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

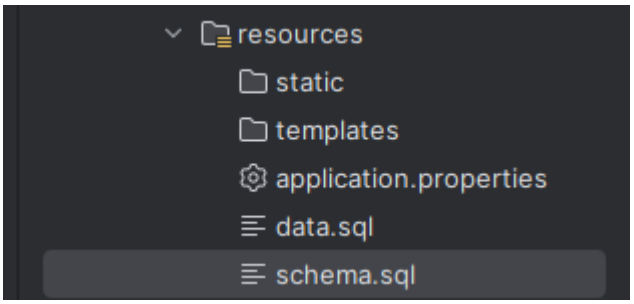
Objetivo:

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



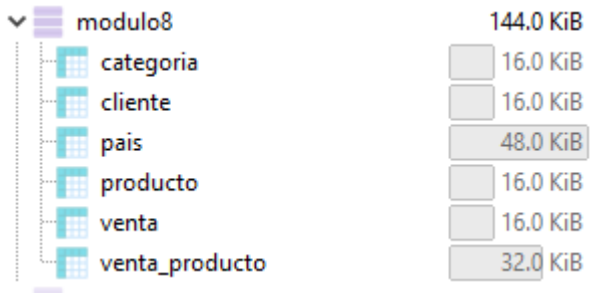
Se creo la clase de la aplicación con el bean del dataSource

```java
@SpringBootApplication
public class M8E1Application {

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


    @Bean
    public DataSource dataSource(){ //Establece la comunicación 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("Mar14D3");
        return dataSource;
    }
```

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

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

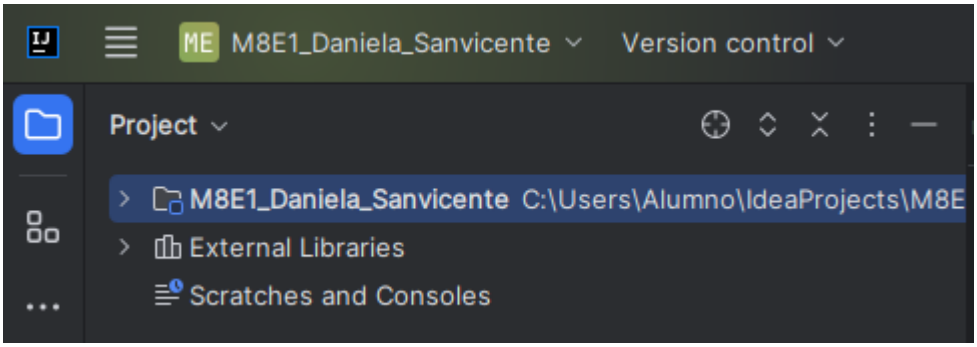
En HeidiSQL se visualizan las tablas creadas



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

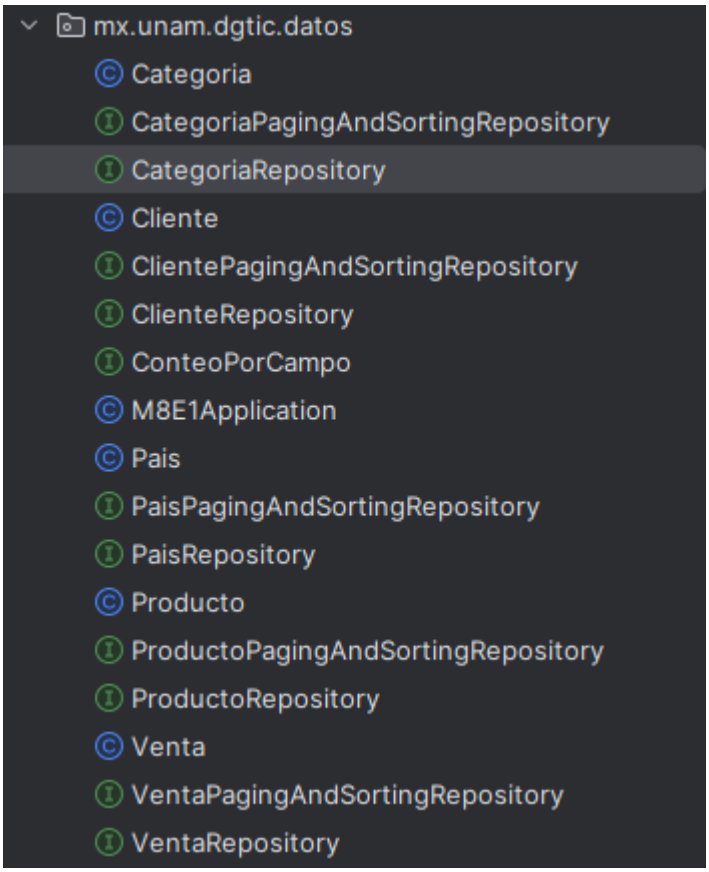
```
<groupId>mx.unam.dgtic.datos</groupId>
<artifactId>M8E1_Daniela_Sanvicente</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>M8E1</name>
<description>Ejercicio 1 para el Modulo 8 Spring Datos</description>
```

3. Con soporte JPA y MariaDb

En el pom.xml se ven las dependencias necesarias

```xml
<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 entra las tablas

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

i. ManyToOne

Para las relaciones ManyToOne como colocamos la anotación @ManyToOne

```java
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), dónde 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)

```java
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).

```java
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 li
    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)

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

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

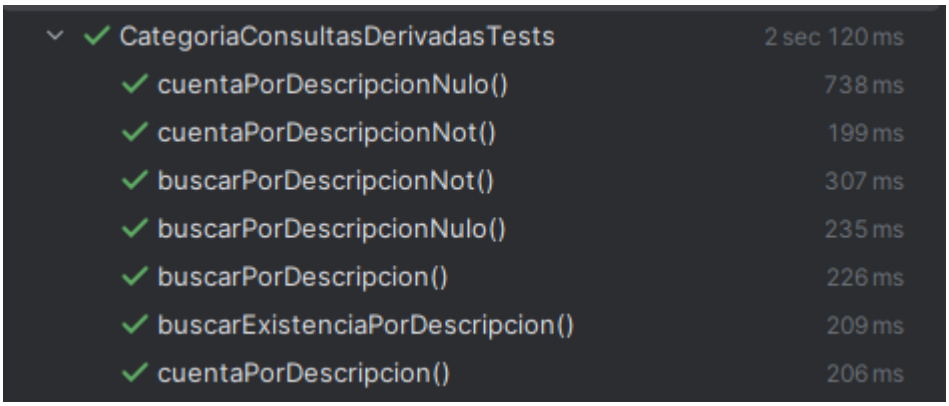
```
package mx.unam.dgtic.datos;

import ...

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

7. Implementar inserción de ventas de productos-cliente

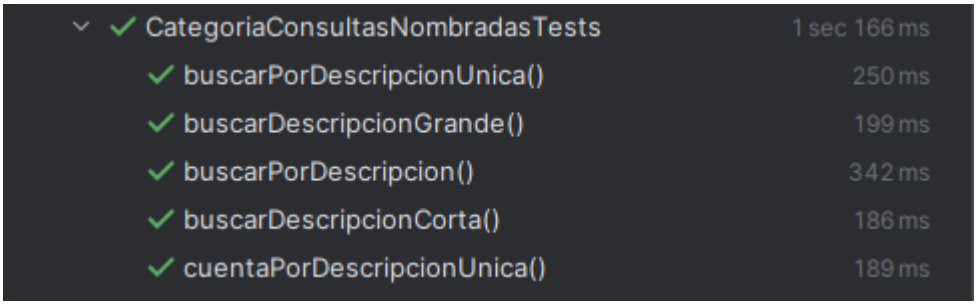
Utilizamos @Transactional para asegurar que sea eager y para que sea la operación totalmente completada o no sea realizada. Se utiliza try-catch para que sean capturados los errores de lectura de la fecha enviada. Para la inserción de la venta se puede comprobar la asociación con un cliente y sus productos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

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

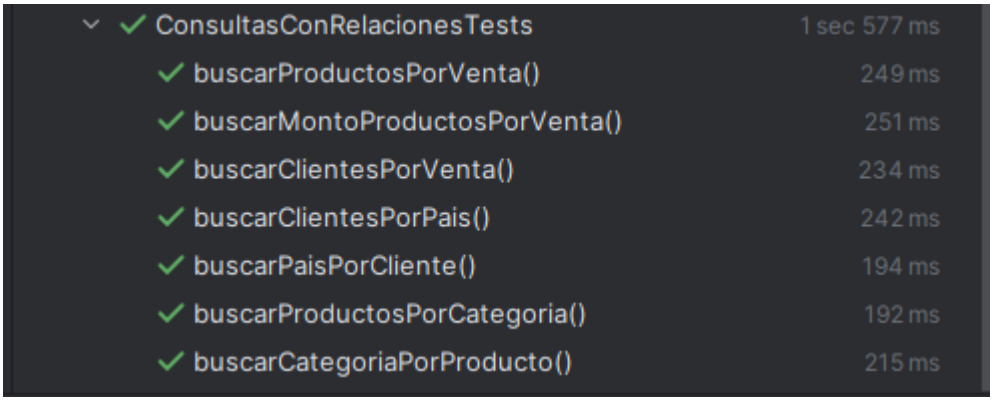
i. Consultas derivadas (al menos 5)



ii. Consultas nombradas (al menos 5)



iii. Consultas con relaciones (al menos 5)



9. Probar las implentaciones 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.