Tarea UD11: Acceso a Bases de datos

Programación

Curso DAM 2024/25

Calificación obtenida: 9/10

Como mejora, estaría bien permitir la prueba de cada una de las opciones con un menú.

Índice

Detalles de la tarea..... 3
Configuración del SGBD..... 3
Crear un proyecto para realizar operaciones contra la Base de datos..... 4
Informe..... 7
Ventajas de la separación del código de las clases que acceden a la BBDD de las clases que implementan la lógica de negocio..... 7
Propuesta para no tener que crear el objeto Connection en la clase principal..... 8
Anexo..... 9
Instalación de MariaDB y HeidiSQL..... 9
Crear una Base de datos desde HeidiSQL..... 14
Instalar el conector de la base de datos..... 19
Código..... 21
Clase ConnectionDB..... 21
Clase PropietariosDAO..... 23
Clase VehiculosDAO..... 29
Clase Prog11_Principal..... 37

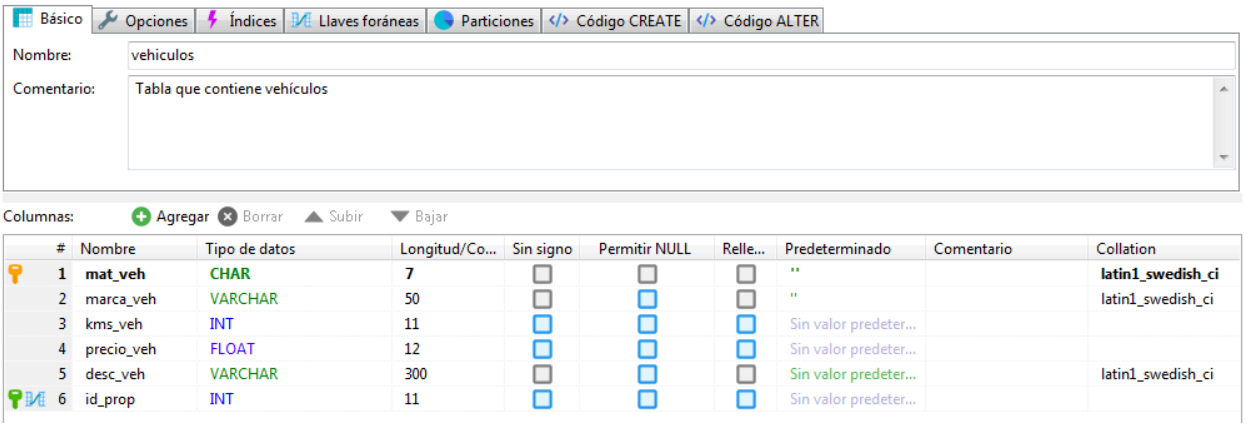
Detalles de la tarea

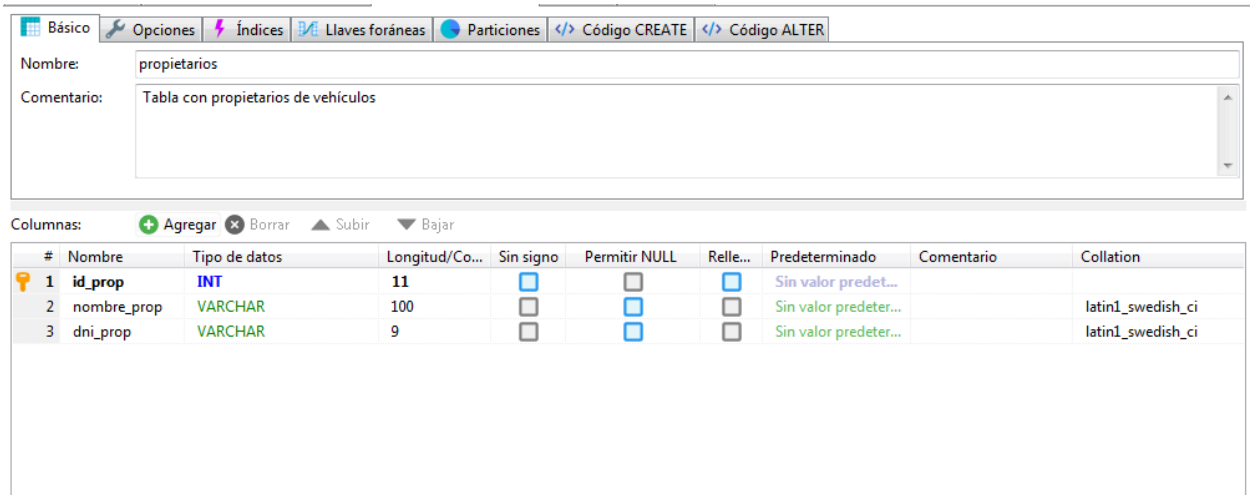
En esta última unidad vamos a implementar una pequeña aplicación que utilice una conexión contra una base de datos relacional, que estará montada sobre MariaDB. Sigue los siguientes pasos:

Configuración del SGBD

- Siguiendo los pasos de los contenidos, instala en tu equipo MariaDB y HeidiSQL como aplicación para acceso y administración de bases de datos en MariaDB. Como ya sabes, no importa el sistema operativo que utilices, hay versiones para los más utilizados.
- Realizada la instalación, inicia el SGBD MariaDB e inicia la aplicación HeidiSQL. Deberás introducir las credenciales de conexión a MariaDB.
- Crea una base de datos desde HeidiSQL denominada **Concesionario**.
- Dentro de esta base de datos deberás crear dos tablas:
    - **Vehículos**: almacenará una serie de vehículos.
    - **Propietarios**: almacenará los propietarios de los vehículos de la tabla anterior.

Observa en las siguientes imágenes la estructura de cada una de las tablas. Observa como en la tabla vehículos, el campo **id_prop** es una clave externa hacia la tabla propietarios.





El campo **id_prop** de la tabla **Propietarios** puede tener un valor **auto_incremento**.

Utilizando HeidiSQL, inserta datos en ambas tablas que pueden servir para realizar pruebas. Introduce propietarios que pueden tener varios vehículos.

Crear un proyecto para realizar operaciones contra la Base de datos

Hasta este punto tenemos la parte de configuración del SGBD. El objetivo desde nuestra aplicación es implementar varias clases que se encarguen de realizar las operaciones contra la base de datos. Para ello:

- Crea un nuevo proyecto denominado **PROG11_tarea**. Configúralo con el driver necesario para acceder al SGBD MariaDB instalado anteriormente.
- Crea un paquete denominado **com.prog11.bbdd**.
    - Dentro del paquete crea la **clase ConnectionDB** que se encargue de realizar la conexión contra la base de datos y el cierre de la misma. Para ello implementará los métodos **openConnection()** y **closeConnection()**.
    - Crea una clase estática denominada **PropietariosDAO.java** que contenga los métodos necesarios para realizar operaciones contra la tabla propietarios.
        - **Insertar un nuevo propietario:** Recibe por parámetro los datos de un propietario a insertar, trata de insertarlo en la BBDD y devuelve 0 si la operación se realizó con éxito o -1 en caso contrario.
        - **Recuperar vehículos de un propietario**: Recibe por parámetro el DNI de un propietario y devuelve una lista con la matrícula, marca, número de kms y precio de todo sus vehículos. Retornará null si hubo problemas.
        - **Eliminar propietario**: Recibe por parámetro el DNI de un propietario y trata de eliminarlo de la BBDD. Devuelve el número de registros eliminados.

- Crea una clase estática denominada **VehiculosDAO.java** que contenga los métodos necesarios para realizar operaciones contra la tabla Vehiculos. Deberá contener métodos para:
  - **Insertar un nuevo vehículo**: Recibe por parámetro los datos del vehículo a insertar, trata de insertarlo en la BBDD y devuelve 0 si la operación se realizó con éxito o -1 en caso contrario.
  - **Actualizar propietario de vehículo**: Recibe por parámetro la matrícula de un vehículo junto al identificador de un propietario y trata de actualizar el vehículo en la BBDD. Devuelve 0 si la operación se realizó con éxito o -1 si el vehículo no existe.
  - **Eliminar vehículo**: Recibe por parámetro la matrícula de un vehículo y trata de eliminarlo de la BBDD. Devuelve 0 si la operación se realizó con éxito o -1 si el vehículo no existe.
  - **Recuperar todos los vehículos**: No recibe parámetros y devuelve una lista con todos los vehículos del concesionario.Para cada vehículo, la lista contendrá una cadena de caracteres con los datos del mismo, incluido el nombre del propietario
  - **Recuperar vehículos por marca**: Recibe una marca por parámetro y devuelve una lista con el listado de vehículos de la citada marca. Para cada vehículo, la lista contendrá una cadena de caracteres con los datos del mismo, incluido el nombre del propietario. Si no existen vehículos, devuelve el ArrayList vacío.
  - **Recuperar vehículos**: No recibe parámetros (solo la coneción con la BBDD) y retorna una lista con la matrícula, marca, kilómetros y precio de cada vehículo.

Para los métodos que devuelven una lista, ten en cuenta que los elementos de la misma serán cadenas de caracteres.

- Crea otro paquete, denominado **com.prog11.princ** que contendrá la clase Principal de la aplicación.
  - Dentro este paquete, crea una clase denominada **Prog11_Principal.java** que contenga el método **main.**
  - Inserta el código necesario en esta clase para poder probar cada uno de los métodos implementados en las clases de acceso a la BBDD, , siguiendo el siguiente orden
    - Insertar varios vehículos y propietarios.
    - Listar todos los vehículos.
    - Actualizar propietario de un vehículo.
    - Listar todos los vehículos.
    - Eliminar un vehículo que exista.
    - Eliminar un vehículo que no exista.
    - Listar todos los vehículos.
    - Listar los vehículos de una marca.

- Listar todos los vehículos de un propietario.
- Eliminar un propietario con vehículos.
- Eliminar un propietario sin vehículos.

IMPORTANTE:

- La conexión se creará en la clase principal y se pasará a los métodos que la necesiten como parámetro.
- Los métodos de las clases DAO no devuelven objetos de tipo **ResultSet**, deben devolver un arraylist de **Strings**.
- Las excepciones de los clases que interactúan con la BBDD deben tratarse dentro de la clase.

El alumno puede añadir cualquier funcionalidad adicional que considere oportuna, siempre y cuando suponga añadir funciones a las mínimas propuestas.

1. **¿Qué ventajas observas de la separación del código de las clases que acceden a la BBDD de las clases que implementan la lógica de negocio, en este caso, solo la clase principal?**
2. **¿Se te ocurre alguna propuesta para no tener que crear el objeto Connection en la clase principal?.**
3. **Escribe en un documento pdf tu reflexión.**

Informe

Ventajas de la separación del código de las clases que acceden a la BBDD de las clases que implementan la lógica de negocio

¿Qué ventajas observas de la separación del código de las clases que acceden a la BBDD de las clases que implementan la lógica de negocio, en este caso, solo la clase principal?

Separar el código de las clases que acceden a la base de datos (como las clases DAO) de las clases que implementan la lógica de negocio (como la clase principal Prog11_Principal) tiene varias ventajas importantes.

Si alguna vez se necesita modificar la forma en que se interactúa con la base de datos, ya sea cambiando el tipo de base de datos o actualizando alguna consulta, se pueden hacer esos cambios directamente en las clases encargadas de acceder a la base de datos sin tocar el resto del sistema. Esto hace que el código sea mucho más fácil de gestionar a medida que la aplicación crece o cambia.

Además, mejora la reutilización del código. Las clases que gestionan la base de datos están diseñadas para ser genéricas y se pueden utilizar en otras partes de la aplicación, o incluso en otros proyectos, sin tener que duplicar la lógica de acceso a datos. De esta forma, se ahorra tiempo y esfuerzo en lugar de repetir el mismo código en diferentes lugares.

También se mejora la organización del código. Cada clase tiene una responsabilidad clara y única: las clases DAO se encargan exclusivamente de las operaciones con la base de datos, mientras que la clase principal maneja la lógica de negocio y el flujo de la aplicación. Esto mejora la claridad del código, lo que facilita tanto su comprensión como su mantenimiento, especialmente si varias personas trabajan en el proyecto.

Si en algún momento se necesita escalar la aplicación, esta separación facilita la expansión. Si se desea agregar nuevas funcionalidades o modificar el comportamiento de la aplicación, se puede hacer sin necesidad de tocar las clases que interactúan con la base de datos. Por ejemplo, si en el futuro se necesita integrar nuevas fuentes de datos, como una API externa, solo será necesario agregar nuevas clases DAO, sin que eso afecte la lógica de negocio.

Por último, esta separación también facilita las pruebas. Se pueden probar las clases DAO por separado, simulando el acceso a la base de datos sin tener que conectarse a una base de datos real, lo que hace que las pruebas sean más rápidas y fáciles. Al mismo tiempo, se puede probar la lógica de negocio sin depender de la base de datos, lo que mejora la eficiencia de las pruebas y facilita el aislamiento de posibles errores.

Propuesta para no tener que crear el objeto Connection en la clase principal

¿Se te ocurre alguna propuesta para no tener que crear el objeto Connection en la clase principal?.

Una propuesta para no tener que crear el objeto Connection en la clase principal sería manejar la conexión dentro de las clases DAO. Si cada método de acceso a la base de datos abre y cierra la conexión por sí mismo, la clase principal solo llamaría a los métodos necesarios sin preocuparse por la conexión. Esto haría que el código fuera más limpio y fácil de mantener. Además, evitaría errores como olvidar cerrar la conexión, ya que cada método se encargaría de hacerlo automáticamente. Como resultado, la clase principal quedaría más enfocada en la lógica del programa y no en la gestión de la base de datos.

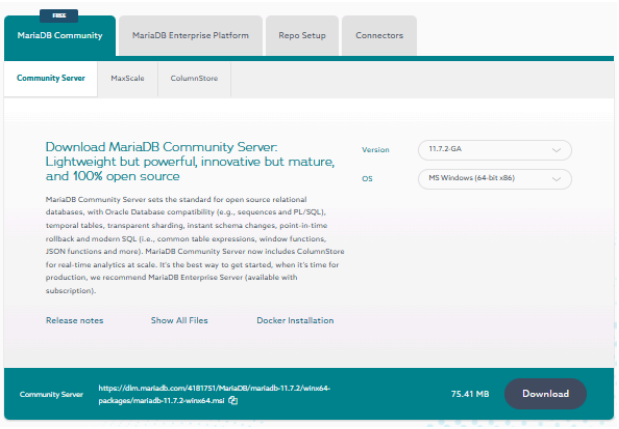
Anexo

Video explicativo de la instalación: https://youtu.be/7h8y-LEfMzA?si=9vB6g-Dso6J3-aty

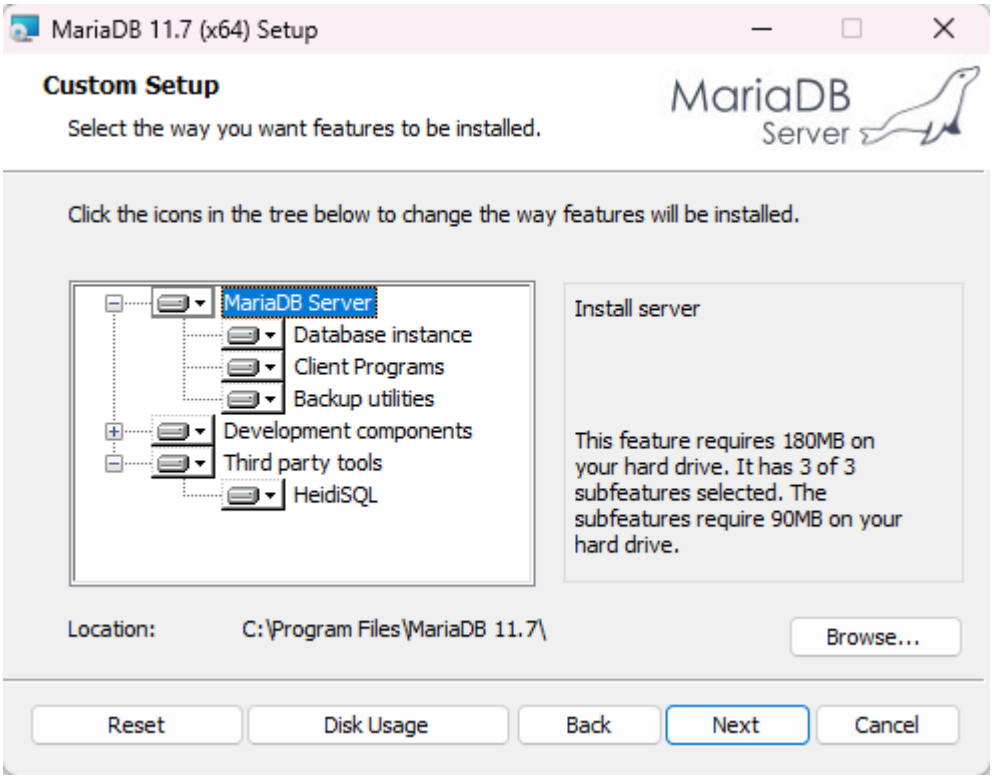
Instalación de MariaDB y HeidiSQL

Instalar MariaDB y HeidiSQL

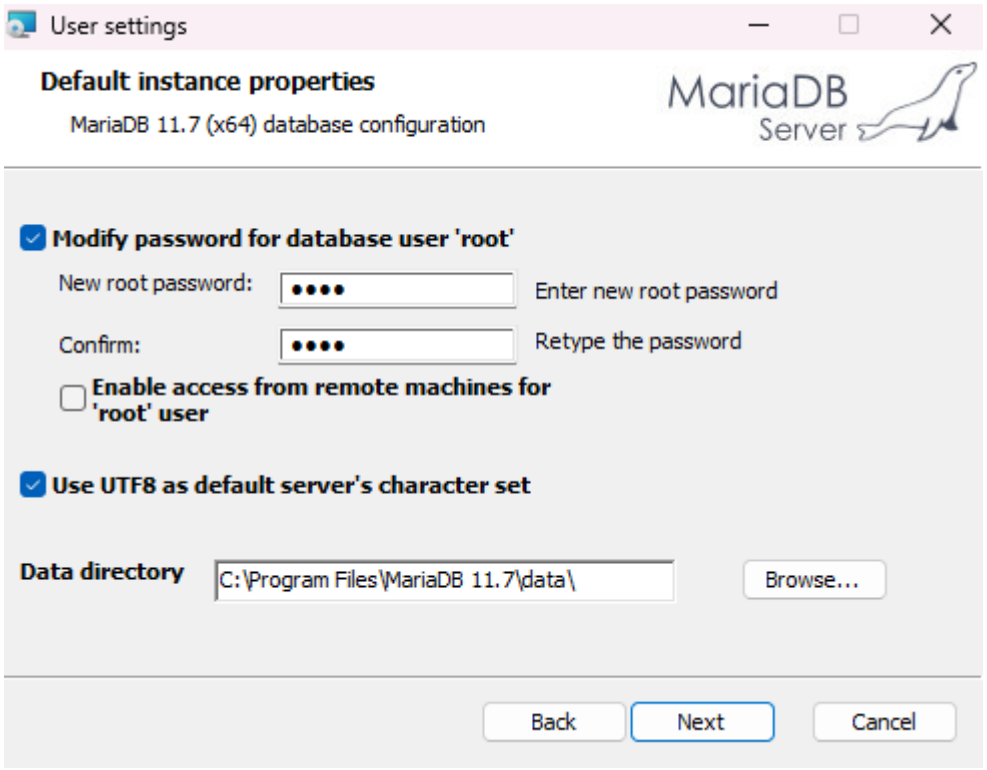
Ir a la página de descargas de la web de MariaDB



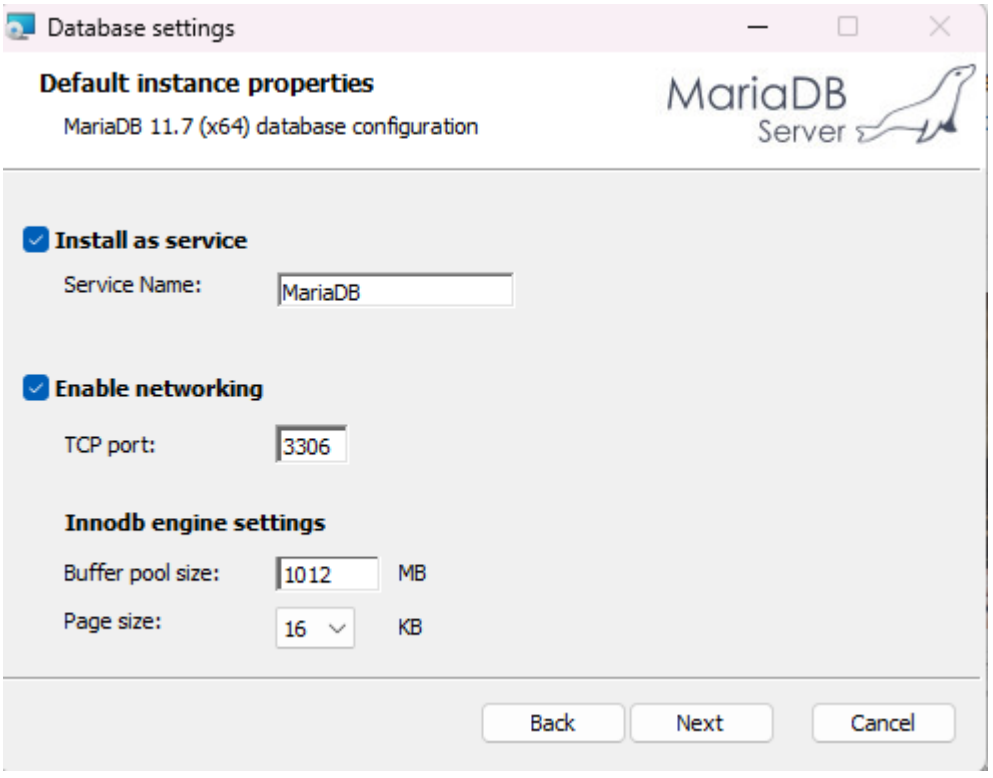
Ejecutar el archivo de instalación y asegurarnos que HeidiSQL está seleccionado



Elegir una contraseña para el usuario root (usamos root)



Dejamos el resto de opciones por defecto

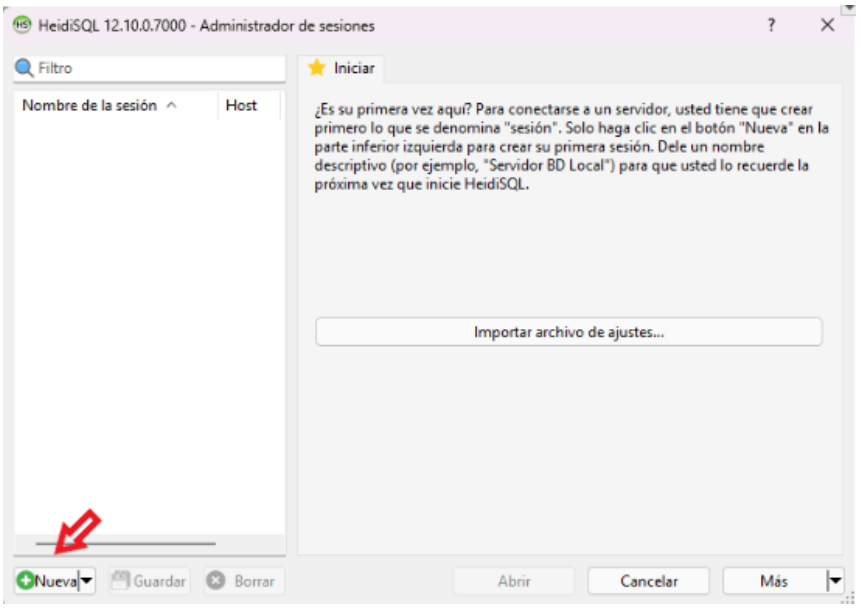


Probar HeidiSQL

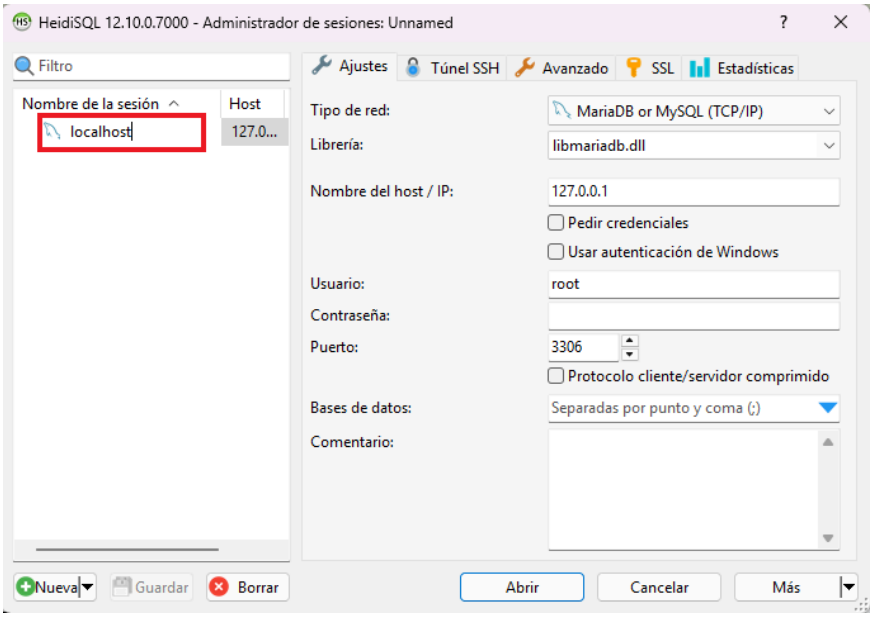
Abrir HeidiSQL desde el acceso directo que se ha creado en el escritorio del ordenador



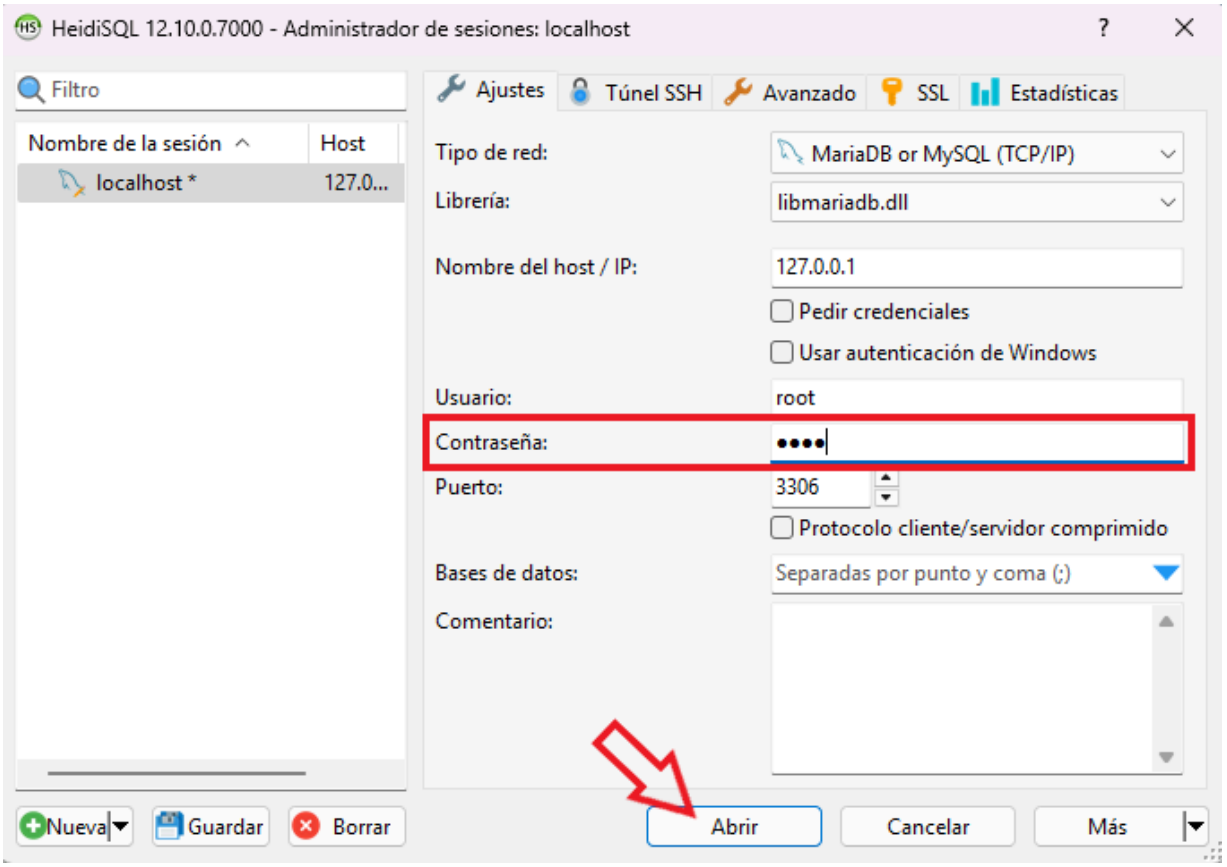
Crear un nueva conexión seleccionando “Nueva”



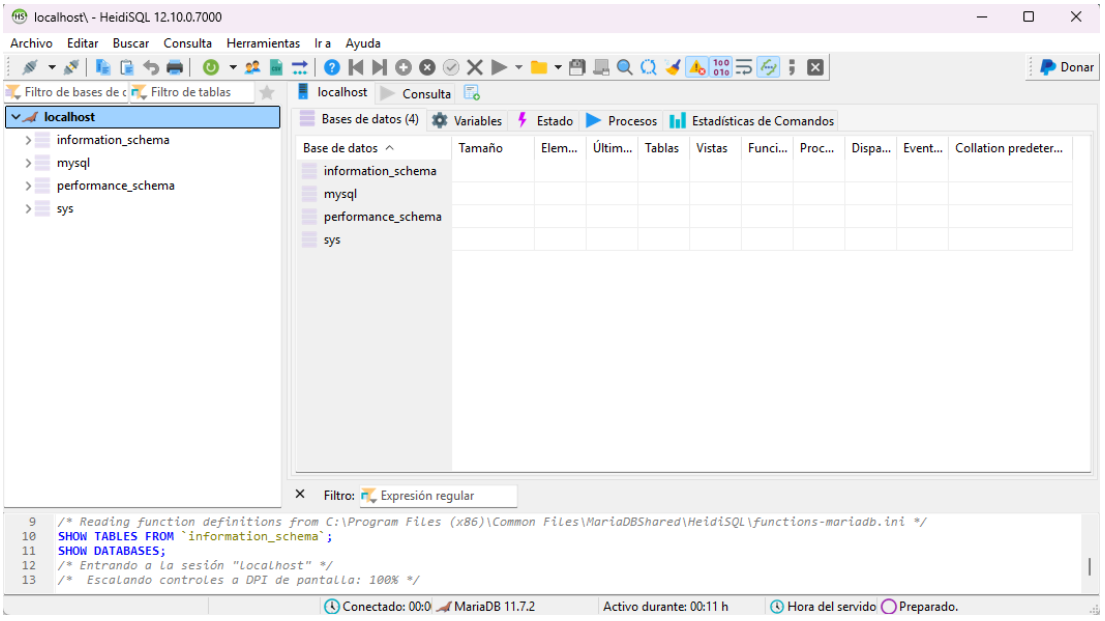
Poner el nombre de “localhost” a la nueva conexión



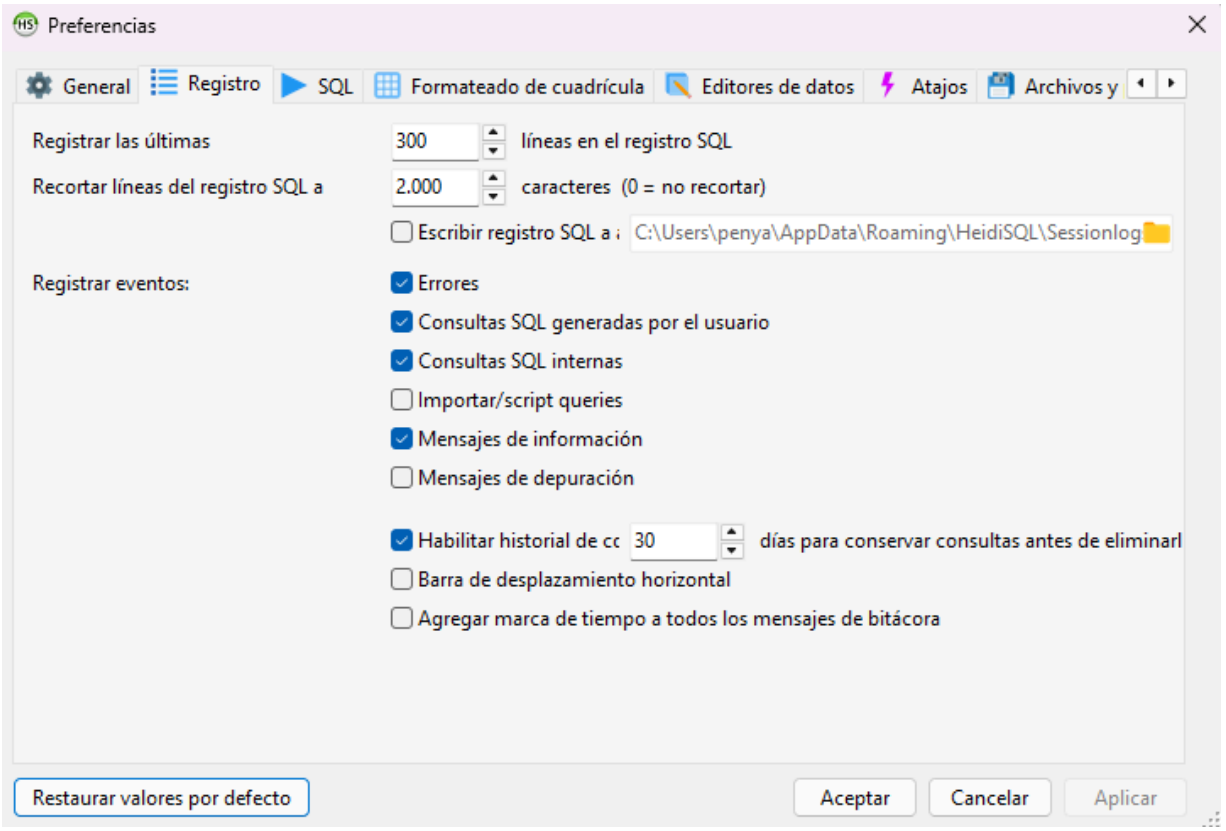
Ponemos la contraseña que hemos utilizado durante la instalación y le damos a Abrir



Ya estamos conectados a la base de datos



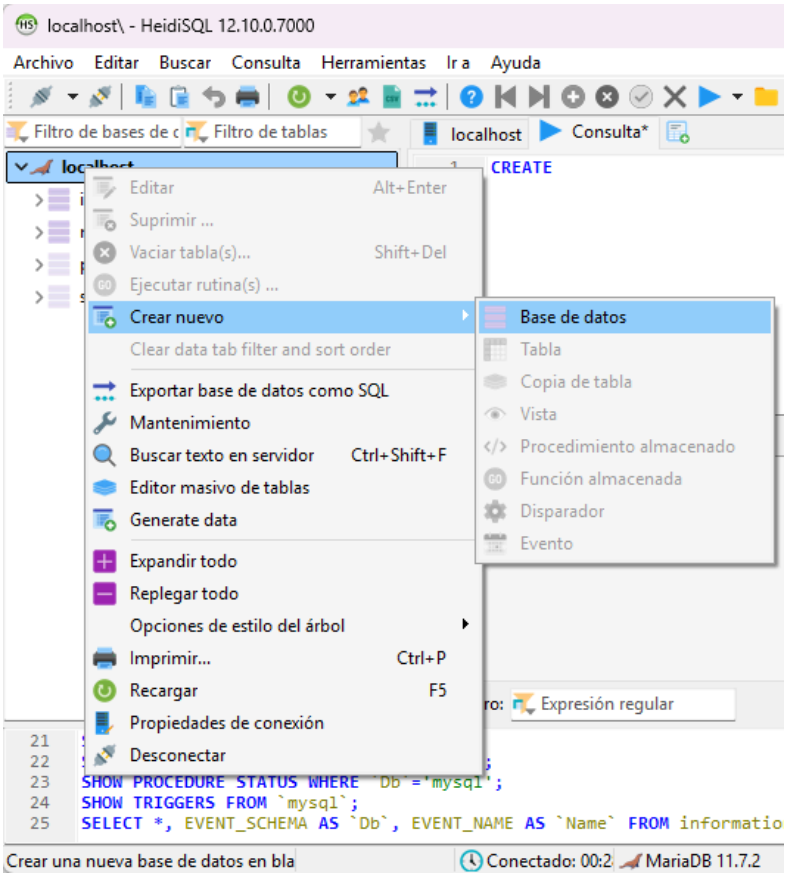
Comprobar las opciones del cliente yendo a Herramientas / Preferencias y seleccionando la pestaña Registro



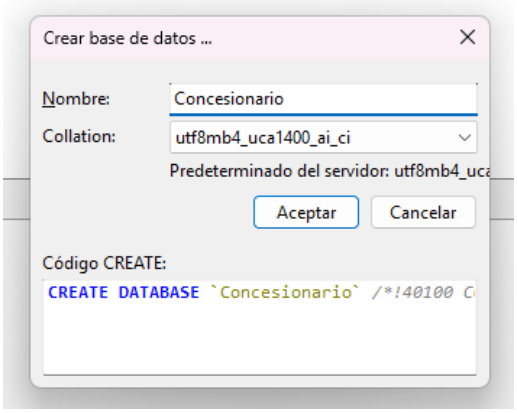
Crear una Base de datos desde HeidiSQL

Crear una base de datos desde HeidiSQL denominada Concesionario.

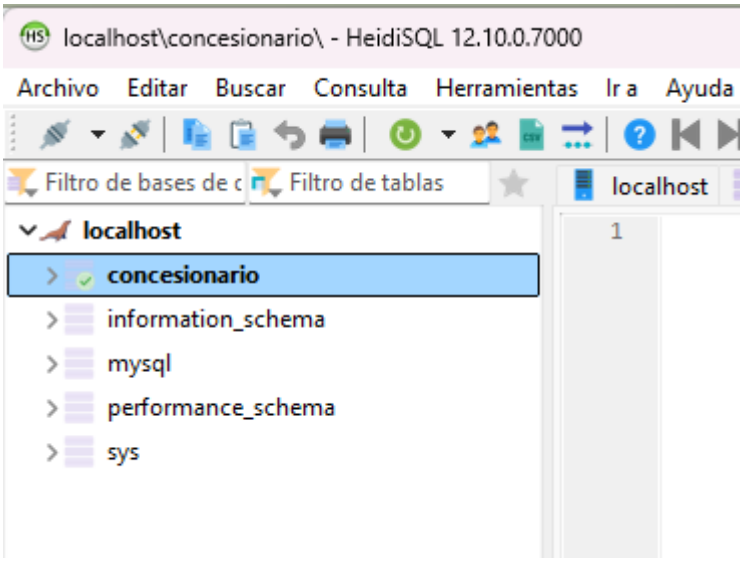
Clic derecho sobre la conexión / Crear nuevo / Base de datos



En la ventana emergente, escribir *Concesionario* como nombre de la base de datos y Aceptar



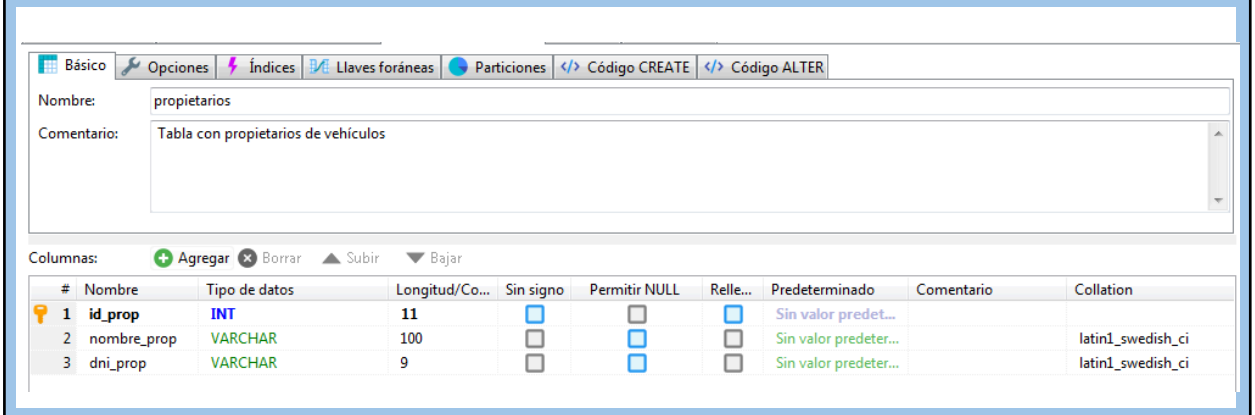
Se ha creado la Base de datos *Concesionario*



Dentro de esta base de datos crear dos tablas

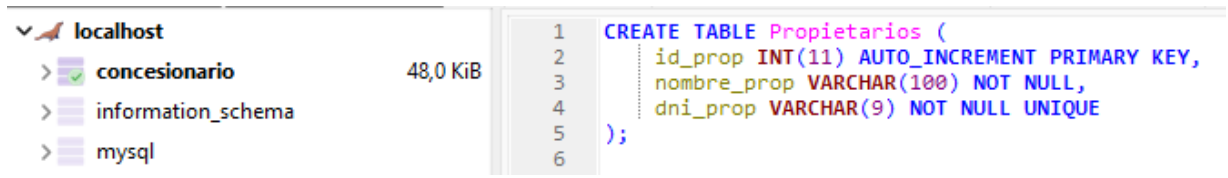
Crear un tabla *Propietarios* que almacenará los propietarios de los vehículos de la tabla anterior.

El **campo id_prop** de la tabla Propietarios puede tener un valor auto_incremento

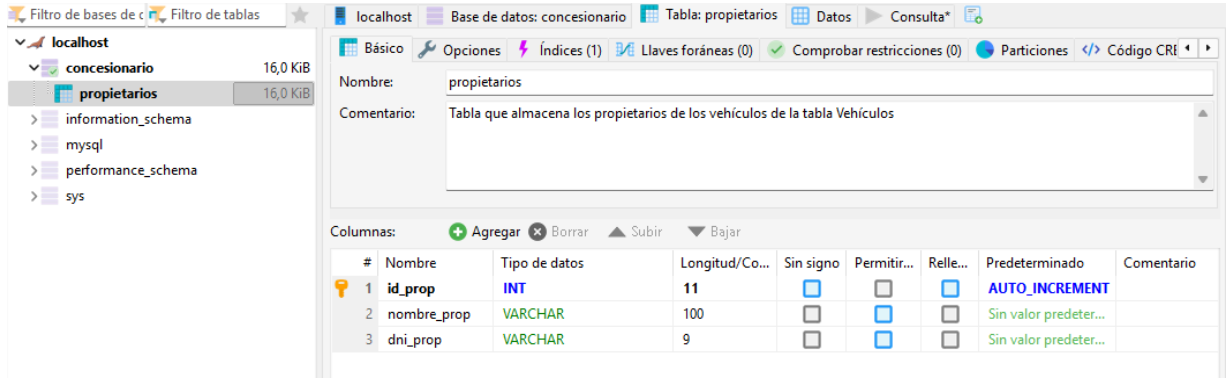


Seleccionar la base de datos del panel de la izquierda y seleccionar la pestaña Consulta del panel de la derecha. Escribir es código para crear la tabla *Propietarios*

```
CREATE TABLE Propietarios (
    id_prop INT(11) AUTO_INCREMENT PRIMARY KEY,
    nombre_prop VARCHAR(100) NOT NULL,
    dni_prop VARCHAR(9) NOT NULL UNIQUE
);
```

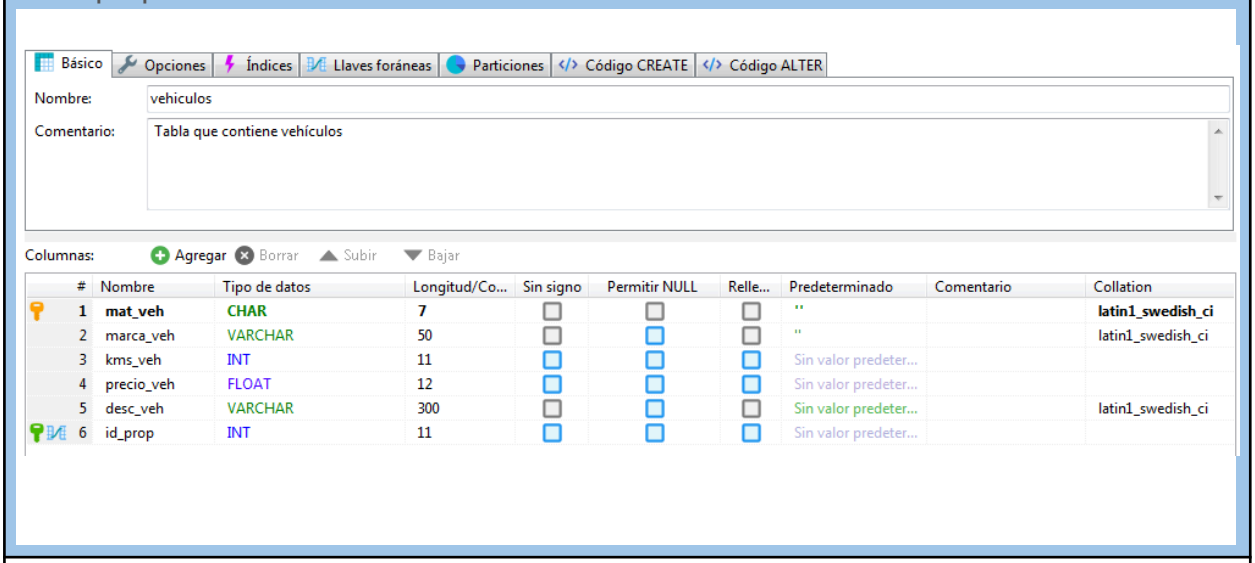


Ejecutar el código y comprobar que la tabla *Propietarios* se ha creado



Crear un tabla *Vehículos* que almacenará una serie de vehículos.

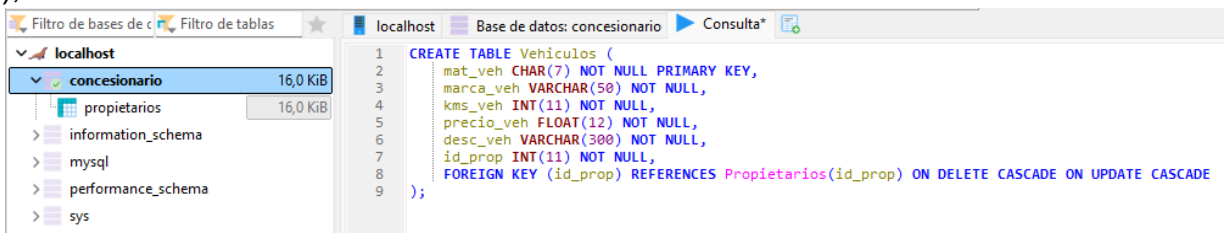
Observa como en la tabla vehículos, el campo **id_prop** es una clave externa hacia la tabla propietarios.



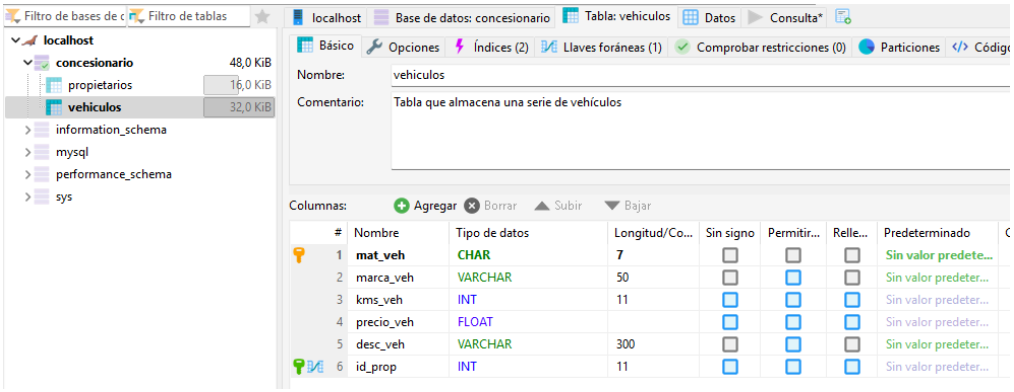
Seleccionar la base de datos del panel de la izquierda y seleccionar la pestaña Consulta del panel de la derecha. Escribir es código para crear la tabla.

```
CREATE TABLE Vehiculos (
   mat_veh CHAR(7) NOT NULL PRIMARY KEY,
   marca_veh VARCHAR(50) NOT NULL,
   kms_veh INT(11) NOT NULL,
```

```
    precio_veh FLOAT(12) NOT NULL,
    desc_veh VARCHAR(300) NOT NULL,
    id_prop INT(11) NOT NULL,
    FOREIGN KEY (id_prop) REFERENCES Propietarios(id_prop) ON DELETE
CASCADE ON UPDATE CASCADE
);
```



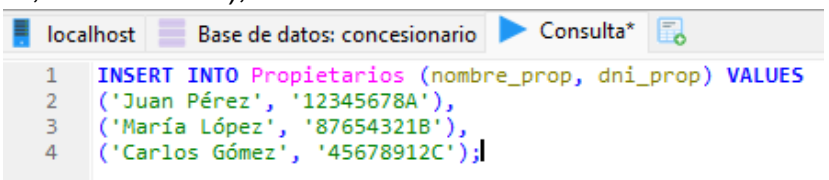
Ejecutar el código y comprobar que la tabla *Vehículos* se ha creado



Insertar datos en ambas tablas

Insertar datos en la tabla Propietarios

```
INSERT INTO Propietarios (nombre_prop, dni_prop) VALUES
('Juan Pérez', '12345678A'),
('María López', '87654321B'),
('Carlos Gómez', '45678912C');
```

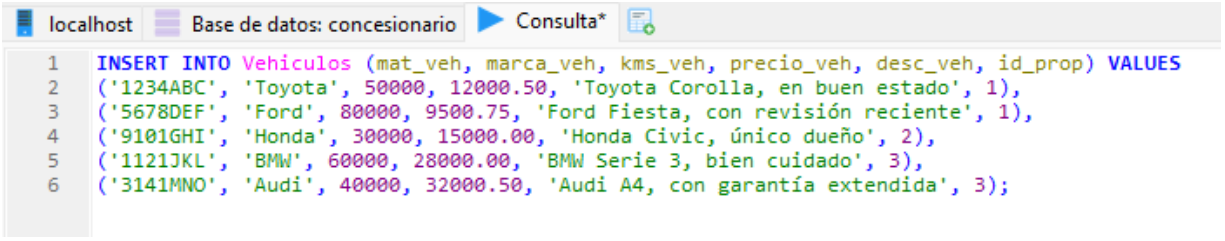


Ejecutar y comprobar

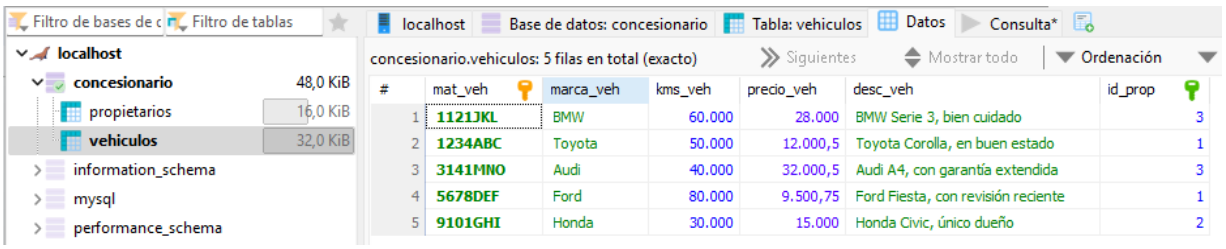


Insertar datos en la tabla Vehículos

INSERT INTO Vehiculos (mat_veh, marca_veh, kms_veh, precio_veh, desc_veh, id_prop) VALUES
('1234ABC', 'Toyota', 50000, 12000.50, 'Toyota Corolla, en buen estado', 1),
('5678DEF', 'Ford', 80000, 9500.75, 'Ford Fiesta, con revisión reciente', 1),
('9101GHI', 'Honda', 30000, 15000.00, 'Honda Civic, único dueño', 2),
('1121JKL', 'BMW', 60000, 28000.00, 'BMW Serie 3, bien cuidado', 3),
('3141MNO', 'Audi', 40000, 32000.50, 'Audi A4, con garantía extendida', 3);



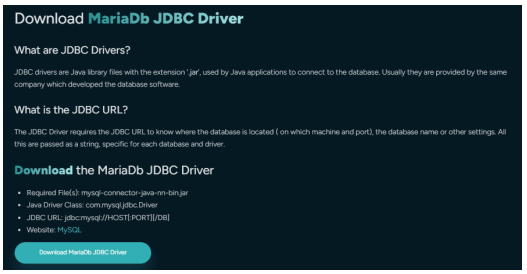
Ejecutar y comprobar



Instalar el conector de la base de datos

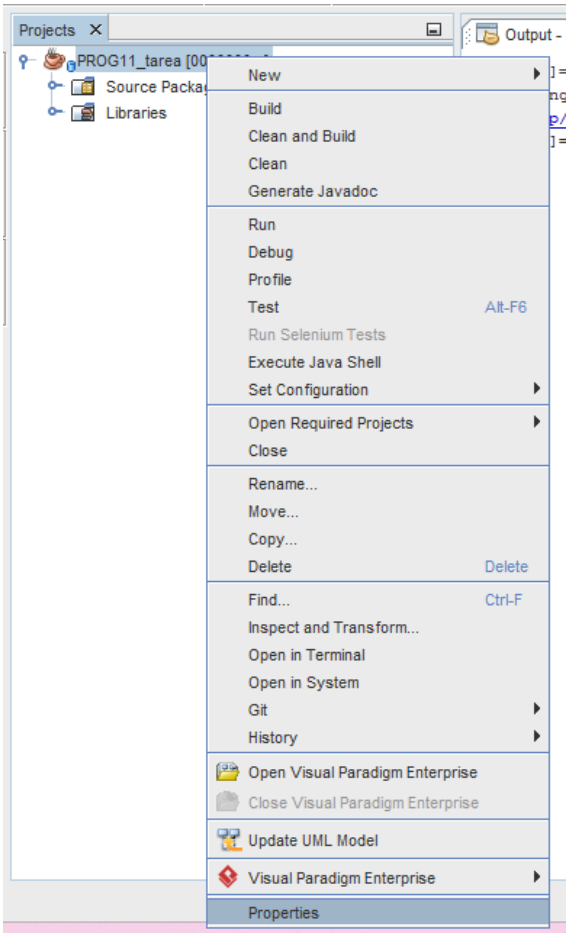
Instalar Driver MariaDB

Descargar el driver desde la web mariadb (se suministra en forma de fichero jar)

