UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INGENIERIA DE SOFTWARE

PROGRAMACIÓN WEB AVANZADA

TAREA 1: REALIZAR UNA APLICACIÓN CON PERSISTENCIA A BASES DE DATOS

ESTUDIANTE: Jairo Smith Bonilla Hidalgo

UNIDAD: 3

NRC: 23275

SANGOLQUÍ, AGOSTO DEL 2025

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 demostrar el ciclo completo de operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre una tabla denominada productos, que contiene los campos id, nombre, fecha y precio. Se parte de la creación del proyecto y la conexión nativa a MySQL para la consulta inicial de datos, avanzando posteriormente hacia la integración con Hibernate para realizar consultas más avanzadas, filtradas y parametrizadas. Además, se incluyen servicios para centralizar la lógica de negocio y se presentan capturas de la funcionalidad y el código fuente para su documentación.

OBJETIVOS

Objetivo general:

Desarrollar una aplicación Java con persistencia en MySQL utilizando Hibernate, que permita gestionar la tabla **productos** mediante operaciones CRUD y consultas personalizadas, aplicando buenas prácticas de programación y organización de código.

Objetivos específicos:

- **Configurar** un proyecto en IntelliJ IDEA que integre Hibernate con MySQL, asegurando una conexión correcta a la base de datos.
- **Implementar** operaciones CRUD y consultas personalizadas sobre la tabla **productos** utilizando Hibernate, incluyendo filtrado por ID y condiciones específicas.
- **Documentar** el desarrollo del proyecto con capturas de pantalla y código fuente, generando un informe final en PDF que evidencie la funcionalidad implementada.

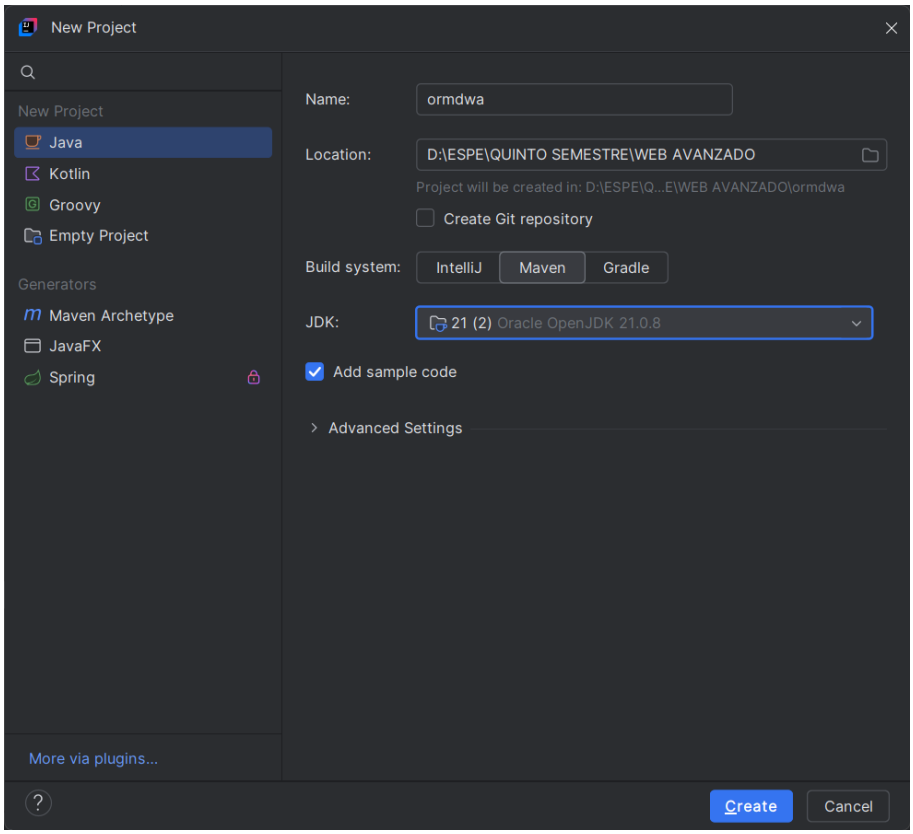
DESARROLLO

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*.

