

Arquitecturas de Software para Aplicaciones Empresariales

Java Persistence API (JPA) en Spring



PROGRAMA DE INGENIERIA DE SISTEMAS

Ing. Daniel Eduardo Paz Perafán (danielp@Unicauca.edu.co)

Ing. Pablo A. Magé (pmage@Unicauca.edu.co)

El Patrón Repository

¿Qué es JPA?

¿Qué es un ORM?

¿Qué es Hibernate-ORM?

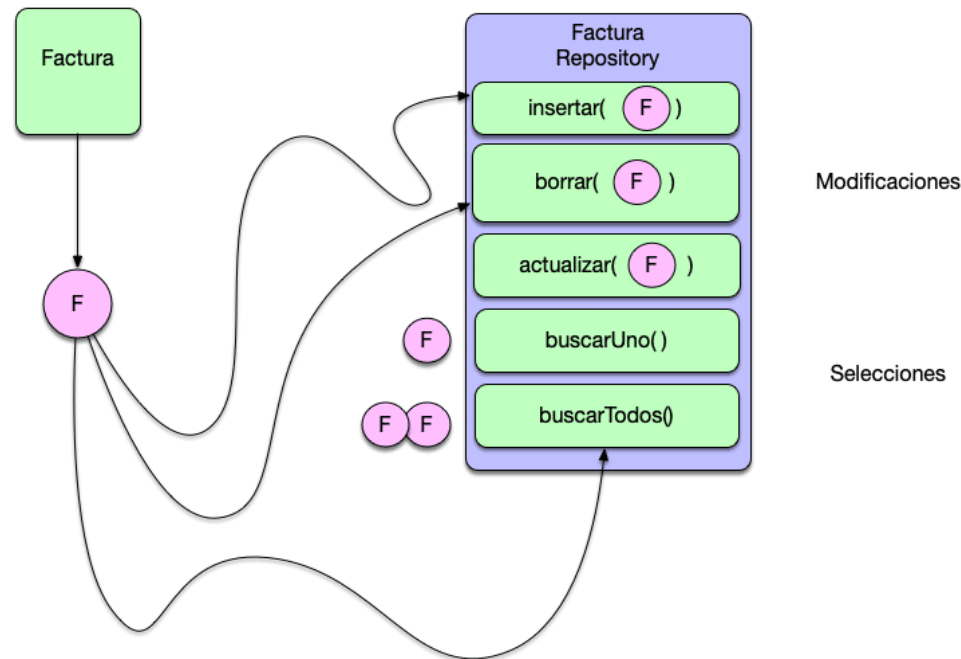
¿Qué es Spring Data JPA?

Relación entre JPA, Hibernate y Spring Data JPA

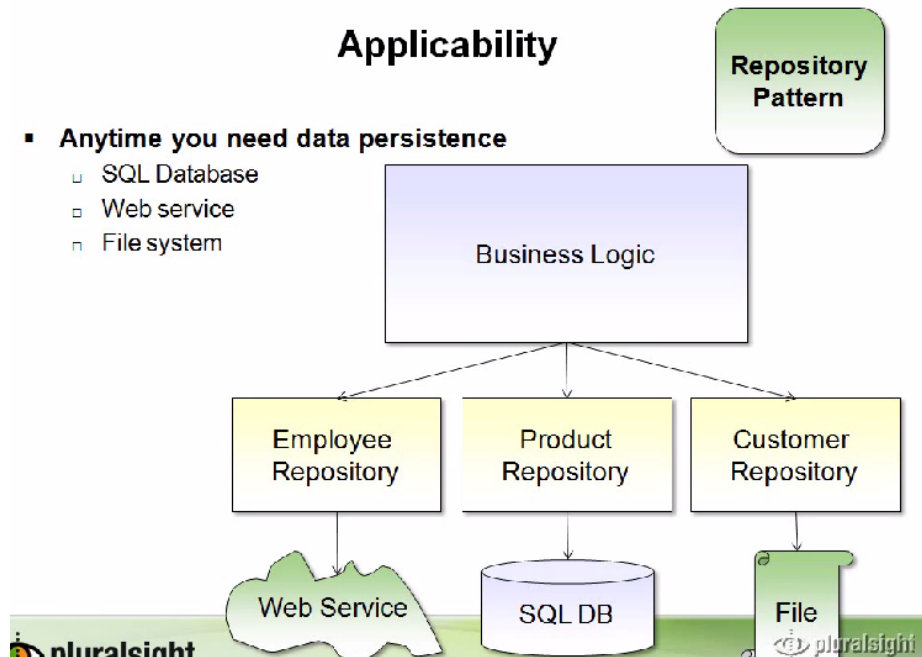
Ejemplo: Configuración e Implementación

El Patrón repositorio

- ❖ El patrón arquitectónico repositorio permite separar la lógica de acceso a datos de los objetos de negocio.
- ❖ Encapsula toda la lógica de acceso a datos al resto de la aplicación.
- ❖ El patrón Repositorio soluciona el problema de que el formato o la implementación de la fuente de datos pueda variar.



- ❖ Como podemos observar en la imagen, la lógica de negocio de nuestro cliente no interactúa directamente con la fuente de datos, ya que con este patrón la lógica de negocio es totalmente agnóstica a los datos, es decir no sabe de dónde provienen ni como se obtuvieron. Y también podemos apreciar que se comunica directamente con el repositorio el cual se encarga de hacer los mapeos necesarios y comunicarse con la fuente de datos.



Estructura si utilizamos múltiples fuentes de datos en nuestra aplicación

- ❖ Con un repositorio, puedes intercambiar los detalles de la implementación, como la migración a una biblioteca de persistencia diferente, sin afectar el código de llamada, como los modelos de vista.
- ❖ En muchos sistemas, en especial los empresariales (enterprise systems) -aquellas que realizan tareas críticas de negocio pero usualmente a una escala no tan alta- a menudo se usan herramientas de mapeo de objetos (ORM en caso de una BD relacional, ODM en caso de BD documental...) como Hibernate (en Java), Eloquent o Doctrine (en PHP), TypeORM (en javascript), etc.
- ❖ Estas herramientas hacen un excelente trabajo en lidiar con complejidades que conlleva convertir tipos y formatos de datos (Objetos a Tablas o Documentos),

¿Qué es JPA?

Java Persistence API (JPA) es una especificación de **Java EE** que permite a los desarrolladores Java hacer un mapeo entre los objetos y las tablas de una base de **datos (Object Relational Mapping)** para facilitar la administración de los datos relacionales en las aplicaciones.

