



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.

DGTIC

Diplomado Desarrollo de Sistemas con tecnología JAVA

Modulo 8: Ejercicio 1

Sanvicente Enríquez Daniela Alejandra

Edición 14

Ciclo 2024



Objetivo:

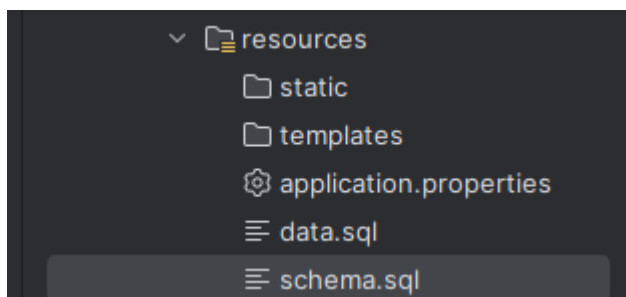
Poner en práctica lo aprendido con respecto a Spring Data para reforzar el conocimiento.

Instrucciones:

1. Se tienen las siguientes tablas:

- i. Producto
- ii. Categoria
- iii. Cliente
- iv. Pais
- v. Venta

En resources añadimos los archivos data.sql y schema.sql



Se creo la clase de la aplicación con el bean del dataSource

```
@SpringBootApplication
public class M8E1Application {

    public static void main(String[] args) { SpringApplication.run(M8E1Application.class, args); }

    @Bean
    public DataSource dataSource() { //Establece la comunicaci3n con la base de datos
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.mariadb.jdbc.Driver");
        dataSource.setUrl("jdbc:mariadb://localhost:3306/modulo8");
        dataSource.setUsername("root");
        dataSource.setPassword("Mar1403");
        return dataSource;
    }
}
```

En cada clase con la anotaci3n @SQL corremos los scripts data y schema antes de las pruebas unitarias

```
@SpringBootTest //@SQL corre los scripts
@Sql({"/schema.sql", "/data.sql"})
//Crea el esquema para nuestro banco de datos modulo8 y a1ade datos a sus tablas
```

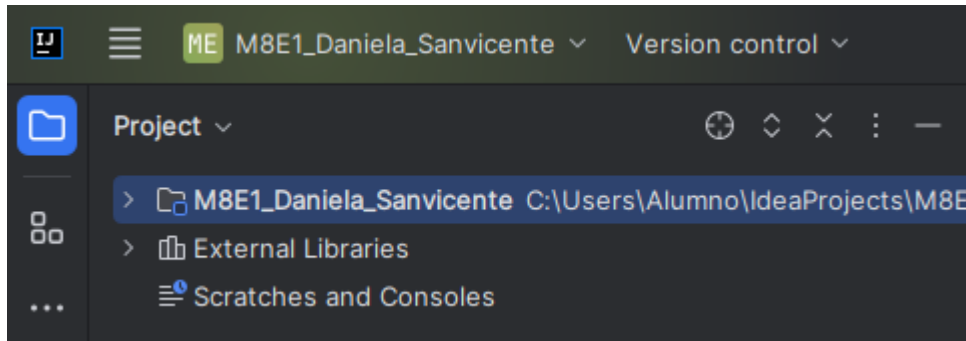
En HeidiSQL se visualizan las tablas creadas

modulo8	144.0 KiB
categoria	16.0 KiB
cliente	16.0 KiB
pais	48.0 KiB
producto	16.0 KiB
venta	16.0 KiB
venta_producto	32.0 KiB

2. Se debe crear un proyecto Spring nuevo con las siguientes características:

- i. Nombre del proyecto:
- ii. M8E1<PrimerNombre>_<PrimerApellido>
- iii. Ejemplo: M8E1_Omar_Mendoza

Creamos nuestro proyecto con el nombre M8E1_Daniela_Sanvicente.



En el pom.xml

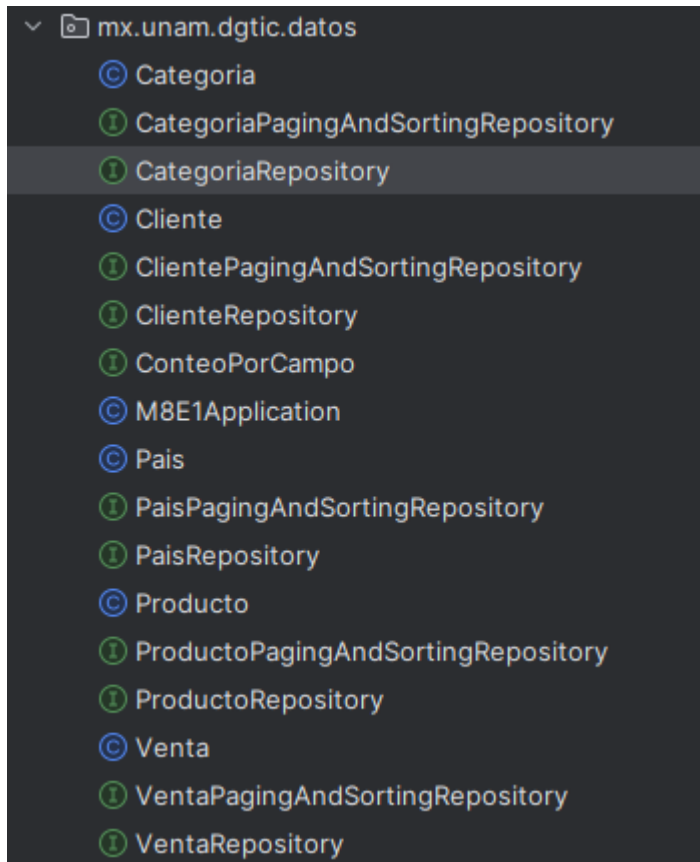
```
<groupId>mx.unam.dgtic.datos</groupId>
<artifactId>M8E1_Daniela_Sanvicente</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>M8E1</name>
<description>Ejercicio 1 para el Modulo 8 Spring Datos</description>
```

3. Con soporte JPA y MariaDb

En el pom.xml se ven las dependencias necesarias

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>
  <dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

4. Debe implementar las clases e interfaces relacionadas con
- i. CrudRepository
 - ii. PagingAndSortingRepository



5. Implementar relaciones entre las tablas

Se crean las clases tomando en cuenta las relaciones que tienen entre sí, colocando anotaciones para las llaves primarias (@Id) e incrementales (@GeneratedValue), en caso de que el nombre sea diferente en SQL (@Column), si son únicas (@Column(unique = true)) y si el nombre es diferente con llaves extranjeras (@JoinColumn).

i. ManyToOne

Para las relaciones ManyToOne como colocamos la anotación @ManyToOne

```
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "clave_cliente")
    private Integer claveCliente;

    private String nombre;
    private String paterno;
    private Date fnac;
    private String curp;

    @ManyToOne
    @JoinColumn(name = "id_pais")
    private Pais pais;
}
```

ii. OneToMany

Para las relaciones OneToMany como colocamos la anotación @OneToMany (mappedBy), donde mappedBy es para que Spring no cree una tabla para relacionar las dos entidades, diciendo que ya está relacionado por el lado dueño que es el lado con la llave foránea (que en el caso de la captura es la entidad/clase de Java que sirven como mapeo para interactuar con la base de datos, desde un programa Java producto)

```
public class Categoria {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_categoria")
    private Integer id;
    private String descripcion;

    @OneToMany(mappedBy = "categoria")
    private List<Producto> productos;
}
```

iii. ManyToMany

Para las relaciones ManyToMany como colocamos la anotación @ManyToMany. Las relaciones ManyToMany necesitan una tabla asociativa (en el caso de la captura es definida por la anotación @JoinTable con el valor venta_producto), con ambas las direcciones de la asociación de esta nueva tabla (definidas por joinColumns y inverseJoinColumns).

```
public class Venta {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id_venta")  
    private Integer id;  
  
    @ManyToOne // unidireccional  
    @JoinColumn(name = "clave_cliente", referencedColumnName = "clave_cliente")//@JoinColumn es para l  
    private Cliente cliente;  
  
    @Column(name = "fecha_venta")  
    private Date fechaVenta;  
  
    @Column(name = "total_venta")  
    private Long totalVenta;  
  
    @ManyToMany  
    @JoinTable(  
        name = "venta_producto",  
        joinColumns = @JoinColumn(name = "id_venta"),  
        inverseJoinColumns = @JoinColumn(name = "id_producto")  
    )  
    private List<Producto> productos;
```

6. Implementar operaciones CRUD sobre cliente y producto

En CrudTests vemos los métodos para las operaciones CRUD de cliente y producto siguientes:

Create (insertar)

Read (buscar)

Update (editar)

Delete (eliminar)

✓ CrudTests	1 sec 951 ms
✓ eliminarProducto()	247 ms
✓ insertarVentaProductoCliente()	205 ms
✓ buscarProducto()	198 ms
✓ editarCliente()	195 ms
✓ eliminarCliente()	255 ms
✓ insertarProducto()	250 ms
✓ insertarCliente()	195 ms
✓ buscarCliente()	211 ms
✓ editarProducto()	195 ms

i. El esquema y el estado de la tablas se deben crear desde una prueba unitaria

Para el para el esquema y estado de las tablas se utilizan las anotaciones @SQL y @Test

```
package mx.unam.dgtic.datos;

> import ...

@SpringBootTest
// prepara esquema de las tablas
@Sql({"schema.sql", "data.sql"})
public class CrudTests {
```


7. Implementar inserción de ventas de productos-cliente

Utilizamos `@Transactional` para asegurar que sea eager y para que sea la operación totalmente completada o no sea realizada. Se utiliza try-catch para que sean capturados los errores de lectura de la fecha enviada. Para la inserción de la venta se puede comprobar la asociación con un cliente y sus productos al ver que los buscamos y los enviamos en el constructor de venta.

```
// prepara estado de la tabla
// es inserción de venta con producto y cliente
@Test
@Transactional
void insertarVentaProductoCliente(){
    try {
        System.out.println("Insertar una Venta");

        Cliente cliente = repositorioCliente.findById(7).get();
        List<Producto> productos = (List<Producto>) repositorioProducto.findAll();

        Venta venta = new Venta(id: 20, cliente, new SimpleDateFormat(pattern: "yyyy-MM-dd").parse(source: "20
        repositorioVenta.save(venta);

        Optional<Venta> optional = repositorioVenta.findById(20);

        if (optional.isPresent()) {
            System.out.println(optional.get().toString());
        } else {
            System.out.println("Venta NO localizada");
        }
    } catch (ParseException e){
        throw new RuntimeException(e);
    }
}
```

8. Implementar consultas sobre cliente, producto, categoría, país y ventas

i. Consultas derivadas (al menos 5)

✓	✓	CategoriaConsultasDerivadasTests	2 sec 120 ms
✓		cuentaPorDescripcionNulo()	738 ms
✓		cuentaPorDescripcionNot()	199 ms
✓		buscarPorDescripcionNot()	307 ms
✓		buscarPorDescripcionNulo()	235 ms
✓		buscarPorDescripcion()	226 ms
✓		buscarExistenciaPorDescripcion()	209 ms
✓		cuentaPorDescripcion()	206 ms

ii. Consultas nombradas (al menos 5)

✓	✓	CategoriaConsultasNombradasTests	1 sec 166 ms
✓		buscarPorDescripcionUnica()	250 ms
✓		buscarDescripcionGrande()	199 ms
✓		buscarPorDescripcion()	342 ms
✓		buscarDescripcionCorta()	186 ms
✓		cuentaPorDescripcionUnica()	189 ms

iii. Consultas con relaciones (al menos 5)

✓	✓	ConsultasConRelacionesTests	1 sec 577 ms
✓		buscarProductosPorVenta()	249 ms
✓		buscarMontoProductosPorVenta()	251 ms
✓		buscarClientesPorVenta()	234 ms
✓		buscarClientesPorPais()	242 ms
✓		buscarPaisPorCliente()	194 ms
✓		buscarProductosPorCategoria()	192 ms
✓		buscarCategoriaPorProducto()	215 ms

9. Probar las implementaciones en pruebas unitarias organizadas por tema (diferentes clases de prueba para cada tema que incluyen varias pruebas unitarias).

Las pruebas se encuentran divididas en temas como se muestra a continuación en la captura.

✓ datos (mx.unam.dgtic)	6 sec 590 ms
> ✓ CategoriaConsultasDerivadasTests	2 sec 126 ms
> ✓ CrudTests	1 sec 964 ms
> ✓ CategoriaConsultasNombradasTests	1 sec 53 ms
> ✓ ConsultasConRelacionesTests	1 sec 447 ms