Los pasos esenciales fueron los siguientes:

- Abrir **In**telliJ IDEA y acceder a la opción *File* > *New* > *Project*.
- Seleccionar Maven como tipo de proyecto e indicar la versión de Java a utilizar.
- Asignar un nombre al proyecto y definir la ubicación de la carpeta de trabajo.



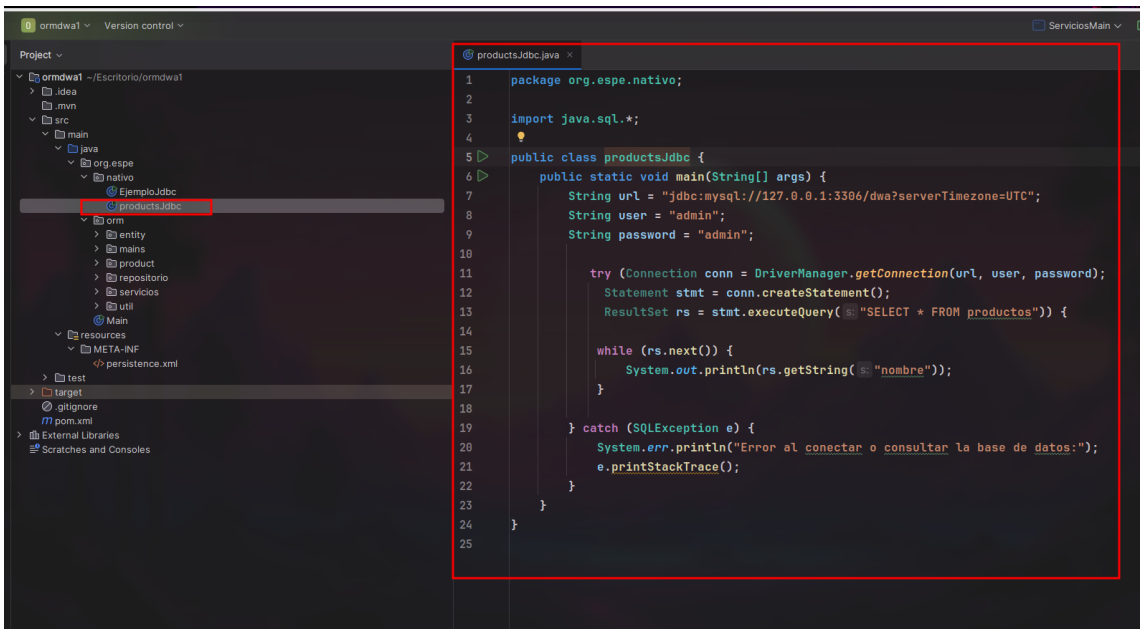
Crear la tabla productos (id, nombre, fecha, precio)

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)