OBJECT

```
public class Vacante {  
  
    private Integer id;  
    private String nombre;  
    private String descripcion;  
    private Date fecha;  
    private Double salario;  
    private String estatus;  
    private Integer destacado;  
    private String imagen;  
    // getters y setters ...  
}
```

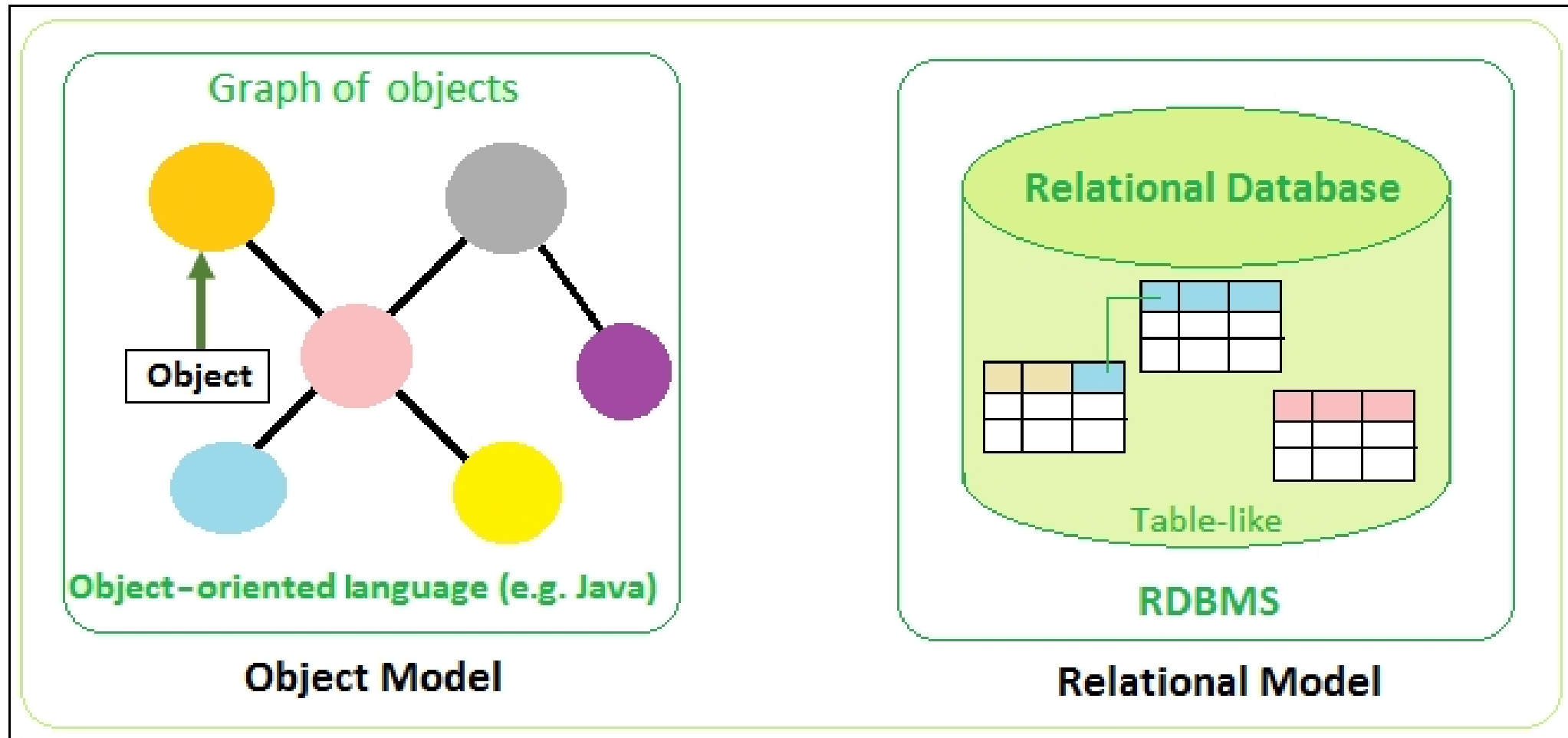
Mapeo



RELATIONAL

Vacantes	
id	INT(11)
nombre	VARCHAR(200)
descripcion	TEXT
fecha	DATE
salario	DOUBLE
estatus	ENUM(...)
destacado	INT(11)
imagen	VARCHAR(250)
Indexes	

¿Qué es un ORM?



- ❖ Un ORM es un modelo de programación que permite mapear las estructuras de una BD relacional (RDBMS Relational Data Base Management System) sobre una estructura lógica de entidades con el objeto de simplificar y acelerar el desarrollo de nuestras aplicaciones.
- ❖ Los objetos o entidades de la base de datos virtual creada en nuestro ORM podrán ser manipulados por medio de algún lenguaje de nuestro interés según el tipo de ORM utilizado, por ejemplo, LINQ sobre Entity Framework de Microsoft o JPA en JAVA.
- ❖ Las acciones CRUD a ejecutar sobre la BD física se realizan de manera indirecta por del ORM.
- ❖ El ORM será “**independiente**” de la base de datos que se este utilizando en el momento, lo cual permite cambiar de motor de base de datos según nuevas necesidades.
 - ¿En qué condiciones encaja mejor el uso de un *ORM*?
 - Si decidimos usar un *ORM* sobre un RDBMS, ¿nos “olvidamos” completamente entonces del *SQL*?

¿Qué es el Hibernate - ORM?

- ❖ Hibernate **es una herramienta de mapeo objeto-relacional (ORM)** bajo licencia GNU LGPL para Java, que facilita el mapeo de atributos en una base de datos tradicional, y el modelo de objetos de un aplicación
- ❖ Los desarrolladores de Hibernate tratan de ajustarse a las especificaciones del modelo de persistencia JPA, que van de la mano de Java EE (mantenida hasta la versión 8 por Oracle) y Jakarta EE (mantenida en la actualidad por Eclipse Foundation).
- ❖ A lo largo de su desarrollo en el tiempo, y a medida que JPA ha ido mejorando, uno de los principales objetivos de Hibernate ha sido adaptarse a la API de la que penden por defecto todos los sistemas que usan Java que usan mapeo entidad-relación



- ❖ Spring Data JPA es un módulo que ayuda a simplificar el desarrollo de la persistencia de datos utilizando el concepto de repositorios .
- ❖ En términos sencillos este módulo de Spring Data agrega una capa de abstracción al API de JPA (podríamos decir es una forma más sencilla y mejorada de trabajar con JPA).

Beneficios de Spring Data JPA

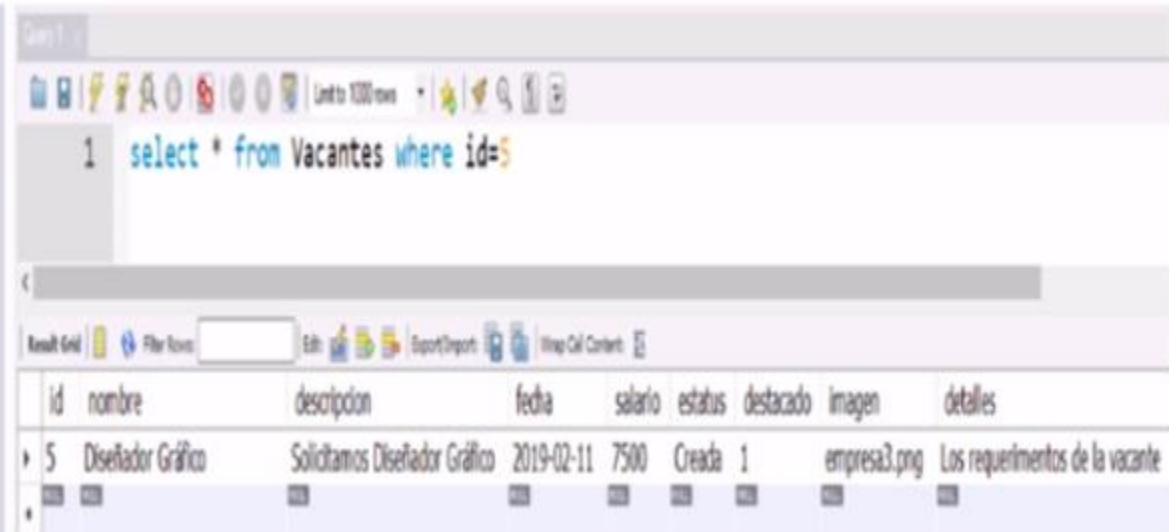
- Desarrollo ágil de la capa de persistencia de datos utilizando bases de datos relacionales
- No es necesario escribir código SQL nativo (aunque también es posible).
- Código más fácil de entender y mantener

¿Qué es un Spring Data JPA?

Ejemplo de persistencia de datos

```
// 1. Crear un objeto de tipo Vacante
Vacante vacante = new Vacante();
vacante.setId(5);
vacante.setNombre("Diseñador Gráfico");
vacante.setDescripcion("Solicitamos Diseñador Gráfico");
vacante.setFecha(sdf.parse("11-02-2019"));
vacante.setSalario(7500.0);
vacante.setDestacado(1);
vacante.setImagen("empresa3.png");
vacante.setEstatus("Creada");
vacante.setDetalles("Los requerimientos de la vacante");

// 2. Persistir (guardar) objeto en la BD
repositoryVacantes.save(vacante);
```



The screenshot shows a database query tool interface. The query entered is `select * from Vacantes where id=5`. The result is displayed in a table with the following data:

id	nombre	descripcion	fecha	salario	estatus	destacado	imagen	detalles
5	Diseñador Gráfico	Solicitamos Diseñador Gráfico	2019-02-11	7500	Creada	1	empresa3.png	Los requerimientos de la vacante

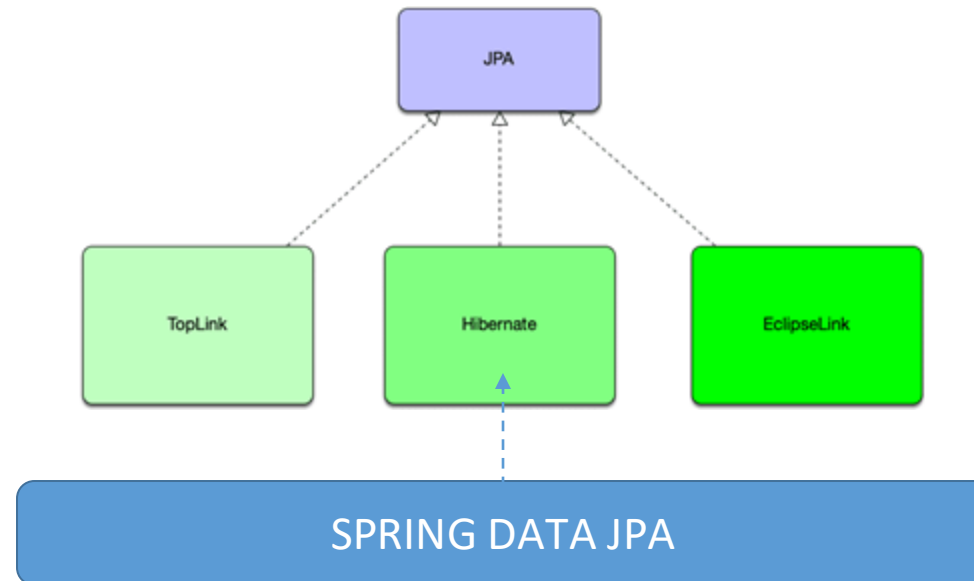
3. Se genera todo el código SQL de forma automática y se ejecuta en la base de datos.

Relación entre JPA, HIBERNATE y SPRING Data JPA

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

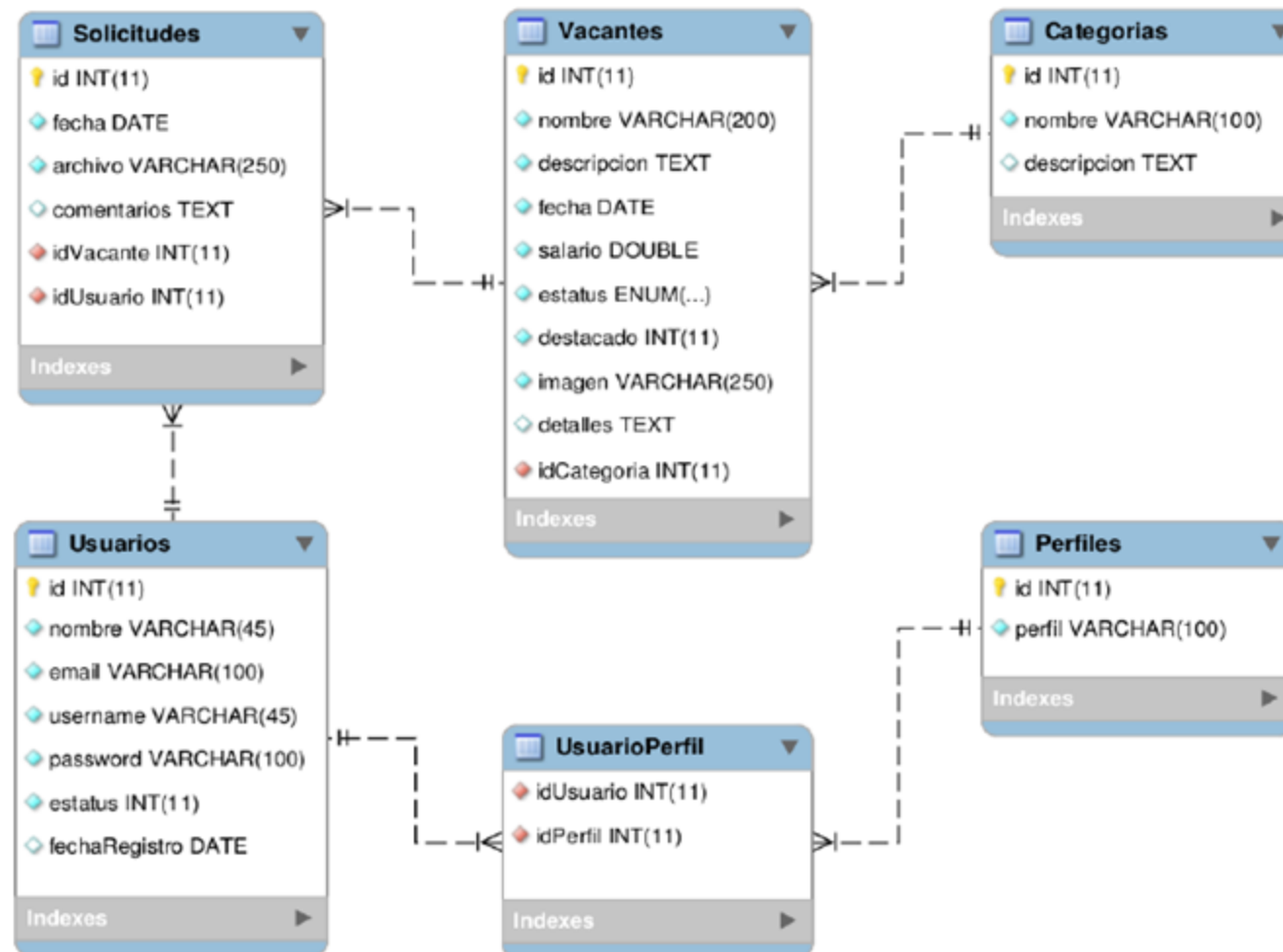
Java Persistence API es una especificación, en ella se define qué anotaciones han de usarse, como han de persistirse los objetos, como han de buscarse los objetos, cuál es su ciclo de vida. Al tratarse de un documento evidentemente no implementa nada.

Para poder trabajar con ella necesitaremos tener un Framework que implemente la especificación, uno de los más conocidos es **Hibernate**, pero hay otros como Eclipse Link, ObjectDB, TOPLink



Spring DATA JPA ofrece una capa que facilita el uso de los frameworks ORM

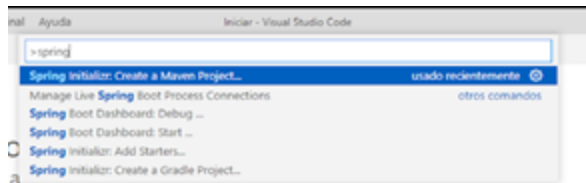
La base de datos que trabajaremos durante las siguientes sesiones es:



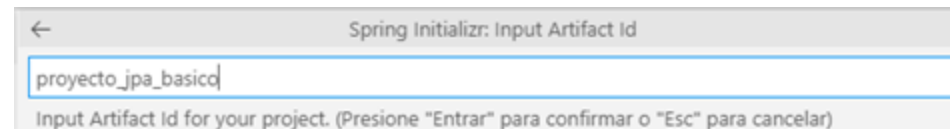
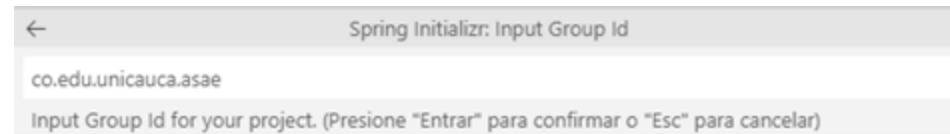
Creación de un proyecto con Spring Data JPA

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

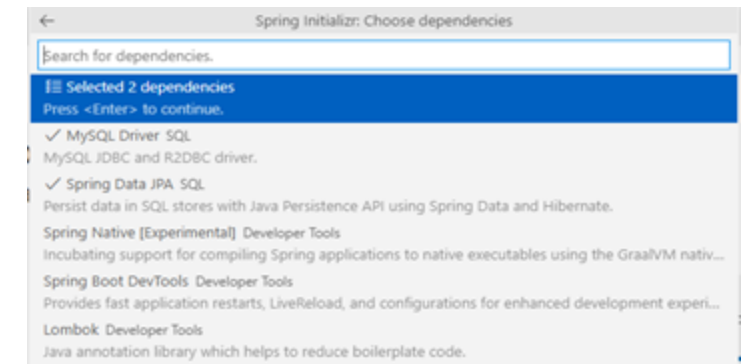
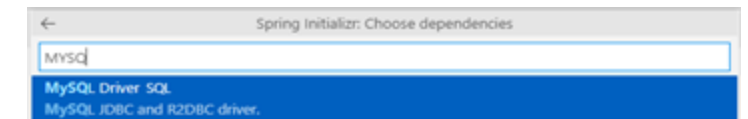
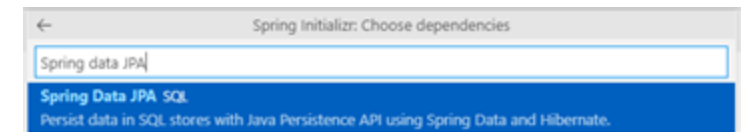
Tipo de proyecto



Configuración



Dependencias seleccionadas



Dependencias generadas en el archivo pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <scope>runtime</scope>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Dependencia que permite utilizar la especificación de JPA

Dependencia que permite conectar el repositorio a la base de datos MySQL

Las siguientes propiedades se deben configurar en el archivo `application.properties` con el fin de realizar la conexión a una base de datos MySQL 8.0:

```
# PUERTO DE ESCUCHA
a server.port=9090
# DATASOURCE (MYSQL)
b spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
c spring.datasource.url=jdbc:mysql://localhost/bdvacantes?useSSL=false&serverTimezone=GMT&allowPublicKeyRetrieval=true
spring.datasource.username=root
d spring.datasource.password=root
#JPA
e spring.jpa.generate-ddl=false
f spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
g spring.jpa.show-sql=true
# Table names physically
h spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
```

En el archivo `pom.xml` por defecto viene incluido el Driver JDBC para MySQL

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>
```


JPA tiene características para la generación de DDL, y estas se pueden configurar para que se ejecuten al inicio en el base de datos. Esto se controla a través de dos propiedades externas:

- `spring.jpa.generate-ddl` Si se establece en true, Spring Boot pedirá al proveedor de JPA respectivo que inicialice la base de datos en función de la definición de la entidad.
- `spring.jpa.hibernate.ddl-auto` Permite tener un control sobre la inicialización de la base de datos. La propiedad tomar uno de los siguientes valores: none, validate, update, create, and create-drop.

La propiedad **spring.jpa.hibernate.ddl-auto** puede tomar uno de los siguientes valores: none, validate, update, create, and create-drop.

Opcion	Efecto
none	Sin inicialización del esquema de la base de datos
create	Quita y crea el esquema al iniciar la aplicación. Con esta opción, todos sus datos desaparecerán en cada inicio.
create-drop	Crea el esquema en el inicio y destruye el esquema al cerrar el contexto. Útil para pruebas unitarias.
validate	Solo comprueba si el esquema coincide con las entidades. Si el esquema no coincide, se producirá un error en el inicio de la aplicación. No realiza ningún cambio en la base de datos.
update	Actualiza el esquema solo si es necesario. Por ejemplo, si se agregó un nuevo campo en una entidad, simplemente modificará la tabla de una nueva columna sin destruir los datos.

Al analizar el log debemos identificar 3 aspectos

```
Console
proyectoJPABasico - ProyectoJpaBasicoApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (6/10/2020 10:54:56 AM)

:: Spring Boot :: (v2.4.0-SNAPSHOT)

2020-10-06 10:54:58.755 INFO 2464 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Starting ProyectoJpaBasicoApplication using Java 1.8.0_241 on ST-LENOVO w
2020-10-06 10:54:58.758 INFO 2464 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : No active profile set, falling back to default profiles: default
2020-10-06 10:54:59.297 INFO 2464 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-10-06 10:54:59.318 INFO 2464 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 8 ms. Found 0 JPA repository
2020-10-06 10:55:00.021 INFO 2464 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-06 10:55:00.037 INFO 2464 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-10-06 10:55:00.359 INFO 2464 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-10-06 10:55:00.441 INFO 2464 --- [task-1] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-10-06 10:55:00.598 INFO 2464 --- [task-1] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.20.Final
2020-10-06 10:55:00.917 INFO 2464 --- [main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories...
2020-10-06 10:55:00.917 INFO 2464 --- [main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-10-06 10:55:00.952 INFO 2464 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Started ProyectoJpaBasicoApplication in 2.653 seconds (JVM running for 3.
2020-10-06 10:55:00.996 INFO 2464 --- [task-1] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2020-10-06 10:55:01.376 INFO 2464 --- [task-1] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
2020-10-06 10:55:01.663 INFO 2464 --- [task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2020-10-06 10:55:01.678 INFO 2464 --- [task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'

2020-10-06 10:58:48.124 INFO 6264 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applic
2020-10-06 10:58:48.125 INFO 6264 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2020-10-06 10:58:48.164 INFO 6264 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

