¿ Qué es H2 ?

H2 es una base de datos relacional llamada *In memory database* (Base de datos en memoria), esto significa que los datos solo vivirán durante la ejecución de nuestra aplicación y cuando esta termine se perderán. El uso de este tipo de bases de datos es muy común para desarrollar pruebas de concepto y realizar pruebas unitarias.

Las bases de datos en memoria son diferentes a las bases de datos normales por lo siguiente, al usar una base de datos común hacemos lo siguiente:

La aplicación y la base de datos son independientes, esto significa que se debe crear la base de datos y mantenerla para poder tener acceso a ella. A diferencia de esto, una base de datos en memoria funcionará del siguiente modo:

Una vez que iniciemos nuestra aplicación se iniciará nuestra base de datos y esta vivirá durante el tiempo que nuestra aplicación funcione, una vez que la aplicación se detiene los datos se pierden, es por esto que el uso de este tipo de bases de datos es ideal para el desarrollo de pruebas de concepto y tests unitarios dentro de nuestras aplicaciones.

Ventajas

El uso de H2 proporciona las siguientes ventajas:

- No es necesario invertir en infraestructura
- No es necesario invertir en configuración
- No es necesario dar mantenimiento
- La configuración con Spring boot es super simple

Funcionamiento y Configuración de H2

Paso 1: *Configuración*

El primer paso será configurar nuestra aplicación con las dependencias (archivo pom.xml):

- Spring Web
- Thymeleaf
- Validation
- **H2**
- JPA

Paso 2: *Crear una Entity*

Como ejemplo utilizaremos Spring data para ver el funcionamiento de H2, para esto crearemos la siguiente Entity:

```
@Entity
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

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

    @Column(name = "nickname", nullable = true)
    private String nickName;

    @Column(name = "age", nullable = false)
    private Integer age;
....Getters and setters
}
```

La clase anterior define una entidad persona con las siguientes características:

- Un Id de tipo entero autoincremental
- Un nombre de tipo String que debe ser not null
- Un nickName de tipo String que puede ser null
- Una edad de tipo Integer que debe ser not null

Paso 3 : *Crear repository JPA*

Como sabemos Spring Data nos permite simplificar el código de acceso a base de datos a través de *Repositories*, veamos el repositorio a crear:

```
import org.springframework.data.jpa.repository.JpaRepository;

import com.devs4j.app.entities.Person;

/**
 * @author raidentrance
 *
 */
public interface PersonRepository extends JpaRepository{
}
```

Como se puede ver PersonRepository nos permitirá manejar entidades de tipo Person, más adelante veremos como utilizaremos esta interfaz en nuestro código.

Paso 4 : *Creando el servicio*

El siguiente paso será crear un servicio de Spring, estos servicios son diseñados para escribir la lógica de negocio que necesitemos*, en este caso no es tan necesario* ya que solo ejecutará la llamada a un repository, pero en casos en los que hace llamadas a multiples repositorios y ejecuta alguna lógica sobre los datos, es muy útil.

```
import java.util.List;
```

```
2
3   import org.springframework.beans.factory.annotation.Autowired;
4   import org.springframework.stereotype.Service;
5
6   import com.devs4j.app.entities.Person;
7   import com.devs4j.app.repositories.PersonRepository;
8
9   /**
10   * @author raidentrance
11   *
12   */
13  @Service
14  public class PersonService {
15
16      @Autowired
17      private PersonRepository repository;
18
19      public List getPeople() {
20          return repository.findAll();
21      }
}
```

Como se puede ver a través de la anotación @**Autowired** inyectamos el repository dentro de nuestro servicio, recordemos que no es necesario escribir la implementación del repository ya que Spring Data lo hace por nosotros, lo se es hermoso :).

Paso 5 : *Creando el controller*

El último paso para completar nuestra aplicación será crear un *Controller* para exponer la información, veamos el código a continuación:

```
1   import java.util.List;
2
3   import org.springframework.beans.factory.annotation.Autowired;
4   import org.springframework.http.HttpStatus;
5   import org.springframework.http.ResponseEntity;
6   import org.springframework.web.bind.annotation.RequestMapping;
7   import org.springframework.web.bind.annotation.ResponseBody;
8   import org.springframework.web.bind.annotation.RestController;
9
10  import com.devs4j.app.entities.Person;
11  import com.devs4j.app.services.PersonService;
12
13  /**
14   * @author raidentrance
15   *
16   */
17  @RestController
18  @RequestMapping("api")
19  public class PersonController {
20
21      @Autowired
22      private PersonService personService;
```

```
20
21          @RequestMapping("/people")
22          @ResponseBody
23          public ResponseEntity<List> getPeople() {
24              return new ResponseEntity(personService.getPeople(), HttpStatus.OK);
25          }
26
27      }
```

Como vemos nuestro endpoint /api/people devolverá todos los registros que se encuentren en la tabla person.