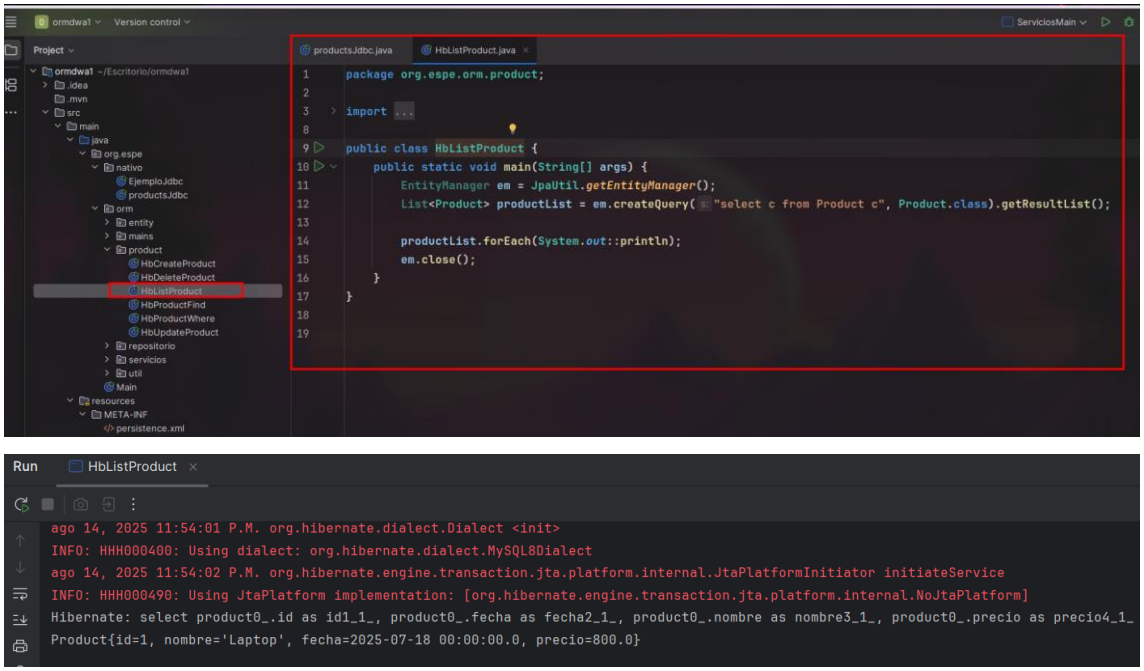
```
mysql> desc productos;
+--------+---------------+------+-----+---------+----------------+
| Field  | Type          | Null | Key | Default | Extra          |
+--------+---------------+------+-----+---------+----------------+
| id     | int           | NO   | PRI | NULL    | auto_increment |
| nombre | varchar(255)  | NO   |     | NULL    |                |
| precio | decimal(10,2) | NO   |     | NULL    |                |
| fecha  | date          | NO   |     | NULL    |                |
+--------+---------------+------+-----+---------+----------------+
4 rows in set (0,00 sec)
```

En esta etapa se realizó el acceso directo a la base de datos empleando JDBC. La aplicación abrió una conexión con MySQL, ejecutó una sentencia SELECT * FROM productos y recuperó la información almacenada. Posteriormente, se imprimió en la consola el valor correspondiente a la columna nombre para cada uno de los registros obtenidos.



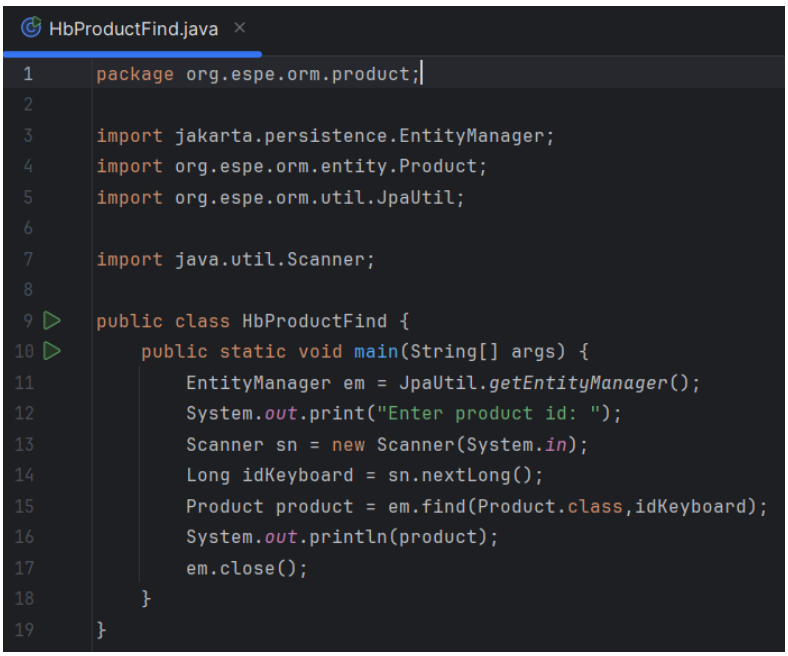
En esta fase se empleó Hibernate para recuperar la totalidad de los registros presentes en la tabla productos utilizando una consulta JPQL.

