```
package bookapp;


import java.util.Scanner;

/*
 *
 *
 *@ Nombre: Jose Armando Villar Almonte
 *@ Matricula: 100413737
 *@ Materia: INF 5150-06
 *
 *
 */

public class BookStoreApp {
    private static final Scanner sc = new Scanner(System.in);

    private static DatabaseManager dbManager;

    public static void main(String[] args) {
        dbManager = new DatabaseManager();
        dbManager.connect();
        new BookStoreManager();
        new BookStoreQueries();

        int option;

        do {
            System.out.println("\n---MENU DE OPCIONES---");
            System.out.print("\n");
            System.out.println("1.Ver tabla");
            System.out.println("2.Añadir Campo nuevo a la tabla");
            System.out.println("3.Modificar un valor a la tabla");
            System.out.println("4.Eliminar un registro");
            System.out.println("5.Consultas Especificas");
            System.out.println("6.Consultar Porcentaje de ventas por titulos");
            System.out.println("7.Consultar Precio de ventas y Total de ventas");
            System.out.println("8.Salir");
            System.out.print("\n");
            System.out.print("Seleccione una opción: ");

            option = sc.nextInt();
            sc.nextLine(); // Consumir la nueva línea

            switch(option) {
            case 1:dbManager.showTablesAndData();break;

            case 2:System.out.print("Ingrese el nombre de la tabla (Por ejemplo,
'authors'): ");
            String tableName = sc.nextLine();
            BookStoreManager.addField(tableName, sc);
```

```
            break;

            case 3: BookStoreManager.modifyField(sc); break;

            case 4:  System.out.println("ELIMINAR FILA DE UNA TABLA.");
            System.out.print("Ingrese el nombre de la tabla (por ejemplo,
'authors'): ");
            String deleteField = sc.nextLine();
            BookStoreManager.deleteRow(deleteField, sc); // Llamamos al método
para eliminar la fila
            break;

            case 5: BookStoreQueries.consultSpecifies();break;
            case 6: BookStoreQueries.getSalesPercentage(dbManager);break;
            case 7: BookStoreQueries.getSalesByTitle(dbManager);break;

            case 8: System.out.println("Saliendo....");break;
            default: System.out.println("Opcion invalida. Intente
nuevamente.");break;

            }

        }while(option != 8);

        dbManager.disconnect();

        }

    }
```

```
package bookapp;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Scanner;



public class DatabaseManager {
    Scanner sc = new Scanner(System.in);
    private Connection connection;

    //conectando la base de datos
```

```
    public Connection connect() {
        try {
            connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/pubs","root","12345");

            System.out.println("Conexion a la base de datos exitosa.");

        }catch (SQLException e) {
        System.out.println("Error en la conexion a la base de datos: " +
e.getMessage());
        e.printStackTrace();
    }
        return connection;
}


public Connection getConnection() {
    if (connection == null) {
        connect(); // Asegura que la conexion este establecida
    }
    return connection;
}


public void disconnect() {
    try {
        if(connection != null && !connection.isClosed()) {
            connection.close();

            System.out.println("Conexion cerrada.");
        }
    }catch (SQLException e) {
        System.out.println("Error al cerrar la conexion: " + e.getMessage());
        }
    }




public class TableObject {
    private int id; // Número identificador de la tabla
    private String tableName; // Nombre de la tabla

    // Constructor
    public TableObject(int id, String tableName) {
        this.id = id;
        this.tableName = tableName;
    }

    // Getter para id
    public int getId() {
        return id;
    }
```

```
    // Getter para tableName
    public String getTableName() {
        return tableName;
    }

    // Método toString (opcional) para mostrar el objeto en formato legible
    @Override
    public String toString() {
        return id + ". " + tableName;
    }
}




public ArrayList<TableObject> showTables() {
    ArrayList<TableObject> tables = new ArrayList<>();
    String query = "SHOW TABLES";

    try (Statement stmt = connection.createStatement();
         ResultSet resultSet = stmt.executeQuery(query)) {

        System.out.println("\n-----------------------------------------");
        System.out.println("Tablas disponibles en la base de datos:");
        System.out.println("-----------------------------------------");

        int index = 1;
        while (resultSet.next()) {
            String tableName = resultSet.getString(1);
            TableObject table = new TableObject(index, tableName);
            tables.add(table);
            System.out.println(index + ". " + tableName);
            index++;
        }

        if (tables.isEmpty()) {
            System.out.println("No hay tablas disponibles.");
        }

        try (// Preguntar si desea continuar
        Scanner sc = new Scanner(System.in)) {
            String response;
            boolean exit = false;

            // Usamos un do-while para ejecutar al menos una vez
            do {
                System.out.print("\n¿Desea ver los datos de alguna tabla? (s/n):
");
                response = sc.nextLine();

                if (response.equalsIgnoreCase("s")) {
                    System.out.print("Ingrese el número de la tabla que desea ver:
");
                    int tableIndex = sc.nextInt();
```

```
                    sc.nextLine(); // Consumir la línea pendiente

                    // Asegurarse de que el índice esté dentro de los límites
                    if (tableIndex > 0 && tableIndex <= tables.size()) {
                        String tableName = tables.get(tableIndex -
1).getTableName();
                        showTableData(tableName);  // Llamar al método para
mostrar los datos de la tabla
                    } else {
                        System.out.println("Número de tabla no válido.");
                    }
                } else if (response.equalsIgnoreCase("n")) {
                    exit = true; // Salir del ciclo
                } else {
                    System.out.println("Respuesta no válida. Por favor ingrese 's'
para sí o 'n' para no.");
                }

            } while (!exit); // Continuar hasta que la respuesta sea 'n'
        }

    } catch (SQLException e) {
        System.out.println("Error al mostrar las tablas: " + e.getMessage());
    }

    return tables;
}




public void showTableData(String tableName) {
    String query = "SELECT * FROM " + tableName;

    try (Statement stmt =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
         ResultSet resultSet = stmt.executeQuery(query)) {

        ResultSetMetaData metaData = resultSet.getMetaData();
        int columnCount = metaData.getColumnCount();

        // Calcular el ancho máximo de cada columna (nombre de columna y datos)
        int[] columnWidths = new int[columnCount];
        for (int i = 1; i <= columnCount; i++) {
            columnWidths[i - 1] = metaData.getColumnName(i).length();
        }

        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                String value = resultSet.getString(i);
                if (value != null) {
                    columnWidths[i - 1] = Math.max(columnWidths[i - 1],
```

```
value.length());
                } else {
                    columnWidths[i - 1] = Math.max(columnWidths[i - 1], 4);  //
"null"
                }
            }
        }

        // Mostrar encabezados de las columnas
        System.out.println("\nDatos de la tabla: " + tableName);
        for (int i = 1; i <= columnCount; i++) {
            System.out.print(String.format("%-" + columnWidths[i - 1] + "s",
metaData.getColumnName(i)) + " | ");
        }
        System.out.println();
        System.out.println("-".repeat(columnWidths.length * 4));

        // Volver a ejecutar la consulta para mostrar los datos
        resultSet.beforeFirst();
        while (resultSet.next()) {
            for (int i = 1; i <= columnCount; i++) {
                String value = resultSet.getString(i);
                // Si el valor es nulo, mostrar "N/A"
                System.out.print(String.format("%-" + columnWidths[i - 1] + "s",
value != null ? value : "N/A") + " | ");
            }
            System.out.println();
        }

    } catch (SQLException e) {
        System.out.println("Error al mostrar los datos de la tabla: " +
e.getMessage());
    }
}









// Método para obtener las columnas de una tabla
public ArrayList<String> getTableColumns(String tableName) {M
    ArrayList<String> columns = new ArrayList<>();
    try {
        String query = "DESCRIBE " + tableName;
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            columns.add(rs.getString("Field"));
        }
    } catch (SQLException e) {
        e.printStackTrace();
```

```
    }
    return columns;
}

// Método para obtener los datos de una columna específica
public ArrayList<String> getColumnData(String tableName, String columnName) {
    ArrayList<String> data = new ArrayList<>();
    try {
        String query = "SELECT " + columnName + " FROM " + tableName;
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            data.add(rs.getString(columnName));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return data;
}

// Método para mostrar los detalles completos de una tienda
public void showTableDetails(String tableName, String column, String value) {
    // Crear consulta SQL dinámica para obtener detalles
    String query = "SELECT * FROM " + tableName + " WHERE " + column + " = ?";
    try (Connection conn = this.connect();
         PreparedStatement stmt = conn.prepareStatement(query)) {

        stmt.setString(1, value);  // Establecer el valor del parámetro

        try (ResultSet rs = stmt.executeQuery()) {
            // Mostrar los detalles de la fila
            ResultSetMetaData rsMetaData = rs.getMetaData();
            int columnCount = rsMetaData.getColumnCount();
            while (rs.next()) {
                for (int i = 1; i <= columnCount; i++) {
                    String columnName = rsMetaData.getColumnName(i);
                    String columnValue = rs.getString(i);
                    System.out.println(columnName + ": " + columnValue);
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void showTablesAndData() {
    ArrayList<TableObject> tables = showTables(); // Obtener las tablas
    if (tables.isEmpty()) {
        System.out.println("No hay tablas disponibles.");
    } else {
        // Mostrar los datos de la primera tabla (sin preguntar más)
        System.out.print("Proceso de ver tablas terminado");;
    }
```

```
}

}
```

```
package bookapp;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Scanner;


public class BookStoreQueries {
     // Método para conectar a la base de datos
    private Connection connect() {
        String url = "jdbc:mysql://localhost:3306/pubs";  // Ajusta la URL de tu
base de datos
        String user = "root";  // Tu usuario de base de datos
        String password = "12345";  // Tu contraseña de base de datos

        try {
            // Establecer conexión con la base de datos
            Connection conn = DriverManager.getConnection(url, user, password);
            System.out.println("Conexión exitosa a la base de datos.");
            return conn;
        } catch (SQLException e) {
            System.out.println("Error al conectar a la base de datos.");
            e.printStackTrace();
            return null;
        }
    }


    // Obtener los nombres de las tablas en la base de datos
    public ArrayList<String> getTableNames() {
        ArrayList<String> tables = new ArrayList<>();
        String query = "SHOW TABLES";  // Query para obtener todas las tablas
        try (Connection conn = this.connect();  // Llamamos a nuestro propio
método connect
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {
                tables.add(rs.getString(1));  // Agregar el nombre de cada tabla a
la lista
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return tables;
    }

    // Obtener las columnas de una tabla específica
    public ArrayList<String> getTableColumns(String tableName) {
        ArrayList<String> columns = new ArrayList<>();
        String query = "DESCRIBE " + tableName;  // Query para obtener las
columnas de la tabla
        try (Connection conn = this.connect();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {
                columns.add(rs.getString(1));  // Agregar el nombre de cada
columna a la lista
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return columns;
    }

    // Obtener los datos de una columna específica de una tabla
    public ArrayList<String> getColumnData(String tableName, String columnName) {
        ArrayList<String> data = new ArrayList<>();
        String query = "SELECT " + columnName + " FROM " + tableName;  // Query
para obtener los datos de la columna
        try (Connection conn = this.connect();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {

            while (rs.next()) {
                data.add(rs.getString(1));  // Agregar los datos de la columna a
la lista
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return data;
    }

    // Mostrar los detalles de una opción seleccionada de una tabla
    public void showTableDetails(String tableName, String columnName, String
value) {
        String query = "SELECT * FROM " + tableName + " WHERE " + columnName + " =
?";
        try (Connection conn = this.connect();
             PreparedStatement stmt = conn.prepareStatement(query)) {

            stmt.setString(1, value);  // Establecer el valor a consultar
            ResultSet rs = stmt.executeQuery();
```

```
            while (rs.next()) {
                // Mostrar los detalles de todos los campos de la fila
seleccionada
                int columnCount = rs.getMetaData().getColumnCount();
                for (int i = 1; i <= columnCount; i++) {
                    String columnNameDB = rs.getMetaData().getColumnName(i);
                    String columnValue = rs.getString(i);
                    System.out.println(columnNameDB + ": " + columnValue);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


 // Método para consultar y obtener datos específicos de la tabla
    public static void consultSpecifies() {
        // Crear instancia de BookStoreQueries para acceder a sus métodos
        BookStoreQueries dbManager = new BookStoreQueries();

        // Obtener todas las tablas de la base de datos
        ArrayList<String> tables = dbManager.getTableNames();

        // Mostrar las tablas disponibles
        System.out.println("Tablas disponibles en la base de datos:");
        for (int i = 0; i < tables.size(); i++) {
