우 Dialect 클래스만 변경하면 SQL이 자동으로 변경되어 생성되므로 유지보수는 크게 향상된다.

#### (나) hibernate.hbm2ddl.auto 속성

속성값	설명
create	기존 데이터베이스를 drop 한 후 엔티티 클래스에 설정된 매핑 설정을 참조하여 새로운 테이블을 생성한다.(drop + create-only를 수행한 것과 같다) classpath에 import.sql이 있을 경우 import.sql을 구동해 준다.
create-drop	create기능과 같지만 애플리케이션이 종료되기 직전에 생성된 테이블을 삭제한다.(CREATE-DROP)
create-only	데이터베이스를 새로 생성
drop	데이터베이스를 drop
update	기존에 사용중인 테이블이 있으면 새 테이블을 생성하지 않고 재사용한다. 만약 엔티티 클래스의 매핑 설정이 변경되면 변경된 내용만 반영한다.(ALTER)
validate	테이블 스키마의 Validation기능만 제공
none	기본값이며 아무 일도 일어나지 않는다.

■ hbm2ddl.auto가 update인 경우 테이블이 없으면 create table 실행하고 있는 경우는 alter table 를 실행

<pre>@Entity @Table(name = "BOARD") public class BoardDTO {     private String img;</pre>	<p>console 실행결과 확인</p> <p>Hibernate:</p> <pre>alter table BOARD add img varchar2(255 char)</pre>
<pre>@Entity @Table(name = "MENU") public class MenuDTO {     @Id     @GeneratedValue     private Integer menu_no;     private String menu_name;     private String program_id;     //setter/getter     ... }</pre>	<p>console 실행결과 확인</p> <p>Hibernate:</p> <pre>create table MENU (     menu_no number(10,0) not null,     menu_name varchar2(255 char),     program_id varchar2(255 char),     primary key (menu_no) )</pre>

## ▣ hbm2ddl.auto가 create인 경우 drop table을 실행한 후 create table 실행

<pre> @Entity @Table(name = "MENU") public class MenuDTO {     @Id     @GeneratedValue     private Integer menu_no;     private String menu_name;     private String program_id;      //setter/getter     ... } </pre>	<p>console 실행결과 확인</p> <p>Hibernate:</p> <p>drop table MENU cascade constraints</p> <p>Hibernate:</p> <pre> create table MENU (     menu_no number(10,0) not null,     menu_name varchar2(255 char),     program_id varchar2(255 char),     primary key (menu_no) ) </pre>
--	--

## (2) Entity 클래스 작성

엔티티의 클래스. 컬럼에 대응한 프로퍼티(필드와 액세스 메서드)를 가지는 클래스

매핑 어노테이션을 생각하면 필드명을 그대로 칼럼명으로 매핑한다.

어노테이션	설 명
@Entity	@Entity 설정된 클래스를 엔티티 클래스라고 하며, @Entity 가 붙은 클래스는 테이블과 매핑된다.
@Table	엔티티와 관련된 테이블을 매핑한다. name 속성을 사용하여 BOARD 테이블과 매핑했는데 생각하면 클래스 이름이 테이블 이름과 매핑된다.
@Id	엔티티 클래스의 필수 어노테이션으로, 특정 변수를 테이블의 기본 키와 매핑한다. 예제에서는 seq 변수를 테이블의 SEQ 칼럼과 매핑했다. @Id가 없는 엔티티 클래스는 JPA가 처리하지 못한다.
@GeneratedValue	@Id가 선언된 필드에 기본 키를 자동으로 생성하여 할당할 때 사용한다. 다양한 옵션이 있지만 @GeneratedValue만 사용하면 데이터베이스에 따라서 자동으로 결정된다. H2는 시퀀스를 이용하여 처리한다.
@Temporal	날짜 타입의 변수에 선언하여 날짜 타입을 매핑할 때 사용한다. TemporalType의 DATE, TIME, TIMESTAMP 중 하나를 선택할 수 있다.
@Column	엔티티 클래스의 변수와 테이블의 컬럼을 매핑할 때 사용한다. 엔티티 클래스의 변수 이름과 컬럼 이름이 다를 때 사용하며, 생각하면 기본으로 변수 이름과 컬럼 이름을 동일하게 매핑한다.
@Transient	엔티티 클래스의 변수들 중에 테이블의 칼럼과 매핑되는 칼럼이 없거나 매핑에서 제외해야 하는 경우 사용한다. 객체에 임시로 어떤 값을 보관하고 싶을 때 사용할 수 있습니다.
@UniqueConstraint	unique key 생성
@Enumerated	자바의 enum 타입을 매핑할 때 사용

### (가) @Entity

어노테이션을 클래스 선언 부분에 부여하여 객체를 테이블과 매핑 할 엔티티라고 JPA에게 알려주는 역할을 한다. ( 엔티티 매핑 ). @Entity가 붙은 클래스는 JPA가 관리하게 된다.

@Entity를 선언할 때 몇 가지 주의 사항이 있습니다.

- 기본 생성자는 꼭 존재해야 합니다.
- final class, inner class, enum , interface에는 사용할 수 없습니다.
- 필드에 final 을 사용하면 안됩니다.

### (나) @Table

name : 매핑될 테이블 이름을 지정한다. ( 대소문자 구분없고 기본값은 엔티티의 이름 )

catalog : 데이터베이스 카탈로그(catalog)를 지정한다.



schema : 데이터베이스 스키마를 지정한다.

uniqueConstraints : 결합 unique 제약조건을 지정하며, 여러 개의 컬럼이 결합되어 유일성을 보장하는 경우 사용한다.

#### (다) @Column

name : 컬럼 이름을 지정한다.(생략 시 프로퍼티명과 동일하게 매핑)

unique : unique 제약조건을 추가한다.(기본값 : false)

nullable : null 상태 허용 여부를 설정한다.(기본값 : false)

insertable : 입력 SQL 명령어를 자동으로 생성할 때 이 컬럼을 포함할 것인지를 지정한다.(default: true)

updatable : 수정 SQL 명령어를 자동으로 생성할 때 이 컬럼을 포함할 것인지를 지정한다.(default: true)

columnDefinition : 이 컬럼에 대한 DDL 문을 직접 설정한다.

length: 문자열 타입의 컬럼 길이를 지정한다. String 타입에만 적용(기본값: 255)

precision : 숫자 타입의 전체 자릿수를 지정한다.(기본값: 0)

scale: 숫자 타입의 소수점 자릿수를 지정한다.(기본값 : 0)

#### (라) @GeneratedValue

strategy : 자동 생성 유형을 지정한다.(GenerationType 지정)

generator : 이미 생성된 Generator 이름을 지정한다.

PK값 자동 생성 전략은 TABLE, SEQUENCE, IDENTITY, AUTO 네 가지가 있다.

1. TABLE: 키 생성 전용 테이블을 만들어서 sequence처럼 사용
2. SEQUENCE: 데이터베이스 시퀀스를 사용해서 기본 키를 할당(Oracle, DB2)
3. IDENTITY: 기본키 생성을 데이터베이스에 위임(MySQL, SQL Server, DB2)
4. AUTO: DB 종류에 따라 JPA가 선택(Oracle은 SEQUENCE, MySQL은 IDENTITY를 선택)

#### (마) @Temporal

날짜 타입( java.util.Date , java.util.Calendar )을 매핑 할 때 사용합니다.

자바에서는 Camel 표기법( regDate )을 사용하고, DB에서는 snake 표기법( reg\_date )을 사용

속성

TemporalType.DATE : 날짜, 데이터베이스 date 타입과 매핑  
(ex) 2013-10-11

TemporalType.TIME : 시간, 데이터베이스 time 타입과 매핑  
(ex) 11:11:11

TemporalType.TIMESTAMP : 날짜와 시간, 데이터베이스 timestamp(datetime) 타입과 매핑  
(ex) 2013-10-11 11:11:11

#### (바) uniqueConstraints

```
@Entity
@Table(name = "MENU", uniqueConstraints= {@UniqueConstraint(columnNames=
{"menu_no","menu_name"})})
public class MenuDTO {
```

console 실행결과 확인

Hibernate:

```
alter table MENU
add constraint UKtb4vs24xojy12ks2ph3l8jkq5 unique (menu_no, menu_name)
```

### (3) DAO 클래스 작성

EntityManager를 통해서 엔티티를 취득하거나 갱신하면 내부에서 RDB에 액세스가 일어난다.

EntityManager 객체가 제공하는 CRUD 기능의 메서드

메서드	설명
<code>persist(Object entity)</code>	엔티티를 영속한다.(INSERT)
<code>merge(Object entity)</code>	준영속 상태의 엔티티를 영속한다.(UPDATE)
<code>remove(Object entity)</code>	영속 상태의 엔티티를 제거한다.(DELETE)
<code>find(Class&lt;T&gt; entityClass, Object promaryKey)</code>	하나의 엔티티를 검색한다.(SELECT ONE)
<code>createQuery(String qString, Class&lt;T&gt; resultClass)</code>	JPQL에 해당하는 엔티티 목록을 검색한다.(SELECT LIST)

## 다. JPQL

JPQL의 탄생 배경은 JPA에서 제공하는 메서드 호출만으로 섬세한 쿼리 작성이 어렵다는 것에 있습니다. JPQL(Java Persistence Query Language)은 JPA를 구현한 프레임워크에서 사용하는 언어입니다. JPA 구현 프레임워크에서는 JPQL을 SQL로 변환해 데이터베이스에 질의하게 됩니다. JPQL은 테이블을 대상으로 쿼리하지 않고 객체를 고려해 쿼리합니다. 이 때문에 JPQL은 데이터베이스 테이블에 직접적인 의존 관계를 맺고 있지 않습니다. JPQL은 SQL과 비슷한 구조로 구성 되었습니다.

```
select b
from guestbook
where b.writer = :writer
order by r.no desc
```

관련사이트:

jpql: [https://docs.oracle.com/html/E13946\\_01/ejb3\\_langref.html](https://docs.oracle.com/html/E13946_01/ejb3_langref.html)

QueryDSL: [http://www.querydsl.com/static/querydsl/3.4.3/reference/ko-KR/html\\_single/](http://www.querydsl.com/static/querydsl/3.4.3/reference/ko-KR/html_single/)

1. Criteria 쿼리 : JPQL을 편하게 작성하도록 도와주는 API. 빌더 클래스 모음.
2. 네이티브 SQL : JPA에서 JPQL대신 직접 SQL을 사용할 수 있다.
3. QueryDSL : Criteria 쿼리처럼 JPQL을 편하게 작성하도록 도와주는 빌더 클래스 모음. 비표준 오픈소스 프레임워크.
4. JDBC직접이용 : MyBatis같은 SQL 매퍼 프레임워크 사용. 필요하면 JDBC를 직접 사용가능함.

## 4. 부록

### 가. H2 Database

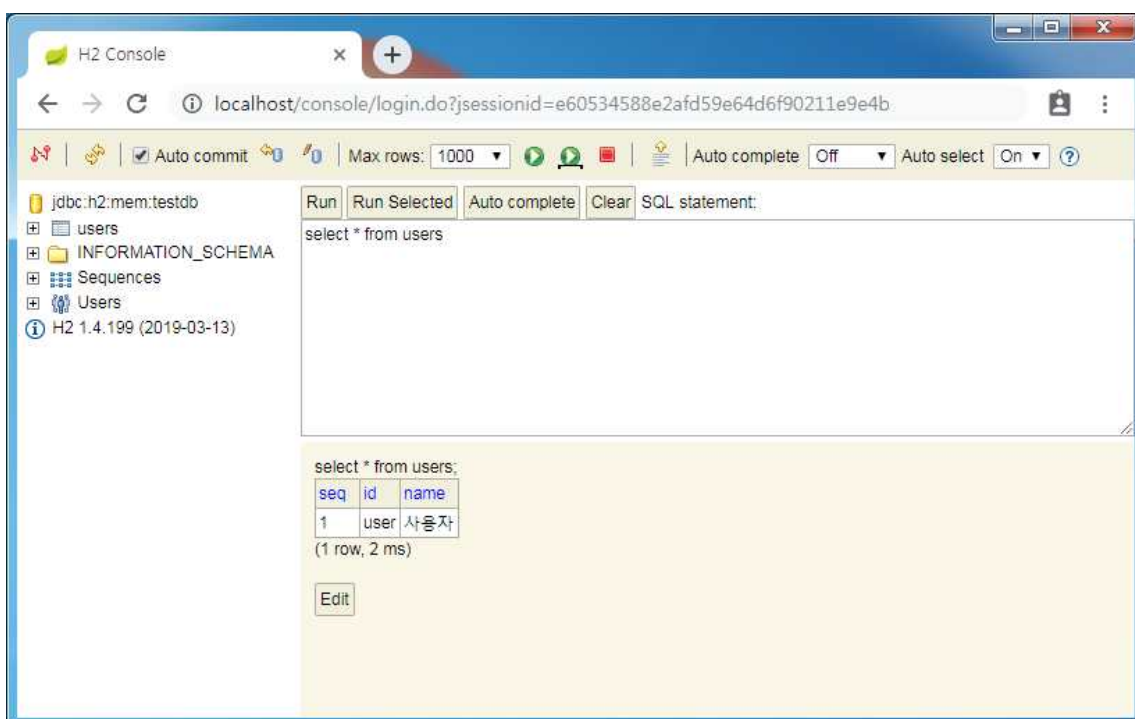
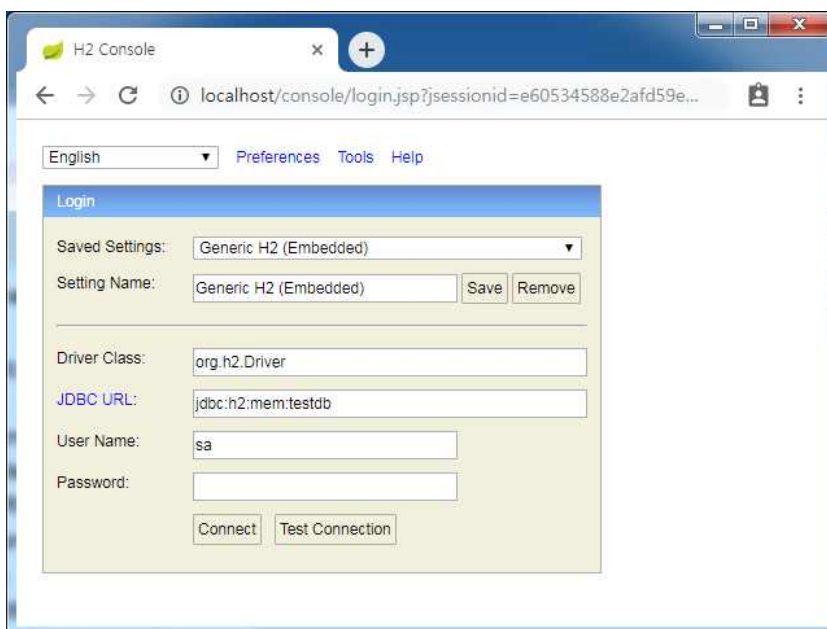
- 인-메모리(in-memory), 파일, TCP 지원 데이터베이스
- JDBC URL 설정으로 데이터베이스 작동방식 지정가능
- 스프링 부트 자동구성으로 /h2-console h2 webconsole 제공
- 로컬 개발환경에서 별도의 DB 설치없이 빠른 프로토타이핑 지원
- 필요에 따라 운영가능한 수준의 데이터베이스 활용가능

#### (1) 설치

참고사이트: <https://www.h2database.com>

에러코드: <http://h2database.com/javadoc/org/h2/api/ErrorCode.html>

#### (2) h2 console



### (3) spring boot 설정

#### (가) application.properties

```
server.port = 80
spring.datasource.url=jdbc:h2:tcp://localhost/~ /test;MODE=ORACLE;DATABASE_TO_UPPER=FALSE;
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

spring.h2.console.enabled=true
spring.h2.console.path=/console
#spring.jpa.hibernate.ddl-auto=update
```