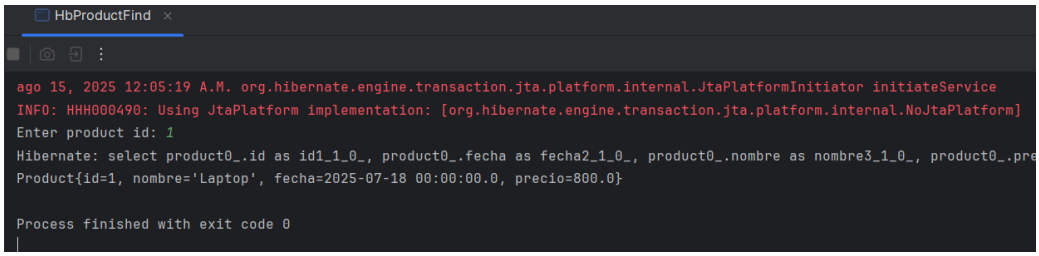
La aplicación estableció la conexión con la base de datos a través de un EntityManager, ejecutó la sentencia SELECT c FROM Product c y posteriormente iteró sobre la colección de objetos Product para mostrar su información en la consola.



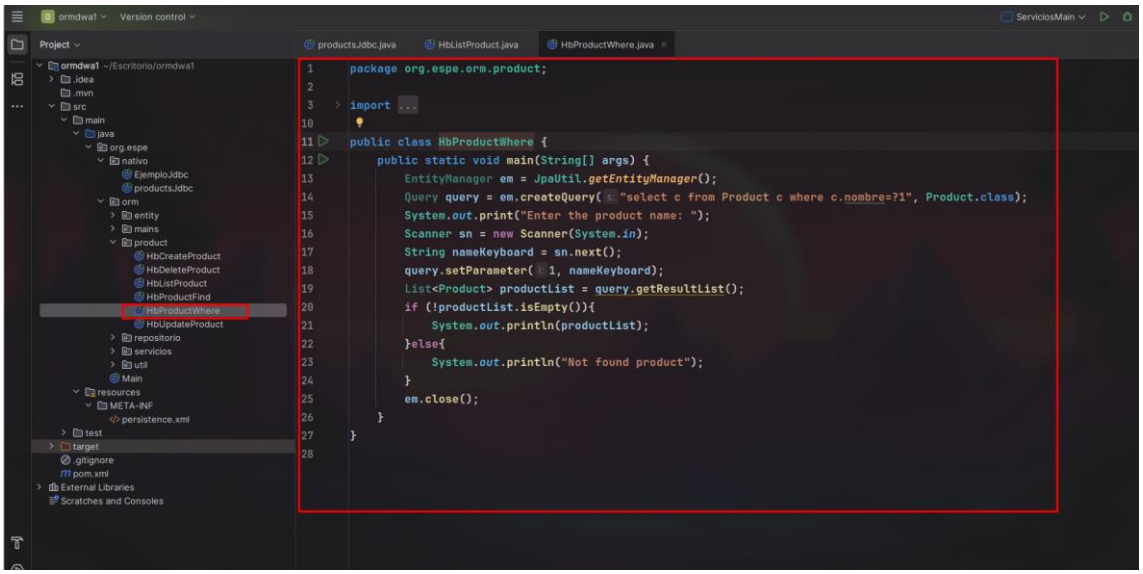
Listar con hibernate pe: hibernatelistarid

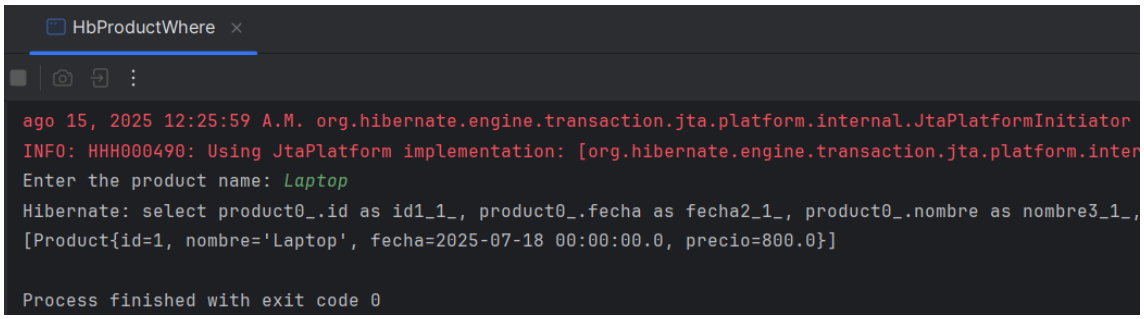
En este paso se utilizó Hibernate para buscar un producto específico según su ID. El programa solicita al usuario que ingrese un identificador, utiliza el método em.find() para obtener el registro correspondiente y lo muestra por consola.





En esta etapa se utilizó Hibernate para ejecutar una consulta JPQL con cláusula WHERE, destinada a obtener únicamente los productos cuyo nombre coincida con un criterio específico. La aplicación solicita al usuario el nombre a buscar, incorpora dicho valor como parámetro en la consulta y, finalmente, presenta en la consola los registros encontrados.



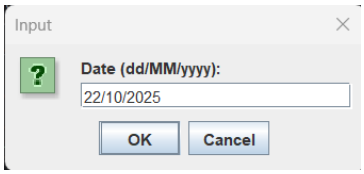


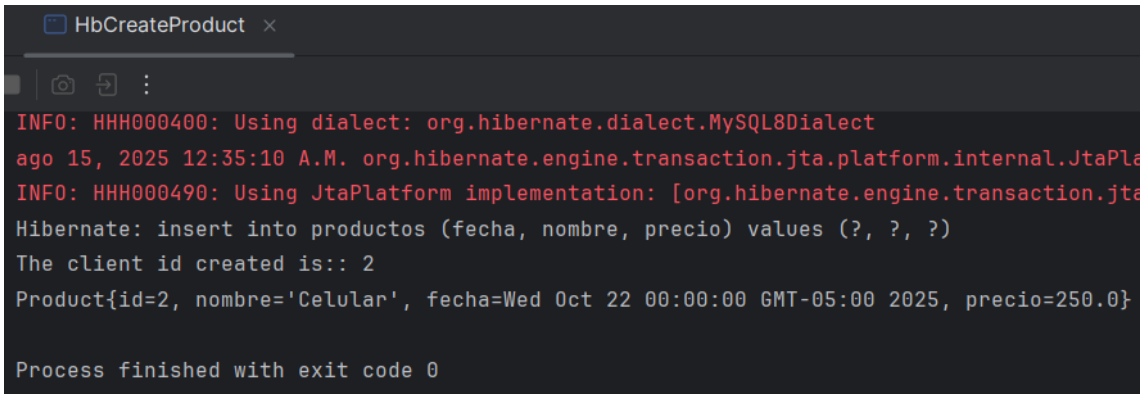
En esta fase se implementó la funcionalidad de creación de registros en la tabla productos utilizando Hibernate.

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);
            } catch (ParseException e) {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Incorrect date format. Use dd/MM/yyyy");
                return;
            }
            String priceStr = JOptionPane.showInputDialog("price: ");
            // Convertir String a float
            float price = 0f;
            try {
                price = Float.parseFloat(priceStr);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid price. Enter a valid number.");
                return;
            }
```





En esta etapa se implementó la actualización de un registro existente en la tabla productos 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.

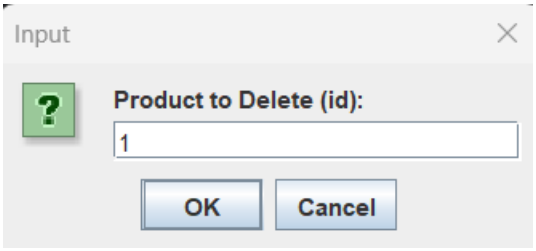
```java
import ...

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");
                                                          wInputDialog(
                                                          y):",
                                                          tFecha())
            try {
                date = sdf.parse(dateStr);
            } catch (ParseException e) {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Incorrect date format. Use dd/MM/yyyy");
                return;
            }
            String priceStr = JOptionPane.showInputDialog( message: "price: ", productNew.getPrecio());
            // Convertir String a float
            float price = 0f;
            try {
                price = Float.parseFloat(priceStr);
            } catch (NumberFormatException e) {
                JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid price. Enter a valid number.");
```

```
java.text
public class SimpleDateFormat
extends java.text.DateFormat
< openjdk-24 >
```

```
mysql> select * from productos;
+----+----------+--------+------------+
| id | nombre   | precio | fecha      |
+----+----------+--------+------------+
|  2 | Leche    |   1.50 | 2025-07-25 |
|  3 | Pan      |   0.75 | 2025-07-20 |
|  4 | Queso    |   2.30 | 2025-08-01 |
|  5 | Producto |  12.00 | 2025-12-12 |
+----+----------+--------+------------+
4 rows in set (0,00 sec)
```

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

```
mysql> select * from productos;
+----+----------+--------+------------+
| id | nombre   | precio | fecha      |
+----+----------+--------+------------+
|  2 | Leche    |   1.50 | 2025-07-25 |
|  3 | Pan      |   0.75 | 2025-07-20 |
|  4 | Queso    |   2.30 | 2025-08-01 |
|  5 | Producto |  12.00 | 2025-12-12 |
+----+----------+--------+------------+
4 rows in set (0,00 sec)
```

En esta etapa se implementó la clase **ServiciosMain**, concebida como el punto de entrada principal para gestionar la comunicación con la base de datos mediante **Hibernate**.

```java
package org.espe;

public class Main {
    public static void main(String[] args) {

        System.out.println("Hello and welcome!");
    }
}
```

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.

PARA MI

Con los conocimientos adquiridos en esta práctica, puedo aplicar en el futuro la creación de aplicaciones Java que requieran persistencia de datos de manera eficiente y segura. Seré capaz de diseñar sistemas que gestionen bases de datos relacionales mediante Hibernate, implementando operaciones CRUD, consultas parametrizadas y transacciones controladas.

BIBLIOGRAFIA

- Lesson: JDBC Basics (The JavaTM Tutorials > JDBC Database Access). (n.d.). https://docs.oracle.com/javase/tutorial/jdbc/basics/
- The leading IDE for professional Java and Kotlin development. (2021, June 1). JetBrains. https://www.jetbrains.com/idea/#
- Documentation - 7.1 - Hibernate ORM. (n.d.). Hibernate. https://hibernate.org/orm/documentation/7.1/

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

Name: Jairo Smith Bonilla Hidalgo

10/10

MODAL VERBS WITH CAN AND SHOULD

- Use *can* to talk about things that are possible:
  - Where **can** I get some nice souvenirs?

Use *should* to suggest things that are good to do:
- You **should** try the local restaurants.

- Use the base form with *can* and *should* – not the infinitive:
  - Where **can I get** some nice souvenirs? (NOT: Where can I to get . . ?.)
  - **You should try** the local restaurants. (NOT: You should to try . . .)

Complete these conversations with *can, can't, should*, or *shouldn't.*

First Conversation

A: I [can't] decide where to go on vacation. [should] I go to Costa Rica or Hawaii?
B: You [should] definitely visit Costa Rica.
A: Really? What [can] I see there?
B: Well, San Jose is an exciting city. You [shouldn't] miss the Museo del Oro. That's the gold museum, and you [can] see beautiful animals made of gold.
A: OK. What else [can] I do there?
B: Well, you [should] visit the museum on Mondays. It's closed then. But you [should] definitely visit the rain forest. It's amazing!

Second Conversation

Ana: Have you ever been to Puerto Plata?
Lucas: Well, Actually, I haven't, but I don't know where to go for my vacation either, where [should] I go?
Ana: Hello, I just told you, you [should] definitely visit Puerto Plata.
Lucas: Ok, what [can] I do there?
Ana: Well there are many places that you [can] visit and there are many activities that you [can] do there too.

Ana: You [should] definitely go to Ocean World, you [shouldn't] miss the act of the dolphins
Lucas: What time of the year [can] I go there?
Ana: It's DR, you [can] go there anytime of the year. That's the best part.
Lucas: Thanks for the Information, I [can't] wait to be there.