            System.out.println((i + 1) + ". " + tables.get(i));
        }

        try (// Crear un objeto Scanner para leer la entrada del usuario
        Scanner sc = new Scanner(System.in)) {
            String response = null;

            // Iniciar un ciclo para permitir múltiples consultas
            do {
                // Selección de la tabla
                System.out.print("\nOPCION PARA BUSCAR DATOS ESPECIFICOS EN LAS
TABLAS ");
                System.out.print("\n-------------------------------------------
-----");
                System.out.print("\nSeleccione una tabla para consultar por
nombre: ");
                String tableName = sc.nextLine();

                // Validar que la tabla exista en la base de datos
                if (!tables.contains(tableName)) {
                    System.out.println("Tabla no válida. Por favor seleccione una
tabla válida.");
                    continue;
                }

                // Mostrar las columnas de la tabla seleccionada
```

```
                ArrayList<String> columns = dbManager.getTableColumns(tableName);
// Obtener las columnas de la tabla seleccionada
                System.out.println("Columnas disponibles en la tabla '" +
tableName + "':");
                for (int i = 0; i < columns.size(); i++) {
                    System.out.println((i + 1) + ". " + columns.get(i));
                }

                // Selección de la columna
                System.out.print("\n-----------------------------------------");
                System.out.print("\n-----------------------------------------");
                System.out.print("\nSeleccione una columna para consultar por
numero: ");
                int columnIndex = sc.nextInt();
                sc.nextLine(); // Consumir nueva línea

                if (columnIndex < 1 || columnIndex > columns.size()) {
                    System.out.println("Selección inválida.");
                    continue;
                }

                String selectedColumn = columns.get(columnIndex - 1);

                // Mostrar los valores de la columna seleccionada
                System.out.println("Valores de la columna '" + selectedColumn +
"':");
                ArrayList<String> columnData = dbManager.getColumnData(tableName,
selectedColumn); // Obtener los datos de la columna
                for (int i = 0; i < columnData.size(); i++) {
                    System.out.println((i + 1) + ". " + columnData.get(i));
                }

                // Selección de un valor de la columna
                System.out.print("\n-----------------------------------------");
                System.out.print("\n-----------------------------------------");
                System.out.print("\nSeleccione una opción por numero para ver más
detalles: ");
                int rowIndex = sc.nextInt();
                sc.nextLine(); // Consumir nueva línea

                if (rowIndex < 1 || rowIndex > columnData.size()) {
                    System.out.println("Selección inválida.");
                    continue;
                }

                String selectedValue = columnData.get(rowIndex - 1);

                // Mostrar detalles de la opción seleccionada
                System.out.println("\nDetalles de la opción seleccionada (" +
selectedValue + "):");
                dbManager.showTableDetails(tableName, selectedColumn,
selectedValue); // Mostrar detalles de la opción seleccionada

                // Preguntar si desea continuar
```

```
                System.out.print("\n¿Desea realizar otra consulta? (s/n): ");
                response = sc.nextLine();

            } while (response.equalsIgnoreCase("s"));
        }
        // Mensaje de despedida al finalizar
        System.out.println("Gracias por usar el sistema de consultas.");
    }







 // Método para consultar ventas por título
    public static void getSalesPercentage(DatabaseManager dbManager) {
        Connection conn = dbManager.getConnection();

        try {
            // Cambiar el tipo de ResultSet a TYPE_SCROLL_INSENSITIVE
            String query = "SELECT price, ytd_sales, title FROM titles";
            PreparedStatement stmt = conn.prepareStatement(query,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
            ResultSet rs = stmt.executeQuery();

            double totalSales = 0;
            while (rs.next()) {
                // Sumar las ventas totales (price * ytd_sales) para todos los
títulos
                double price = rs.getDouble("price");
                int ytdSales = rs.getInt("ytd_sales");
                totalSales += price * ytdSales;
            }

            // Volver al inicio del ResultSet
            rs.beforeFirst();

            // Calcular el porcentaje de ventas por título
            System.out.println("\n--- PORCENTAJE DE VENTAS POR TITULOS ---");
            System.out.print("\n------------------------------------------------
-----------------------------------\n");

            // Mostrar el porcentaje de ventas por título
            while (rs.next()) {
                double price = rs.getDouble("price");
                int ytdSales = rs.getInt("ytd_sales");

                if (totalSales > 0) {
                    // Calcular porcentaje de ventas
                    double salesValue = price * ytdSales;
                    double salesPercentage = (salesValue / totalSales) * 100;
                    System.out.printf("Título: %s | Porcentaje de Ventas:
%.2f%%\n", rs.getString("title"), salesPercentage);
```

```
                } else {
                    System.out.println("No se encontraron ventas para calcular.");
                }
            }

            // Al finalizar la ejecución, el método termina y regresa al menú de
opciones (o lo que sea que venga después)

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }







    public static void getSalesByTitle(DatabaseManager dbManager) {
        String query = "SELECT title, price, ytd_sales FROM titles";  //
Seleccionar los campos que necesitamos
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        Scanner scanner = new Scanner(System.in);  // Mover el Scanner fuera del
ciclo

        try {
            // Obtener la conexión
            conn = dbManager.getConnection();

            // Crear el Statement
            stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

            // Ejecutar la consulta
            rs = stmt.executeQuery(query);

            // Hacer que el proceso de selección se repita si el usuario lo desea
            boolean continueSelection = true;

            do {
                // Listar todos los títulos con su índice
                System.out.print("\n-----------------------------------------
\n");
                System.out.println("Selecciona un título de la lista introduciendo
el número correspondiente:");
                System.out.print("-----------------------------------------\n");

                // Crear una lista de títulos para permitir la selección
                int index = 1;
                while (rs.next()) {
                    String title = rs.getString("title");
                    System.out.println(index + ". " + title);
```

```
                    index++;
                }

                // Solicitar la selección del usuario
                System.out.print("\n-----------------------------------------
\n");
                System.out.print("Ingresa el número del título que deseas ver: ");
                int selectedIndex = scanner.nextInt();

                // Reestablecer el ResultSet para recorrerlo nuevamente
                rs.beforeFirst();

                // Variable para acumular las ventas totales de todos los títulos
                double grandTotalSales = 0.0;

                // Calcular las ventas totales acumuladas
                while (rs.next()) {
                    double price = rs.getDouble("price");
                    int ytdSales = rs.getInt("ytd_sales");

                    // Ignorar registros con datos incompletos
                    if (!rs.wasNull() && price != 0.0 && ytdSales != 0) {
                        grandTotalSales += price * ytdSales; // Acumular el total
                    }
                }

                // Reestablecer el ResultSet nuevamente para encontrar el título
seleccionado
                rs.beforeFirst();
                int currentIndex = 1;
                while (rs.next()) {
                    // Verificar si el índice actual corresponde al título
seleccionado
                    if (currentIndex == selectedIndex) {
                        String title = rs.getString("title");
                        double price = rs.getDouble("price");
                        int ytdSales = rs.getInt("ytd_sales");

                        // Calcular el total de ventas por título
                        double totalSales = price * ytdSales;

                        // Imprimir la información del título seleccionado
                        System.out.println("\n-----------------------------------
----");
                        System.out.println("Título seleccionado: " + title);
                        System.out.println("Precio: $" + price);
                        System.out.println("Ventas acumuladas: " + ytdSales);
                        System.out.println("Total de ventas por título: $" +
totalSales);
                        System.out.println("\n-----------------------------------
----");

                        // Mostrar también las ventas totales acumuladas de todos
los títulos
```

```
                        System.out.printf("Ventas totales acumuladas de todos los
títulos: $%.2f%n", grandTotalSales);
                        System.out.println("----------------------------------------
--");
                        break;  // Terminamos el ciclo una vez que se ha mostrado
el título seleccionado
                    }
                    currentIndex++;
                }

                // Preguntar si el usuario desea continuar
                System.out.print("\n¿Deseas continuar? (s/n): ");
                String respuesta = scanner.next();
                if (respuesta.equalsIgnoreCase("n")) {
                    continueSelection = false;  // Salir del ciclo
                } else {
                    // Reestablecer el ResultSet para el próximo ciclo
                    rs.beforeFirst();
                }

            } while (continueSelection);  // Continuar mientras el usuario lo
desee

        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                // Eliminar el cierre de la conexión aquí
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }


}
```

```
package bookapp;


import java.sql.Connection;
import java.sql.DatabaseMetaData;
```

```
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class BookStoreManager {
    Scanner sc = new Scanner(System.in);

    // Método para obtener la conexión a la base de datos
    public static Connection getConnection() throws SQLException {
            return DriverManager.getConnection("jdbc:mysql://localhost:3306/pubs",
"root", "12345"); // Ajusta esto según tu configuración
      }


    public static void addField(String tableName, Scanner sc) {
        try (Connection connection = getConnection()) {
            // Consulta para obtener las columnas de la tabla
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet columns = metaData.getColumns(null, null, tableName, null);

            // Mostrar las columnas de la tabla
            System.out.println("Columnas de la tabla " + tableName + ":");
            StringBuilder columnNames = new StringBuilder();
            while (columns.next()) {
                columnNames.append(columns.getString("COLUMN_NAME")).append(", ");
            }
            // Eliminar la última coma y espacio
            columnNames.setLength(columnNames.length() - 2);
            System.out.println(columnNames);

            String response;

            do {
                // Obtener los valores para cada columna
                StringBuilder insertColumns = new StringBuilder();
                StringBuilder insertValues = new StringBuilder();
                StringBuilder constraints = new StringBuilder();
                columns = metaData.getColumns(null, null, tableName, null); //
reiniciar el result set

                while (columns.next()) {
                    String columnName = columns.getString("COLUMN_NAME");
                    System.out.print("\n----------------------------------------
-\n");
                    System.out.print("Ingrese el valor para la columna " +
columnName + ": ");
                    String value = sc.nextLine();

                    // Se agrega la columna y el valor en los respectivos
```

```
StringBuilder
                    insertColumns.append(columnName).append(", ");
                    insertValues.append("'").append(value).append("', ");

                    // Elegir si el valor es Primary Key, Not Null o Default Null
                    System.out.print("¿El valor para " + columnName + " será
Primary Key, Not Null o Default Null? (Ingrese: primary, notnull, default): ");
                    String constraint = sc.nextLine();

                    if ("primary".equalsIgnoreCase(constraint)) {
                        constraints.append("PRIMARY KEY, ");
                    } else if ("notnull".equalsIgnoreCase(constraint)) {
                        constraints.append("NOT NULL, ");
                    } else if ("default".equalsIgnoreCase(constraint)) {
                        constraints.append("DEFAULT NULL, ");
                    }
                }

                // Eliminar la última coma y espacio de las listas de columnas,
valores y restricciones
                if (insertColumns.length() > 0) {
                    insertColumns.setLength(insertColumns.length() - 2);
                }

                if (insertValues.length() > 0) {
                    insertValues.setLength(insertValues.length() - 2);
                }

                if (constraints.length() > 0) {
                    constraints.setLength(constraints.length() - 2);  //
Eliminamos la coma extra al final de las restricciones
                }

                // Crear la consulta SQL de inserción
                String sql = "INSERT INTO " + tableName + " (" + insertColumns +
") VALUES (" + insertValues + ")";
                System.out.println("Consulta SQL generada: " + sql);

                // Ejecutar la consulta de inserción
                try (Statement stmt = connection.createStatement()) {
                    int rowsAffected = stmt.executeUpdate(sql);
                    if (rowsAffected > 0) {
                        System.out.println("Fila insertada correctamente en la
tabla " + tableName);

                        // Manejo de las restricciones de la tabla si es necesario
                        if (constraints.length() > 0) {
                            String alterSQL = "ALTER TABLE " + tableName + " ADD
CONSTRAINT " + tableName + "_constraints (" + constraints + ")";
                            stmt.executeUpdate(alterSQL);
                        }
                    }
                }
```

```
                // Preguntar al usuario si desea continuar con la inserción de más
filas
                System.out.print("\n¿Desea continuar insertando otra fila? (s/n):
");
                response = sc.nextLine();

            } while (response.equalsIgnoreCase("s"));  // Continuar si la
respuesta es 's'

        } catch (SQLException e) {
            System.err.println("Error al acceder a la base de datos: " +
e.getMessage());
        }
    }




    public static void modifyField(Scanner sc) {
        try (Connection connection = getConnection()) {
            // Solicitar el nombre de la tabla y la columna a modificar
            System.out.print("Ingrese el nombre de la tabla: ");
            String tableName = sc.nextLine();

            // Obtener los nombres de las columnas
            DatabaseMetaData metaData = connection.getMetaData();
            ResultSet columns = metaData.getColumns(null, null, tableName, null);

            // Mostrar las columnas disponibles
            System.out.println("Columnas en la tabla " + tableName + ":");
            while (columns.next()) {
                System.out.println(columns.getString("COLUMN_NAME"));
            }

            // Seleccionar la columna a modificar
            System.out.print("\n-----------------------------------------");
            System.out.print("\nIngrese el nombre de la columna a modificar: ");
            String columnNameToModify = sc.nextLine();

            // Obtener los valores actuales de la columna y su ID (o una columna
única)
            String selectQuery = "SELECT ID, " + columnNameToModify + " FROM " +
tableName;
            try (PreparedStatement selectStatement =
connection.prepareStatement(selectQuery);
                 ResultSet columnValues = selectStatement.executeQuery()) {

                // Mostrar los registros
                System.out.println("\nRegistros de la columna " +
columnNameToModify + ":");
                List<String> valuesList = new ArrayList<>();
                List<Integer> idsList = new ArrayList<>();
                int rowNumber = 1;
```

```
                while (columnValues.next()) {
                    int id = columnValues.getInt("ID"); // Suponiendo que la
columna ID es la clave primaria
                    String value = columnValues.getString(columnNameToModify);
                    valuesList.add(value);
                    idsList.add(id);
                    System.out.println(rowNumber++ + ". " + value + " (ID: " + id
+ ")");
                }

                // Solicitar el registro a modificar
                System.out.print("\n-----------------------------------------");
                System.out.print("\nSeleccione el número del registro a modificar:
");
                int recordNumber = sc.nextInt();
                sc.nextLine(); // Limpiar buffer

                if (recordNumber > 0 && recordNumber <= valuesList.size()) {
                    int recordId = idsList.get(recordNumber - 1); // Obtener el ID
de la fila seleccionada

                    // Solicitar el nuevo valor
                    System.out.print("\nIngrese el nuevo valor para " +
columnNameToModify + ": ");
                    String newValue = sc.nextLine();

                    // Actualizar el registro usando el ID como referencia única
                    String updateSQL = "UPDATE " + tableName + " SET " +
columnNameToModify + " = ? WHERE ID = ?";
                    try (PreparedStatement updateStatement =
connection.prepareStatement(updateSQL)) {
                        updateStatement.setString(1, newValue);
                        updateStatement.setInt(2, recordId); // Usamos el ID único
para asegurar que se modifique solo esa fila
                        int rowsAffected = updateStatement.executeUpdate();
                        System.out.println(rowsAffected > 0 ? "Registro
actualizado exitosamente." : "No se pudo actualizar el registro.");
                    } catch (SQLException e) {
                        System.out.println("Error al actualizar el registro: " +
e.getMessage());
                    }
                } else {
                    System.out.println("Número de registro no válido.");
                }
            }

            // Preguntar al usuario si desea continuar
            System.out.print("\n¿Desea continuar modificando otro registro? (s/n):
");
            String response = sc.nextLine();
            if (response.equalsIgnoreCase("s")) {
                modifyField(sc);  // Llamada recursiva para continuar
            } else {
                System.out.println("Proceso de modificación terminado.");
```

```
            }

        } catch (SQLException e) {
            System.out.println("Error al acceder a la base de datos: " +
e.getMessage());
        }
    }



    public static void deleteRow(String tableName, Scanner sc) {
        try (Connection connection = getConnection()) {
            boolean continueDeletion = true; // Bandera para controlar si el
usuario desea continuar

            while (continueDeletion) {
                // Obtener los nombres de las columnas de la tabla
                DatabaseMetaData metaData = connection.getMetaData();
                ResultSet columns = metaData.getColumns(null, null, tableName,
null);

                // Mostrar las columnas de la tabla
                System.out.println("Columnas de la tabla " + tableName + ":");
                StringBuilder columnNames = new StringBuilder();
                while (columns.next()) {
                    columnNames.append(columns.getString("COLUMN_NAME")).append(",
");
                }

                // Eliminar la última coma y espacio
                if (columnNames.length() > 0) {
                    columnNames.setLength(columnNames.length() - 2);
                }
                System.out.println(columnNames);

                // Solicitar al usuario seleccionar una columna
                System.out.print("\n----------------------------------------
\n");
                System.out.print("Ingrese el nombre de la columna por la cual
desea eliminar la fila: ");
                String columnToDelete = sc.nextLine();

                // Verificar el tipo de la columna para asegurarse de que sea
String o int
                boolean isValidColumn = false;
                columns = metaData.getColumns(null, null, tableName,
columnToDelete); // Volver a obtener los datos de la columna seleccionada

                while (columns.next()) {
                    String columnType = columns.getString("TYPE_NAME");
                    // Permitir tanto columnas de tipo String como numéricas (INT)
                    if (columnType.equalsIgnoreCase("VARCHAR") ||
columnType.equalsIgnoreCase("CHAR") || columnType.equalsIgnoreCase("INT")) {
                        isValidColumn = true;
```

```
                    }
                }

                if (!isValidColumn) {
                    System.out.println("La columna seleccionada no es válida para
eliminación. Solo se permiten columnas de tipo String o int.");
                    continue; // Volver a pedir la columna si no es válida
                }

                // Mostrar los valores de la columna seleccionada
                String selectQuery = "SELECT " + columnToDelete + " FROM " +
tableName;
                try (PreparedStatement selectStatement =
connection.prepareStatement(selectQuery);
                     ResultSet resultSet = selectStatement.executeQuery()) {

                    System.out.println("\nValores de la columna " + columnToDelete
+ ":");
                    int rowNumber = 1;
                    while (resultSet.next()) {
                        String value = resultSet.getString(1);
                        System.out.println(rowNumber++ + ". " + value);
                    }
                }

                // Solicitar el valor para la columna seleccionada
                System.out.print("\n----------------------------------------
\n");
                System.out.print("Ingrese el valor de la columna " +
columnToDelete + " para eliminar la fila: ");
                String valueToDelete = sc.nextLine();

                // Crear la consulta SQL para eliminar la fila usando la columna
seleccionada
                String sql = "DELETE FROM " + tableName + " WHERE " +
columnToDelete + " = ?";

                // Ejecutar la consulta DELETE
                try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
                    pstmt.setString(1, valueToDelete);
                    int rowsAffected = pstmt.executeUpdate();
                    if (rowsAffected > 0) {
                        System.out.println("Fila eliminada correctamente de la
tabla " + tableName);
                    } else {
                        System.out.println("No se encontró ninguna fila con el
valor proporcionado para eliminar.");
                    }
                }

                // Preguntar al usuario si desea continuar con la eliminación o
salir
                System.out.print("\n¿Desea eliminar otra fila? (s/n): ");
                String response = sc.nextLine().trim().toLowerCase();
```

```
                if (!response.equals("s")) {
                    continueDeletion = false; // Terminar el ciclo si la respuesta
no es 's'
                    System.out.println("Operación de eliminación finalizada.");
                    // Aquí puedes regresar al menú principal o realizar alguna
otra acción
                    return; // Esto termina el método y regresa al flujo de
selección de tabla
                }
            }

        } catch (SQLException e) {
            System.err.println("Error al acceder a la base de datos: " +
e.getMessage());
        }
    }



}
```