UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

CONTENIDO

REALIZAR UNA APLICACIÓN CON PERSISTENCIA A BASES DE DATOS

INTRODUCCION

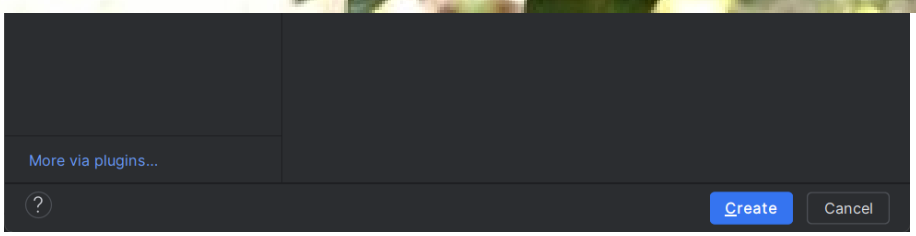
En el presente trabajo se desarrolla una aplicación Java con persistencia de datos utilizando Hibernate como framework ORM (Object Relational Mapping) y MySQL como sistema de gestión de base de datos relacional. El proyecto fue implementado en IntelliJ IDEA y tiene como propósito

- **Documentar** el desarrollo del proyecto con capturas de pantalla y código fuente, generando un informe final en PDF que evidencie la funcionalidad implementada.

DESARROLLO

Creacion del proyecto

Para dar inicio al desarrollo, se procedió a crear un nuevo proyecto en IntelliJ IDEA utilizando el lenguaje Java. Durante la configuración inicial, se optó por Maven como gestor de dependencias, lo que permitió simplificar la incorporación de Hibernate y el conector de MySQL a través del archivo *pom.xml*.



Crear la tabla productos (id, nombre, fecha, precio)

En **mysql** se generó la tabla **productos** con los siguientes campos:

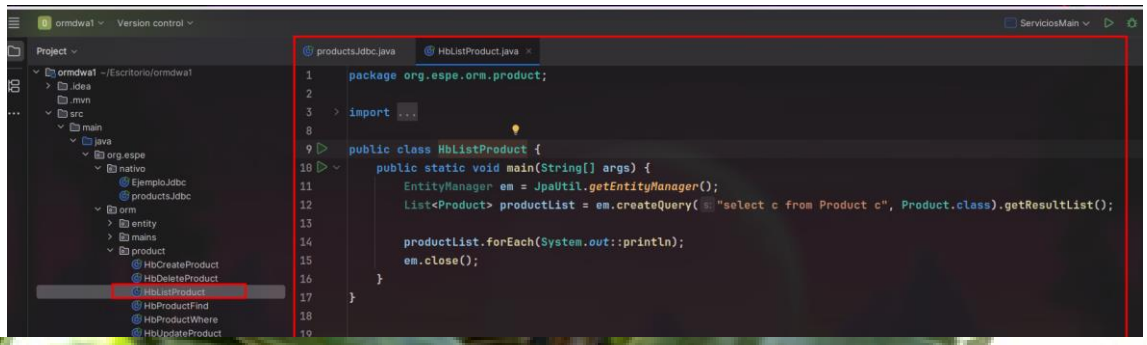
- id → entero, clave primaria, autoincremental.
- nombre → texto (VARCHAR 200).
- fecha → tipo DATE.
- precio → tipo DECIMAL(10,0)

```
mysql> desc productos;
+--------+---------------+------+-----+---------+----------------+
| Field  | Type          | Null | Key | Default | Extra          |
+--------+---------------+------+-----+---------+----------------+
```

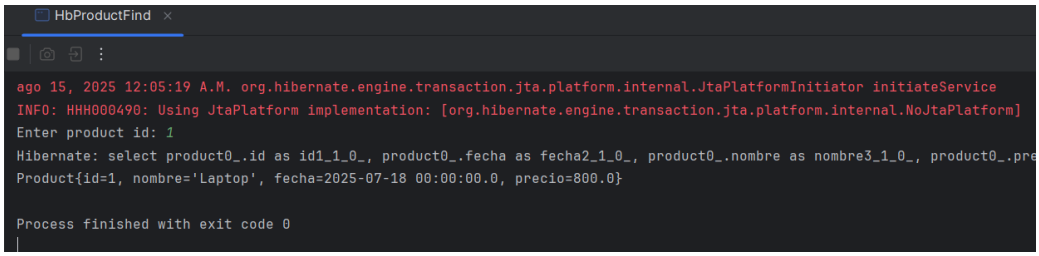
```
        }
    }
}
```

En esta fase se empleó Hibernate para recuperar la totalidad de los registros presentes en la tabla productos utilizando una consulta JPQL.