Configuración de Spring Boot para trabajar por consola

UNIVERSIDAD DEL CAUCA – FIET
DEPARTAMENTO DE SISTEMAS

```
package co.edu.unicauca.asae.core;

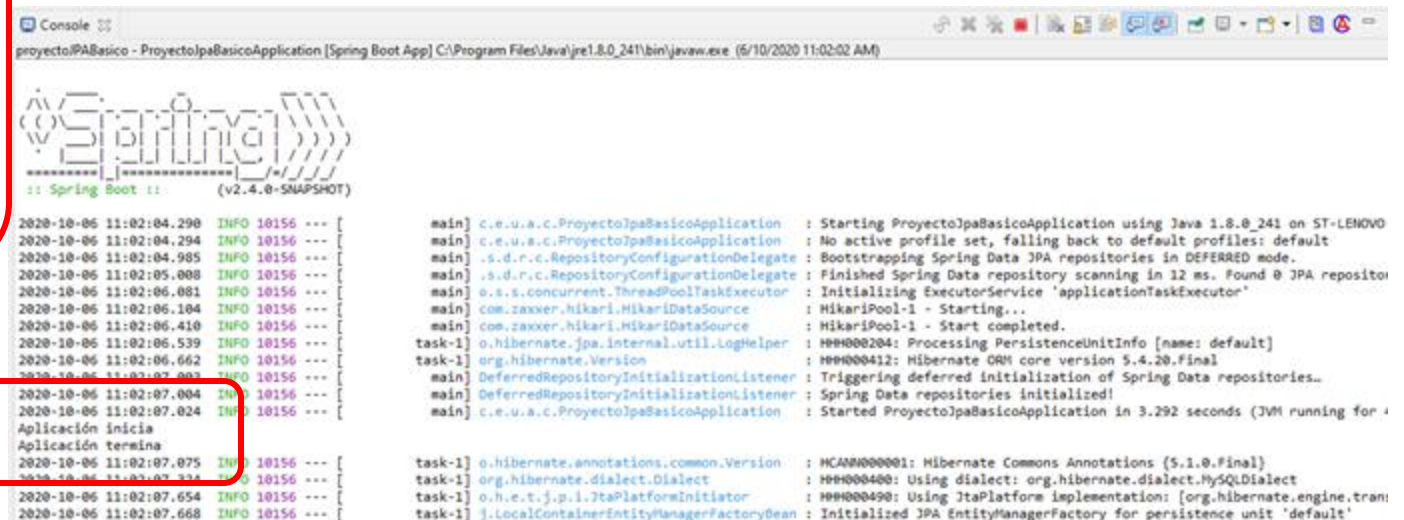
import org.springframework.boot.CommandLineRunner;

@SpringBootApplication
public class ProyectoJpaBasicoApplication implements CommandLineRunner{

    public static void main(String[] args) {
        SpringApplication.run(ProyectoJpaBasicoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println("Aplicación inicia");

        System.out.println("Aplicación termina");
    }
}
```



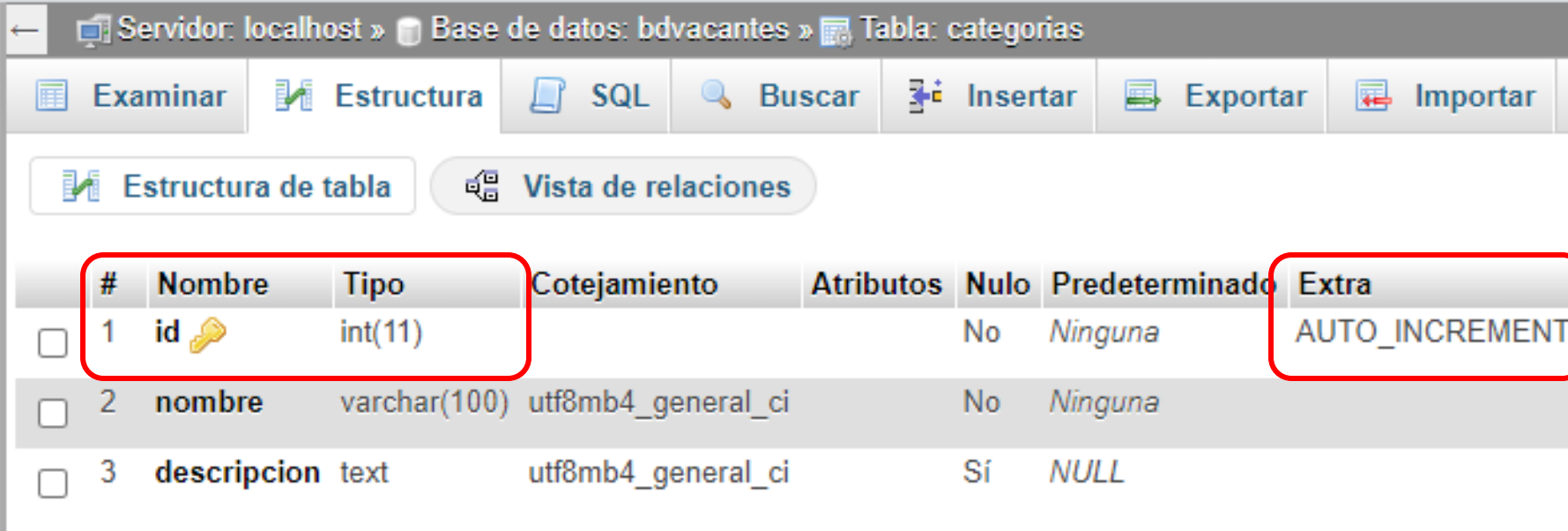
```
Console
projectoJpaBasico - ProyectoJpaBasicoApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (6/10/2020 11:02:02 AM)

:: Spring Boot ::
(v2.4.0-SNAPSHOT)


2020-10-06 11:02:04.290 INFO 10156 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Starting ProyectoJpaBasicoApplication using Java 1.8.0_241 on ST-LENOVO
2020-10-06 11:02:04.294 INFO 10156 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : No active profile set, falling back to default profiles: default
2020-10-06 11:02:04.985 INFO 10156 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-10-06 11:02:05.008 INFO 10156 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 12 ms. Found 0 JPA repository
2020-10-06 11:02:06.081 INFO 10156 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-06 11:02:06.104 INFO 10156 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-10-06 11:02:06.410 INFO 10156 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-10-06 11:02:06.539 INFO 10156 --- [task-1] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-10-06 11:02:06.662 INFO 10156 --- [task-1] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.20.Final
2020-10-06 11:02:07.003 INFO 10156 --- [main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories...
2020-10-06 11:02:07.004 INFO 10156 --- [main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-10-06 11:02:07.024 INFO 10156 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Started ProyectoJpaBasicoApplication in 3.292 seconds (JVM running for 4
Aplicación inicia
Aplicación termina
2020-10-06 11:02:07.075 INFO 10156 --- [task-1] o.hibernate.annotations.common.Version : HCANNN000001: Hibernate Commons Annotations {5.1.0.Final}
2020-10-06 11:02:07.334 INFO 10156 --- [task-1] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
2020-10-06 11:02:07.654 INFO 10156 --- [task-1] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2020-10-06 11:02:07.668 INFO 10156 --- [task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
```

Configurar el mapeo entre una clase y una tabla

Debemos analizar la estructura de la tabla



The screenshot shows a database management interface with a toolbar at the top containing buttons for 'Examinar', 'Estructura', 'SQL', 'Buscar', 'Insertar', 'Exportar', and 'Importar'. Below the toolbar, there are two tabs: 'Estructura de tabla' (selected) and 'Vista de relaciones'. The main area displays a table structure for 'categorias' with the following columns: #, Nombre, Tipo, Cotejamiento, Atributos, Nulo, Predeterminado, and Extra. The first row is highlighted with a red box, indicating the primary key 'id' of type 'int(11)' with the 'Extra' attribute 'AUTO_INCREMENT'. Three blue arrows point to the first three columns of the table structure.

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
<input type="checkbox"/> 1	id 	int(11)			No	Ninguna	AUTO_INCREMENT
<input type="checkbox"/> 2	nombre	varchar(100)	utf8mb4_general_ci		No	Ninguna	
<input type="checkbox"/> 3	descripcion	text	utf8mb4_general_ci		Sí	NULL	

Configurar el mapeo entre una clase y una tabla

Debemos utilizar 4 anotaciones **@Entity**, **@Table**, **@Id**, **@GeneratedValue**

```
package co.edu.unicauca.asae.core.modelo;

public class Categoria {

    private Integer id;
    private String nombre;
    private String descripcion;

    public Integer getId() {
    }
    public void setId(Integer id) {
    }
    public String getNombre() {
    }
    public void setNombre(String nombre) {
    }
    public String getDescripcion() {
    }
    public void setDescripcion(String descripcion) {
    }
}
```

```
package co.edu.unicauca.asae.core.modelo;

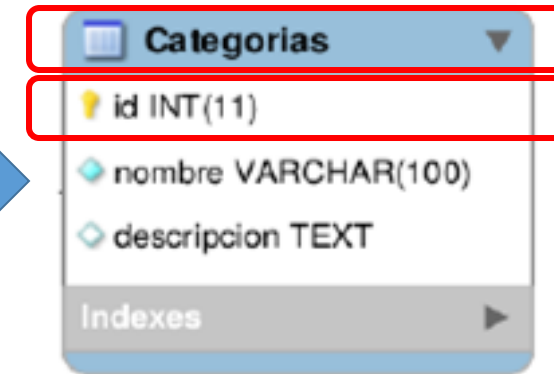
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="Categorias")
public class Categoria {

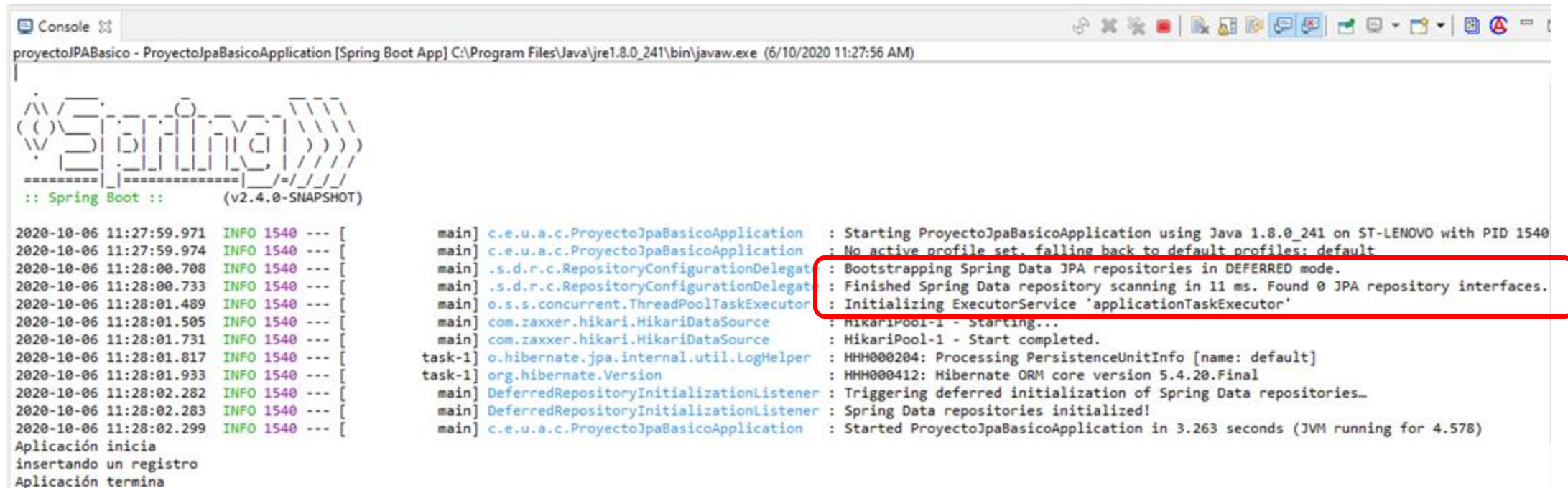
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Integer id;
    private String nombre;
    private String descripcion;

    public Integer getId() {
    }
    public void setId(Integer id) {
    }
    public String getNombre() {
    }
    public void setNombre(String nombre) {
    }
    public String getDescripcion() {
    }
    public void setDescripcion(String descripcion) {
    }
}
```

Este tipo de generación se basa en IdentityGenerator, que espera los valores generados por una columna de identidad en la base de datos. Esto significa que se incrementan automáticamente



Un repositorio es una interface que tiene definidos un conjunto de métodos para guardar, consultar, listar, eliminar, actualizar registros de una tabla de una base de datos



```
Console
proyectoJpaBasico - ProyectoJpaBasicoApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (6/10/2020 11:27:56 AM)

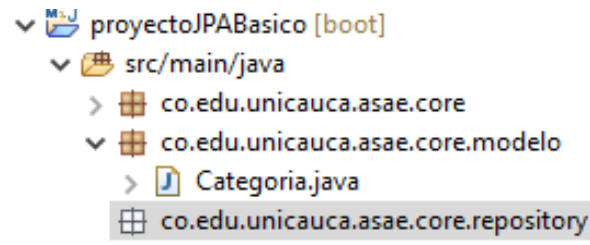
:: Spring Boot :: (v2.4.0-SNAPSHOT)

2020-10-06 11:27:59.971 INFO 1540 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Starting ProyectoJpaBasicoApplication using Java 1.8.0_241 on ST-LENOVO with PID 1540
2020-10-06 11:27:59.974 INFO 1540 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : No active profile set, falling back to default profiles: default
2020-10-06 11:28:00.708 INFO 1540 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-10-06 11:28:00.733 INFO 1540 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 11 ms. Found 0 JPA repository interfaces.
2020-10-06 11:28:01.489 INFO 1540 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-10-06 11:28:01.505 INFO 1540 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-10-06 11:28:01.731 INFO 1540 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-10-06 11:28:01.817 INFO 1540 --- [task-1] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-10-06 11:28:01.933 INFO 1540 --- [task-1] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.20.Final
2020-10-06 11:28:02.282 INFO 1540 --- [main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories...
2020-10-06 11:28:02.283 INFO 1540 --- [main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-10-06 11:28:02.299 INFO 1540 --- [main] c.e.u.a.c.ProyectoJpaBasicoApplication : Started ProyectoJpaBasicoApplication in 3.263 seconds (JVM running for 4.578)

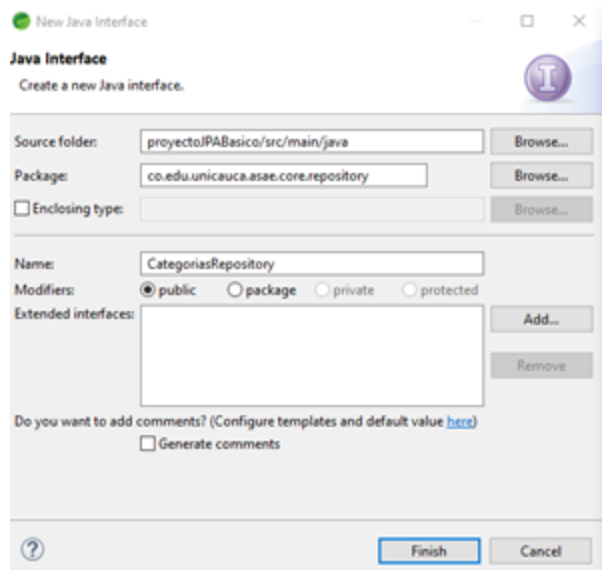
Aplicación inicia
insertando un registro
Aplicación termina
```

Interfaz CrudRepository

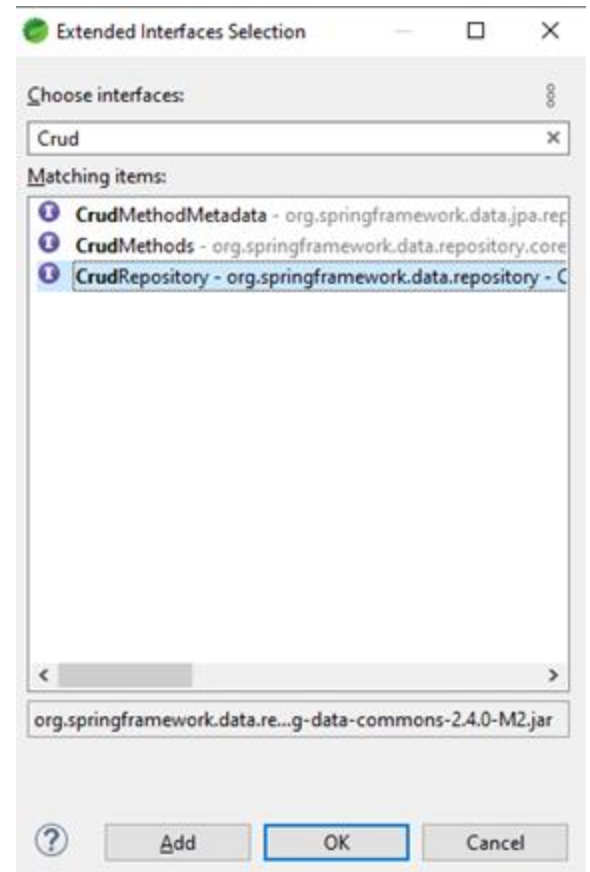
Crear un paquete repository



Crear una interface



Heradar la interface de CrudRepository



```
package co.edu.unicauca.asae.core.repository;

import org.springframework.data.repository.CrudRepository;
import co.edu.unicauca.asae.core.modelo.Categoria;

public interface CategoriasRepository extends CrudRepository<Categoria, Integer> {
}
```

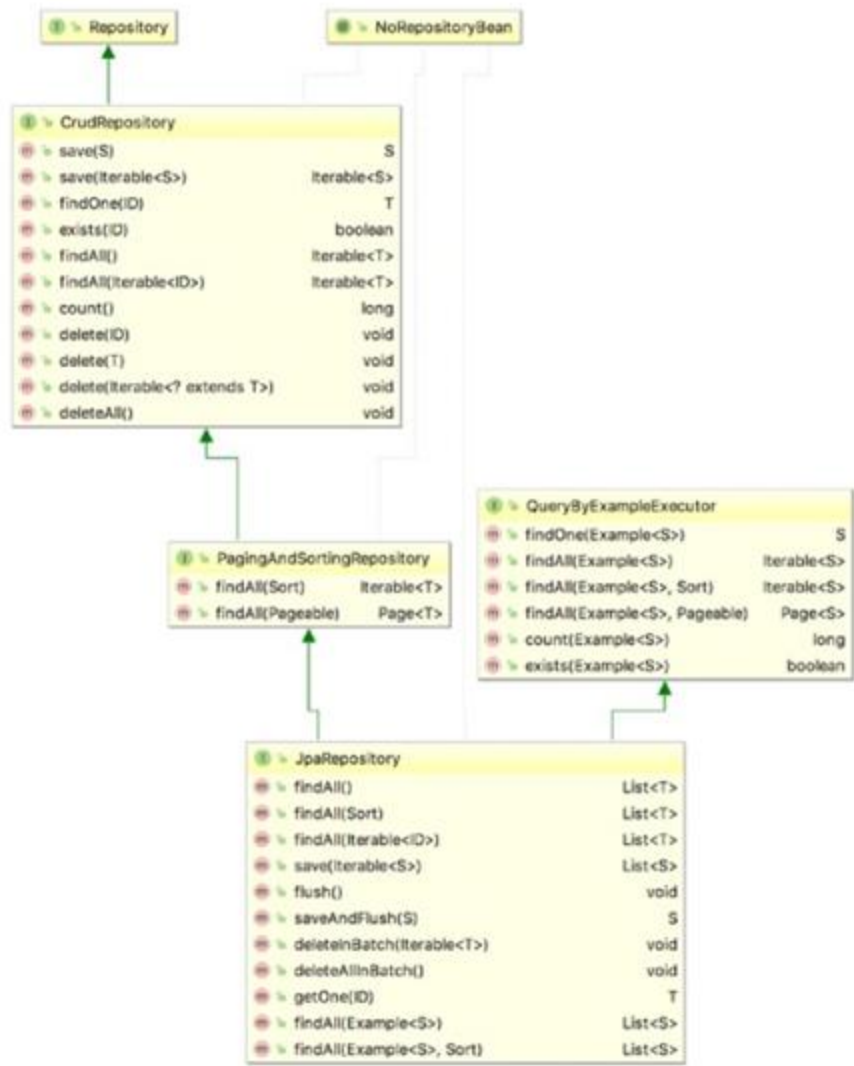
Clase asociada a una tabla



Tipo de dato de la clave primaria



Interfaz CrudRepository



La interfaz **CrudRepository** proporciona métodos para operaciones CRUD, por lo que le permite crear, leer, actualizar y eliminar registros sin tener que definir sus propios métodos.

PagingAndSortingRepository proporciona métodos adicionales para recuperar las entidades que utilizan la paginación y la clasificación.

QueryByExampleExecutor presenta más variantes del método *find ()*

JpaRepository agrega algunas funciones más específicas de JPA.

Para inyectar una instancia de nuestro repositorio debemos utilizar la notación **@autowired**

```
package co.edu.unicauca.asae.core;

import org.springframework.beans.factory.annotation.Autowired;

@SpringBootApplication
public class ProyectoJpaBasicoApplication implements CommandLineRunner{

    @Autowired
    private CategoriasRepository servicioAccesoBD;

    public static void main(String[] args) {
        SpringApplication.run(ProyectoJpaBasicoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        System.out.println(this.servicioAccesoBD);
    }

    private void guardar(){}

}
```

CommandLineRunner es una interfaz simple de Spring Boot con un método `run`. Spring Boot llamará automáticamente al método `run` de todos los beans que implementen esta interfaz después de que se haya cargado el contexto de la aplicación.



```
Console
<terminated> proyectoJpaBasico - ProyectoJpaBasicoApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_241\bin\java.exe (E/10/2020 11:48:01 AM - 11:49:09 AM)

Spring
:: Spring Boot ::
(v2.4.0-SNAPSHOT)

2020-10-06 11:48:04.989 INFO 4184 --- main c.e.u.a.c.ProyectoJpaBasicoApplication : Starting ProyectoJpaBasicoApplication using Java 1.8.0_241 on ST-LENOVO with PID 4184
2020-10-06 11:48:05.983 INFO 4184 --- main c.e.u.a.c.ProyectoJpaBasicoApplication : No active profile set, falling back to default profiles: default
2020-10-06 11:48:06.089 INFO 4184 --- main o.s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2020-10-06 11:48:06.819 INFO 4184 --- main o.s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 91 ms. Found 1 JPA repository interfaces.
2020-10-06 11:48:06.838 INFO 4184 --- main com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2020-10-06 11:48:07.059 INFO 4184 --- main com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2020-10-06 11:48:07.150 INFO 4184 --- task-1 org.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-10-06 11:48:07.300 INFO 4184 --- task-1 org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.20.Final
2020-10-06 11:48:07.609 INFO 4184 --- task-1 o.hibernate.annotations.common.Version : Triggering deferred initialization of Spring Data repositories...
2020-10-06 11:48:07.627 INFO 4184 --- task-1 org.hibernate.dialect.Dialect : HHH000080:1: Hibernate Commons Annotations {5.1.0.Final}
2020-10-06 11:48:07.897 INFO 4184 --- task-1 org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
2020-10-06 11:48:08.777 INFO 4184 --- task-1 j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-10-06 11:48:08.786 INFO 4184 --- task-1 org.springframework.data.jpa.repository.support.SimpleJpaRepository : Spring Data repositories initialized!
2020-10-06 11:48:09.083 INFO 4184 --- main c.e.u.a.c.ProyectoJpaBasicoApplication : Started ProyectoJpaBasicoApplication in 4.819 seconds (JVM running for 6.089)
2020-10-06 11:48:09.095 INFO 4184 --- main org.springframework.data.jpa.repository.support.SimpleJpaRepository@7f0ab6 : Closing JPA EntityManagerFactory for persistence unit 'default'
2020-10-06 11:49:08.816 INFO 4184 --- [extShutdownHook] o.s.s.concurrent.ThreadPoolTaskExecutor : Shutting down ExecutorService 'applicationTaskExecutor'
2020-10-06 11:49:08.817 INFO 4184 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2020-10-06 11:49:08.858 INFO 4184 --- [extShutdownHook] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

Operaciones CRUD – (Create)

Persistir una entidad