Trabajando con H2

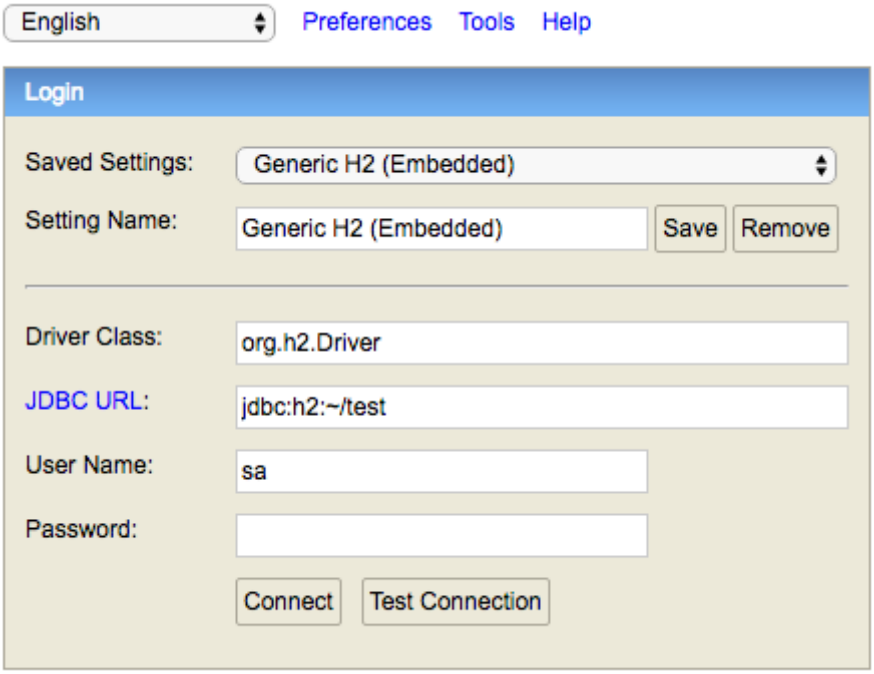
Como comentamos al principio del post, H2 es una base de datos en memoria, lo cual significa que mantendrá nuestros registros mientras nuestra aplicación esté en ejecución, lo primero que haremos será ver como acceder a la base de datos mientras la aplicación esta en ejecución.

Accediendo a la consola de H2

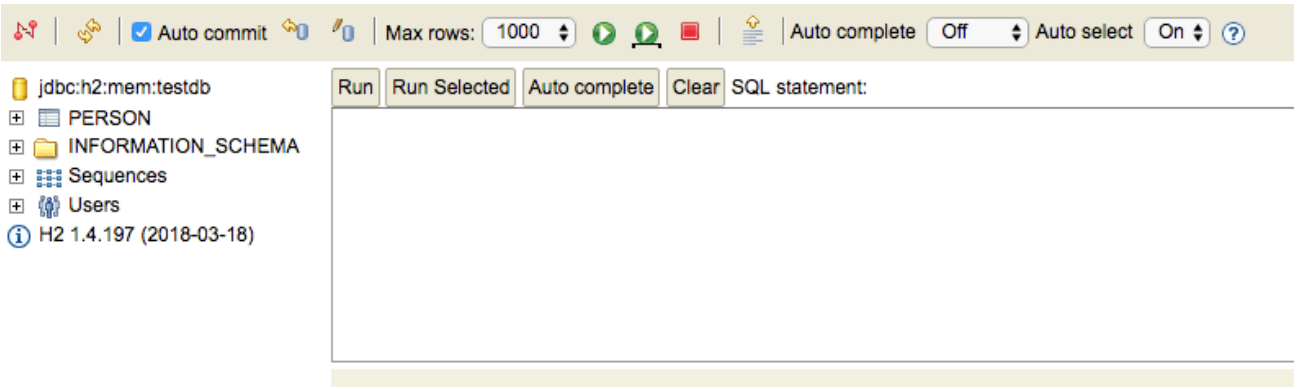
Lo primero que debemos aprender es a utilizar la consola de H2, para esto agregaremos la siguiente línea a nuestro archivo /src/main/resources/application.properties:

```
spring.thymelaf.cache = false

# This will load import.sql
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.properties.javax.persistence.validation.mode=none

# datasource: h2
spring.datasource.url=jdbc:h2:./Ej114
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.h2.console.enabled=true

logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} - %msg%n
logging.level.org.hibernate.SQL=debug
logging.level.org.hibernate.type.descriptor.sql=trace
logging.level.=info
```

Esto nos permitirá habilitar la consola de administración de H2 mientras nuestra aplicación se ejecuta. Una vez hecho esto iniciaremos nuestra aplicación y accederemos a la URL http://localhost:8080/h2-console, esto nos mostrará la siguiente vista:



Es importante cambiar la JDBC URL a **jdbc:h2:./NombreProyecto** (o **jdbc:h2:mem:testdb)** y oprimir el botón *connect*, una vez hecho esto veremos lo siguiente:



Como pueden ver hay una tabla llamada **PERSON**, nosotros no creamos esta tabla si no que fue creada de forma automática por H2 y spring boot, desde esta consola podremos ejecutar sentencias sql para agregar valores, modificarlos o realizar consultas durante la ejecución de nuestra aplicación.

Lo anterior nos da flexibilidad para realizar modificaciones en nuestros datos, pero si nuestra aplicación require que se carguen algunos scripts o que se inserten algunos valores de ejemplo tenemos otra forma de hacerlo, a través de los siguientes archivos:

- schema.sql : Permite ejecutar sentencias DDL antes de ejecutar la aplicación
- init.sql : Permite realizar sentencias DML una vez que nuestro archivo schema.sql se ejecutó de forma correcta.

Veamos un ejemplo, crearemos el archivo **/src/main/resources/data.sql** con las siguientes sentencias:

```
1  insert into person (id,name,nickname,age)values(1, 'alex','raidentrance',29);
2  insert into person (id,name,nickname,age)values(2, 'juan','juanito dubalin',80);
3  insert into person (id,name,nickname,age)values(3, 'pedro','mascara sagrada',29);
```

Como se puede ver solo son sentencias insert que generarán algunos valores de ejemplo en nuestra base de datos.

Al momento de ejecutar la aplicación **Spring boot detectó que usamos H2 y que existe un archivo llamado *data.sql* y ejecutó ese archivo en nuestra base de datos en memoria**, esto permitió que nuestro servicio contara con datos desde el principio sin necesidad de cargar ningún valor en la base de datos manualmente.