La aplicación estableció la conexión con la base de datos a través de un EntityManager, ejecutó la sentencia SELECT c FROM Product c y posteriormente iteró sobre la colección de objetos Product para mostrar su información en la consola.



```java
public class HbProductFind {
    public static void main(String[] args) {
        EntityManager em = JpaUtil.getEntityManager();
        System.out.print("Enter product id: ");
        Scanner sn = new Scanner(System.in);
        Long idKeyboard = sn.nextLong();
        Product product = em.find(Product.class,idKeyboard);
        System.out.println(product);
        em.close();
    }
}
```



En esta etapa se utilizó Hibernate para ejecutar una consulta JPQL con cláusula WHERE,

En esta fase se implementó la funcionalidad de creación de registros en la tabla productos utilizando Hibernate.

La aplicación solicita al usuario la información del nuevo producto nombre, fecha y precio, realiza la conversión de la fecha y el precio a los tipos de datos adecuados y, posteriormente, guarda el objeto **Product** en la base de datos dentro de una transacción controlada.

productsJdbc.java | HbListProduct.java | HbProductWhere.java | HbCreateProduct.java

```java
package org.espe.orm.product;

import ...

public class HbCreateProduct {
    public static void main(String[] args) {
        EntityManager em = JpaUtil.getEntityManager();
        try {
            String name = JOptionPane.showInputDialog("name: ");
            String dateStr = JOptionPane.showInputDialog("Date (dd/MM/yyyy):");
            // Convertir String a Date
            SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd/MM/yyyy");
            Date date = null;
            try {
                date = sdf.parse(dateStr);
```

utilizando Hibernate.

La aplicación solicita al usuario el ID del producto que desea modificar, obtiene la información actual del registro y ofrece la posibilidad de cambiar sus valores a través de cuadros de diálogo. En caso de que algún campo no sea modificado, se mantiene el dato original. Finalmente, los cambios se aplican en la base de datos empleando el método merge() dentro de una transacción.

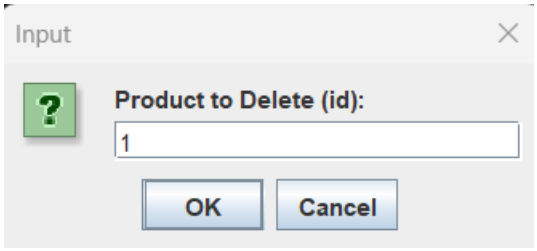
```java
import ...

public class HbUpdateProduct {
    public static void main(String[] args) {
        EntityManager em = JpaUtil.getEntityManager();
        try {
            Long id = Long.valueOf(JOptionPane.showInputDialog("Product to find: "));
            Product productNew = em.find(Product.class, id);

            String name = JOptionPane.showInputDialog( message: "name: ", productNew.getNombre());
            SimpleDateFormat sdf = new SimpleDateFormat( pattern: "dd/MM/yyyy");
```

```
java.text
public class SimpleDateFormat
extends java.text.DateFormat
```

```
wInputDialog(
y):",
tFecha())
```

En esta fase se desarrolló la operación de eliminación de registros en la tabla productos utilizando Hibernate.

La aplicación solicita al usuario el ID del producto que desea suprimir, localiza el registro correspondiente en la base de datos y, en caso de existir, procede a eliminarlo empleando el método remove() dentro de una transacción.



Verificamos en la base de datos, y se elimino el producto con ID 1

CONCLUSIONES

1. Se comprobó que Hibernate facilita la gestión de la persistencia de datos en Java, permitiendo realizar operaciones CRUD de manera eficiente y reduciendo la complejidad de las consultas SQL nativas.

2. El uso de JPQL y consultas parametrizadas demostró la capacidad de Hibernate para realizar búsquedas dinámicas y filtradas, mejorando la interacción con la base de datos frente a consultas nativas.
3. La implementación de la clase ServiciosMain permitió centralizar las operaciones sobre la base de datos, promoviendo un diseño más limpio y facilitando futuras ampliaciones del proyecto.
4. La práctica permitió entender de forma práctica cómo crear, leer, actualizar y eliminar registros en la base de datos usando Hibernate, así como manejar transacciones y parámetros de manera segura.

Elaborado por:	Revisador por:
---------- Nombre: Jairo Smith Bonilla Hidalgo Fecha:15/08/2025	---------- Nombre: Fecha: