

**PROGRAMACION ORIENTADA A OBJETOS II**

**SEMANA 5**

Jorge Peñaloza Guerra

Lunes 10 de julio de 2023

ingeniería en informática

# **DESARROLLO**

Contexto

Alberto Pueblas recientemente heredó un negocio familiar. Su padre y abuelo, reconocidos agricultores de la zona, llevan años con el negocio de la fruta, específicamente en los cerezos. Ellos tienen excelentes registros de datos históricos que podrían constituir la base para un sistema de información. Alberto, al estar iniciando en el conocimiento del negocio, le interesa informarse sobre:

* Las variedades de cerezos que existen en cada huerto.
* Cantidad de árboles que existen en cada huerto.
* Variedad de árboles que más hay.

Para lograr este objetivo, te piden crear la base de datos en MySql o SqlServer a partir de este sencillo diagrama Entidad/Relación puedes construir lo que te solicitan con las respectivas tablas, agregando sus atributos y relaciones.

Diagrama

Descripción generada automáticamente

En Eclipse, tienes que crear un programa y la conexión con la base de datos. Debes poblar las tablas con al menos 3 huertos, 10 árboles, 1 especie (cerezo) y 5 variedades.

El sistema debe permitir:

* Listar las variedades de cerezos por huerto.
* Listar el total de árboles por variedad y huerto.
* Ingreso de nuevas variedades.
* Ingreso de nuevas especies.
* Ingreso de nuevos árboles.

Cabe mencionar que la codificación debe estar correctamente documentada, dando énfasis en lo que se refiere a la conexión a la base de datos, y desarrollo de consultas SQL con sus respectivas capturas de pantallas.

A continuación, responde las preguntas y realiza la siguiente actividad:

1. Explica la relación que existe entre la base de datos del caso con la programación orientada al objeto del lenguaje Java.
   1. En el presente caso, la base de datos representa una estructura de almacenamiento de datos persistente que utiliza el sistema de gestión de bases de datos MySQL. La base de datos contiene tablas que representan entidades como huertos, especies, variedades y árboles, y estas entidades tienen atributos que definen sus características.
   2. La programación orientada a objetos en Java se utiliza para modelar y representar estas entidades y sus relaciones en el sistema. Cada tabla de la base de datos se puede mapear a una clase en Java, donde los atributos de la clase representan los campos de la tabla y los métodos de la clase fundamentan la funcionalidad asociada a esas entidades.
   3. En el caso de la clase AdministradorDeHuertos, se utiliza la programación orientada a objetos para encapsular la lógica de administración de huertos, especies, variedades y árboles en métodos específicos. Estos métodos interactúan con la base de datos mediante consultas SQL para realizar como listar variedades de cerezos por huerto, ingresar nuevas variedades o consultar el total de árboles por variedad y huerto.
   4. La relación entre la base de datos y la programación orientada a objetos en Java radica en que la base de datos proporciona la persistencia de los datos, mientras que la programación orientada a objetos permite modelar y manipular esos datos de manera estructurada y orientada a objetos, utilizando clases, objetos y métodos.
2. Señala al menos 3 beneficios que proporciona la conexión Java, apropiada al contexto del caso planteado. Toma en consideración las formas de conexión de Java que existen.
   1. Facilidad de integración: La conexión Java permite establecer una comunicación fluida y directa entre la aplicación desarrollada en Java y la base de datos MySQL. Esto facilita la integración de la lógica de negocio de la aplicación con los datos almacenados en la base de datos, lo que permite acceder, manipular y actualizar los datos de manera eficiente.
   2. Mejor rendimiento y escalabilidad: La conexión Java optimiza el rendimiento de la aplicación al utilizar técnicas como el uso de Prepared Statements y Batch Updates. Estas técnicas permiten minimizar la transferencia de datos necesarios entre la base de datos y la aplicación, mejorando así el rendimiento y la eficiencia. Además, la conexión Java permite gestionar eficientemente transacciones y consultas complejas, lo que facilita la escalabilidad de la aplicación a medida que aumentan los datos y las requeridas.
   3. Seguridad y control: La conexión Java proporciona mecanismos de seguridad para proteger los datos almacenados en la base de datos. Utilizando técnicas como Prepared Statements y consultas parametrizadas, se evita la posibilidad de ataques de inyección de SQL. Además, Java ofrece opciones para establecer conexiones seguras mediante el uso de SSL o autenticación basada en certificados. Esto garantiza que los datos estén protegidos durante la comunicación entre la aplicación y la base de datos.
3. Con ayuda del lenguaje SQL para aplicaciones en Java, crea las consultas adecuadas a las solicitudes del caso expuesto.
   1. En consulta SQL 1 se puede apreciar como consultar las variedades de cerezos
   2. La consulta SQL 2 permite obtener las cantidades de arboles por variedad y huerto
   3. La consulta SQL 3 se permite ejecutar el INSERT a una nueva especie
   4. La consulta SQL 4 se permite ejecutar el INSERT a una nueva variedad
   5. La consulta SQL 5 se permite ejecutar el INSERT a un nuevo árbol
4. Aplica la herramienta de mapeo y establece el vínculo al manejo de la relación BBDD y aplicaciones en Java, con el propósito de atender las necesidades del negocio de la fruta (Cerezo)
   1. En la figura 1 se puede ver como se configura un proyecto que integra Hibernate
   2. En la figura 2 se puede observar los getter’s y setter’s de la tabla huerto luego de mapearla
   3. En la figura se puede un diagrama E-R realizado a través de Hibernate

Scripts y Consultas SQL

Consulta SQL 1

Obtiene variedades por huerto

SELECT  
 arbol.idHuerto,  
 huerto.nombre,  
 huerto.ubicacion,  
 huerto.superficie,  
 arbol.idVariedad,  
 variedad.nombre AS "nombreVariedad"  
FROM  
 arbol  
 INNER JOIN variedad ON arbol.idVariedad = variedad.id  
 INNER JOIN huerto ON arbol.idHuerto = huerto.id  
 INNER JOIN especie ON variedad.idEspecie = especie.id

WHERE  
 especie.nombre = 'Cerezo'

GROUP BY  
 arbol.idHuerto,  
 huerto.nombre,  
 huerto.ubicacion,  
 huerto.superficie,  
 arbol.idVariedad,  
 variedad.nombre;

Fuente: Elaboración propia

Consulta SQL 2

Obtiene el total de árboles por variedad y huerto

SELECT  
 h.nombre AS huerto,  
 v.nombre AS variedad,  
 COUNT(\*) AS total\_arboles  
FROM  
 Huerto h  
 INNER JOIN Arbol a ON h.id = a.idHuerto  
 INNER JOIN Variedad v ON a.idVariedad = v.id  
GROUP BY  
 h.nombre,  
 v.nombre;

Fuente: Elaboración propia

Consulta SQL 3

Inserta nueva especie a base de datos

INSERT INTO Especie (id, nombre) VALUES (?, ?)

Fuente: Elaboración propia

Consulta SQL 4

Inserta nueva variedad a base de datos

INSERT INTO Variedad (id, nombre, idEspecie) VALUES (?, ?, ?)

Fuente: Elaboración propia

Consulta SQL 5

Inserta nuevo árbol a base de datos

"INSERT INTO Arbol (id, idHuerto, idVariedad) VALUES (?, ?, ?)

Fuente: Elaboración propia

Figura 1

Inserta nuevo árbol a base de datos

Captura de pantalla de computadora

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 2

Mapeo de la tabla huerto con Hibernate

Captura de pantalla de computadora

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 3

Diagrama E-R con Hibernate

Interfaz de usuario gráfica

Descripción generada automáticamente

Fuente: Elaboración propia

A continuación, se presentan los códigos fuente y capturas de pantalla del programa solicitado en modo consola. Desde la figura 4 hasta la figura 9 se puede apreciar como se probó la aplicación hasta que en la figura se muestra una consulta general para verificar que los datos ingresados fueron correctamente almacenados en la base de datos.

Script principal 1

Clase: AdministradorDeHuertos.java

import java.sql.\*;  
import java.util.Scanner;  
import java.io.File;  
import java.io.FileNotFoundException;  
*/\*\*  
\* Clase principal que permite administrar huertos, especies, variedades y árboles en una base de datos.  
\* Proporciona funcionalidades como listar las variedades de cerezos por huerto,  
\* listar el total de árboles por variedad y huerto, ingresar nuevas variedades,  
\* ingresar nuevas especies e ingresar nuevos árboles.  
\*/*public class AdministradorDeHuertos {  
 private Connection connection;  
 public AdministradorDeHuertos(Connection connection) {  
 this.connection = connection;  
 }  
 */\*\*  
 \* Muestra las variedades de cerezos por huerto en la base de datos.  
 \* Lee una consulta SQL desde un archivo y la ejecuta.  
 \* La consulta devuelve el nombre del huerto y el nombre de la variedad de cerezo.  
 \* Muestra los resultados por consola.  
 \*/* public void listarVariedadesCerezosPorHuerto() {  
 try {  
 // Ruta del archivo de consulta SQL  
 String filePath = "SQL/queryVariedadesxHuerto.sql";  
  
 // Leer la consulta SQL desde el archivo  
 String sqlQuery = *readQueryFromFile*(filePath);  
  
 //System.out.println(sqlQuery);  
  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery(sqlQuery);  
  
 System.*out*.println("Variedades de cerezos por huerto:");  
 while (resultSet.next()) {  
 String huerto = resultSet.getString("nombre");  
 String variedad = resultSet.getString("nombreVariedad");  
 System.*out*.println(huerto + ": " + variedad);  
 }  
 resultSet.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Permite ingresar una nueva variedad en la base de datos.  
 \* Solicita al usuario el ID, nombre y ID de la especie a la que pertenece la variedad.  
 \* Los valores ingresados se utilizan para ejecutar una instrucción INSERT en la tabla Variedad de la base de datos.  
 \* Muestra un mensaje indicando si la nueva variedad fue ingresada correctamente.  
 \*/* public void listarTotalArbolesPorVariedadYHuerto() {  
 try {  
 // Ruta del archivo de consulta SQL  
 String filePath = "SQL/TotalArbolesPorVariedadYHuerto.sql";  
  
 // Leer la consulta SQL desde el archivo  
 String sqlQuery = *readQueryFromFile*(filePath);  
  
 Statement statement = connection.createStatement();  
 ResultSet resultSet = statement.executeQuery(sqlQuery);  
  
 System.*out*.println("Total de árboles por variedad y huerto:");  
 while (resultSet.next()) {  
 String huerto = resultSet.getString("huerto");  
 String variedad = resultSet.getString("variedad");  
 int totalArboles = resultSet.getInt("total\_arboles");  
 System.*out*.println(huerto + " - " + variedad + ": " + totalArboles + " árbol(es)");  
 }  
 resultSet.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Permite ingresar una nueva variedad en la base de datos.  
 \* Solicita al usuario el ID, nombre y ID de la especie a la que pertenece la variedad.  
 \* Los valores ingresados se utilizan para ejecutar una instrucción INSERT en la tabla Variedad de la base de datos.  
 \* Muestra un mensaje indicando si la nueva variedad fue ingresada correctamente.  
 \*/* public void ingresarNuevaVariedad() {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.println("Ingreso de nueva variedad:");  
 System.*out*.print("ID: ");  
 int idVariedad = scanner.nextInt();  
 scanner.nextLine(); // Consumir el salto de línea  
  
 System.*out*.print("Nombre: ");  
 String nombreVariedad = scanner.nextLine();  
  
 System.*out*.print("ID de la especie a la que pertenece: ");  
 int idEspecie = scanner.nextInt();  
  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO Variedad (id, nombre, idEspecie) VALUES (?, ?, ?)");  
 preparedStatement.setInt(1, idVariedad);  
 preparedStatement.setString(2, nombreVariedad);  
 preparedStatement.setInt(3, idEspecie);  
 preparedStatement.executeUpdate();  
 System.*out*.println("Nueva variedad ingresada correctamente.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Realiza una consulta en la base de datos para obtener el total de árboles por variedad y huerto.  
 \* La consulta se lee desde un archivo de consulta SQL y se ejecuta en la base de datos.  
 \* Los resultados se imprimen en la consola, mostrando el nombre del huerto, la variedad y el total de árboles.  
 \*/* public void ingresarNuevaEspecie() {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.println("Ingreso de nueva especie:");  
 System.*out*.print("ID: ");  
 int idEspecie = scanner.nextInt();  
 scanner.nextLine(); // Consumir el salto de línea  
  
 System.*out*.print("Nombre: ");  
 String nombreEspecie = scanner.nextLine();  
  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO Especie (id, nombre) VALUES (?, ?)");  
 preparedStatement.setInt(1, idEspecie);  
 preparedStatement.setString(2, nombreEspecie);  
 preparedStatement.executeUpdate();  
 System.*out*.println("Nueva especie ingresada correctamente.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Permite ingresar un nuevo árbol en la base de datos.  
 \* Se solicita al usuario ingresar el ID, ID del huerto al que pertenece y ID de la variedad a la que pertenece.  
 \* Los datos ingresados se insertan en la tabla "Arbol".  
 \*/* public void ingresarNuevoArbol() {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.println("Ingreso de nuevo árbol:");  
 System.*out*.print("ID: ");  
 int idArbol = scanner.nextInt();  
 scanner.nextLine(); // Consumir el salto de línea  
  
 System.*out*.print("ID del huerto al que pertenece: ");  
 int idHuerto = scanner.nextInt();  
 System.*out*.print("ID de la variedad a la que pertenece: ");  
 int idVariedad = scanner.nextInt();  
  
 try {  
 PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO Arbol (id, idHuerto, idVariedad) VALUES (?, ?, ?)");  
 preparedStatement.setInt(1, idArbol);  
 preparedStatement.setInt(2, idHuerto);  
 preparedStatement.setInt(3, idVariedad);  
 preparedStatement.executeUpdate();  
 System.*out*.println("Nuevo árbol ingresado correctamente.");  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Punto de entrada principal del programa.  
 \*  
 \* @param args los argumentos de la línea de comandos  
 \*/* public static void main(String[] args) {  
 try {  
 // Crear conexión a la base de datos  
 Connection connection = DriverManager.*getConnection*("jdbc:mysql://127.0.0.1:3306/agricultura", "root", "");  
  
 // Crear objeto HuertoManager  
 AdministradorDeHuertos huertoManager = new AdministradorDeHuertos(connection);  
  
 // Menú  
 Scanner scanner = new Scanner(System.*in*);  
 int opcion = 0;  
  
 while (opcion != 6) {  
 System.*out*.println("---------------------");  
 System.*out*.println("MENU");  
 System.*out*.println("---------------------");  
 System.*out*.println("1. Listar las variedades de cerezos por huerto");  
 System.*out*.println("2. Listar el total de árboles por variedad y huerto");  
 System.*out*.println("3. Ingreso de nuevas variedades");  
 System.*out*.println("4. Ingreso de nuevas especies");  
 System.*out*.println("5. Ingreso de nuevos árboles");  
 System.*out*.println("6. Salir");  
 System.*out*.print("Ingrese una opción: ");  
 opcion = scanner.nextInt();  
  
 switch (opcion) {  
 case 1:  
 huertoManager.listarVariedadesCerezosPorHuerto();  
 break;  
 case 2:  
 huertoManager.listarTotalArbolesPorVariedadYHuerto();  
 break;  
 case 3:  
 huertoManager.ingresarNuevaVariedad();  
 break;  
 case 4:  
 huertoManager.ingresarNuevaEspecie();  
 break;  
 case 5:  
 huertoManager.ingresarNuevoArbol();  
 break;  
 case 6:  
 System.*out*.println("Hasta luego!");  
 break;  
 default:  
 System.*out*.println("Opción inválida. Intente nuevamente.");  
 break;  
 }  
 }  
  
 connection.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 */\*\*  
 \* Lee el contenido de un archivo de consulta y lo devuelve como una cadena.  
 \*  
 \* @param filePath la ruta del archivo de consulta  
 \* @return una cadena con el contenido del archivo de consulta  
 \*/* public static String readQueryFromFile(String filePath) {  
 StringBuilder queryBuilder = new StringBuilder();  
  
 try {  
 File file = new File(filePath);  
 Scanner scanner = new Scanner(file);  
  
 while (scanner.hasNextLine()) {  
 String line = scanner.nextLine();  
 queryBuilder.append(line);  
 }  
  
 scanner.close();  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 return queryBuilder.toString();  
 }  
}

Fuente: Elaboración propia

Figura 4

Menú: opción 1

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 5

Manu: opción 2

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 6

Menú: opción 3

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 7

Menú: opción 4

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 8

Menú: opción 5

Texto

Descripción generada automáticamente

Fuente: Elaboración propia

Figura 9

Vista de datos después de usar la aplicación

Tabla

Descripción generada automáticamente

Fuente: Elaboración propia

# **REFERENCIAS BIBLIOGRÁFICAS**

IACC. (2022). Uso de base de datos en JAVA. En *Programación orientada a objetos II. Semana 5.*