```
package co.edu.unicauca.asae.core;

import org.springframework.beans.factory.annotation.Autowired;

@SpringBootApplication
public class ProyectoJpaBasicoApplication implements CommandLineRunner{

    @Autowired
    private CategoriasRepository servicioAccesoBD;

    public static void main(String[] args) {
        SpringApplication.run(ProyectoJpaBasicoApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        guardar();
    }

    private void guardar()
    {
        Categoria objCategoria= new Categoria();
        objCategoria.setNombre("Desarrollador en java");
        objCategoria.setDescripcion("Se requiere un desarrollador con conocimientos en Spring boot");

        if(this.servicioAccesoBD.save(objCategoria)!=null)
        {
            System.out.println("registro almacenado exitosamente");
        }
        else
        {
            System.out.println("error al realizar el registro");
        }
    }
}
```

← Creamos el objeto a almacenar

← Invocamos el método save de la instancia del repositorio creada automáticamente

El método crea internamente el código SQL para insertar un registro en la base de datos

Operaciones CRUD – (Read)

Recuperar una entidad por id

```
private void buscarPorID()
{
    Optional<Categoria> optional=this.servicioAccesoBD.findById(1);
    if(optional.isPresent()) {
        Categoria objCategoria=optional.get();
        System.out.println("Id categoria: " + objCategoria.getId());
        System.out.println("Nombre categoria: " + objCategoria.getNombre());
        System.out.println("Descripción categoria: " + objCategoria.getDescripcion());

    }else {
        System.out.println("Categoria no encontrada");
    }
}
```

- ❖ Al invocar al método **findById()**, Spring automáticamente crea una sentencia SQL search que buscará un objeto Categoria con Id=1 en la BD.
- ❖ La clase Opcional nos permite verificar si el objeto fue encontrado o no. El método **isPresente()** **retorna true**, si el objeto fue encontrado, y con el método **get()** **retorna el objeto encontrado**

Operaciones CRUD – (Update)

Actualizar una entidad

```
private void modificar()
{
    Optional<Categoria> optional=this.servicioAccesoBD.findById(1);
    if(optional.isPresent()) {
        Categoria objCategoria = optional.get();
        objCategoria.setNombre("Ing. Sistemas");
        objCategoria.setDescripcion("Ingeniero de sistemas con 2 años de experiencia");
        this.servicioAccesoBD.save(objCategoria);
    }else {
        System.out.println("Categoria no encontrada");
    }
}
```

-
- ❖ Al invocar al método **save()**, Spring automáticamente crea una sentencia SQL **update** que actualizará el estado del objeto Categoria a la BD.

Operaciones CRUD – (Delete)

Eliminar una entidad por id

```
private void eliminar()
{
    Optional<Categoria> optional=this.servicioAccesoBD.findById(2);
    if(optional.isPresent()) {
        this.servicioAccesoBD.deleteById(2);
        System.out.println("Categoria eliminada");
    }else {
        System.out.println("Categoria no encontrada");
    }
}
```

- ❖ Al invocar al método **deleteById()**, Spring automáticamente crea una sentencia SQL **delete** que eliminará registro con Id=2 de la BD.

Método findById – Recuperar varias entidades por Id

```
INSERT INTO `categorias` (`id`, `nombre`, `descripcion`) VALUES (NULL, 'ventas', 'Descripción de la categoría ventas');  
INSERT INTO `categorias` (`id`, `nombre`, `descripcion`) VALUES (NULL, 'ingeniería civil', 'Descripción de la categoría ingeniería civil');  
INSERT INTO `categorias` (`id`, `nombre`, `descripcion`) VALUES (NULL, 'desarrollo de software', 'Descripción de la categoría desarrollo de software');  
INSERT INTO `categorias` (`id`, `nombre`, `descripcion`) VALUES (NULL, 'cocina', 'Descripción de la categoría cocina');
```

```
private void consultarCategoriasPorID()  
{  
  
    List<Integer> listaDeIdentificadores= new LinkedList<Integer>();  
    listaDeIdentificadores.add(1);  
    listaDeIdentificadores.add(3);  
    listaDeIdentificadores.add(5);  
    Iterable<Categoria> categorias=this.servicioAccesoBD.findById(listaDeIdentificadores);  
    for(Categoria objCategoria : categorias) {  
        System.out.println("Id categoria: " + objCategoria.getId());  
        System.out.println("Nombre categoria: " + objCategoria.getNombre());  
        System.out.println("Descripción categoria: " + objCategoria.getDescripcion());  
    }  
}
```

- ❖ En este método se crea una colección, en este caso tipo List, en donde se almacenan los id que se desean buscar.
- ❖ El método retorna en un tipo Iterable las entidades encontradas.
- ❖ En tiempo de ejecución cuando se invoque al método **findById()**, Spring ejecuta una instrucción **select from Categorias ...**

Método existsById – Consultar si existe una entidad por ID

```
private void consultarSiExisteCategoria()
{
    boolean bandera = this.servicioAccesoBD.existsById(10);
    if(bandera)
    {
        System.out.println("La categoria existe");
    }
    else
    {
        System.out.println("La categoria no existe");
    }
}
```

- ❖ En tiempo de ejecución cuando se invoque al método **existsById()**, Spring ejecuta una instrucción **select count(*) as ...**

Método saveAll – Almacenar varias entidades

```
private void guardarVariasCategorias()
{
    List<Categoria> categorias = cargarCategorias();
    this.servicioAccesoBD.saveAll(categorias);
    System.out.println("Categorias almacenadas");
}
```

```
private LinkedList<Categoria> cargarCategorias()
{
    Categoria objCategoria1= new Categoria();
    objCategoria1.setNombre("Cocina");
    objCategoria1.setDescripcion("Descripción de la categoria cocina");

    Categoria objCategoria2= new Categoria();
    objCategoria2.setNombre("Ventas");
    objCategoria2.setDescripcion("Descripción de la categoria ventas");

    Categoria objCategoria3= new Categoria();
    objCategoria3.setNombre("Desarrollo de software");
    objCategoria3.setDescripcion("Descripción de la categoria desarrollo de software");

    LinkedList<Categoria> listaCategorias= new LinkedList<Categoria>();

    listaCategorias.add(objCategoria1);
    listaCategorias.add(objCategoria2);
    listaCategorias.add(objCategoria3);

    return listaCategorias;
}
```

- ❖ Al invocar al método **saveAll()**, Spring automáticamente crea varias instrucciones SQL **insert** que guardará las entidades que están en la lista categorías.

Muchas gracias
Preguntas