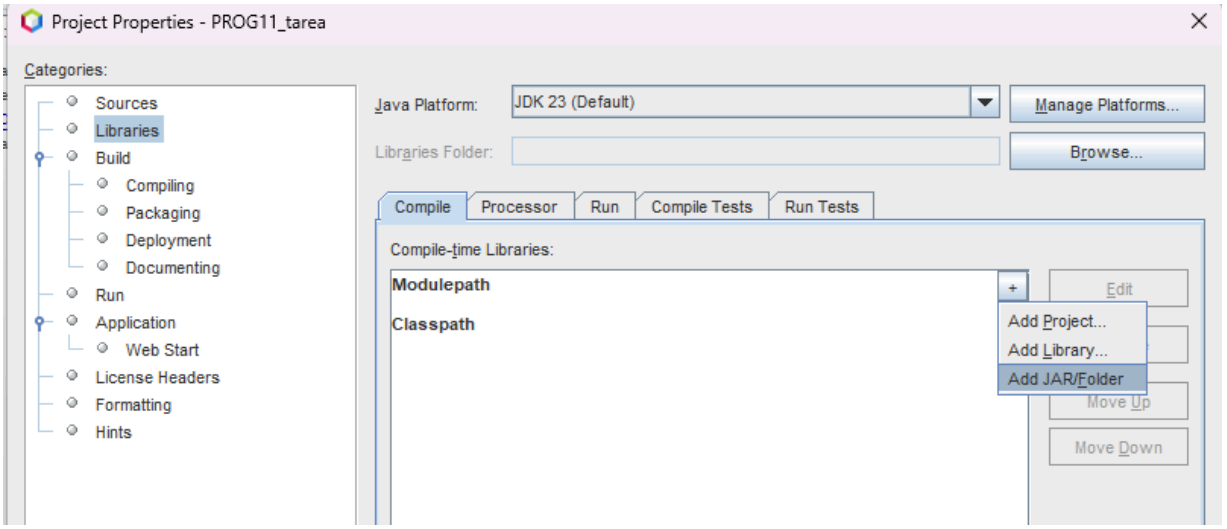


Añadir a todo proyecto que quiera establecer conexiones contra MariaDB el fichero .jar descargado

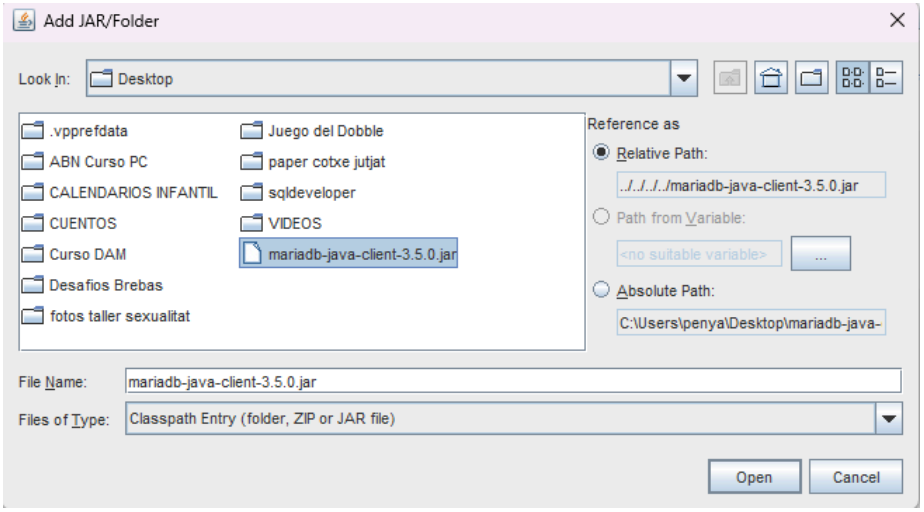
Crear un proyecto en NetBeans y acceder a las propiedades del proyecto (botón derecho sobre su nombre y pulsar en Properties)



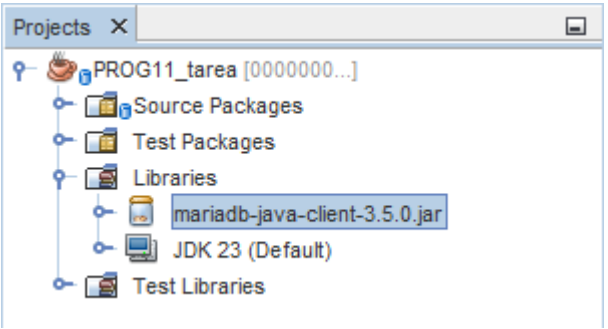
Seleccionaren el panel izquierdo la opción Libraries y agregar el driver en forma de fichero jar.



Seleccionar el archivo descargado



Comprobar que aparece en las librerías del proyecto



Código

Clase ConnectionDB

```
package com.prog11.bbdd;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.PreparedStatement;

/**
 * Clase encargada de gestionar la conexión con la base de datos MySQL
 * (MariaDB). Proporciona métodos para abrir y cerrar la conexión, así como
para
 * cerrar recursos asociados como PreparedStatement y ResultSet.
 */
public class ConnectionDB {

    /**
     * Establece una conexión con la base de datos MariaDB. Utiliza el driver
     * JDBC y conecta a la base de datos "concesionario" ubicada en localhost,
     * utilizando el usuario y contraseña "root".
     *
     * @return Connection Objeto de conexión a la base de datos o null si
ocurre
     * un error.
     */
    public static Connection openConnection() {
        Connection con = null;
        try {
            // Cargar el driver de MariaDB
            Class.forName("org.mariadb.jdbc.Driver");
            // Cadena de conexión a la base de datos
            String connectionUrl =
"jdbc:mariadb://localhost:3306/concesionario?user=root&password=root";
            // Obtener la conexión
            con = DriverManager.getConnection(connectionUrl);
            if (con != null) {
```

```
                System.out.println("Conexión establecida exitosamente.");
            }
        } catch (SQLException e) {
            System.out.println("SQL Exception: " + e.toString());
        } catch (ClassNotFoundException cE) {
            System.out.println("Excepcion: " + cE.toString());
        }
        return con;
    }

    /**
     * Cierra los recursos de base de datos: ResultSet, PreparedStatement y
     * Connection. Verifica si cada recurso no es null antes de intentar
     * cerrarlo, y captura posibles errores.
     *
     * @param con Objeto Connection a cerrar.
     * @param pstmt Objeto PreparedStatement a cerrar.
     * @param rs Objeto ResultSet a cerrar.
     */

    public static void closeConnection(Connection con, PreparedStatement pstmt,
ResultSet rs) {
        // Cerrar ResultSet si está abierto
        try {
            if (rs != null) {
                rs.close();
                System.out.println("ResultSet cerrado correctamente.");
            }
        } catch (SQLException e) {
            System.out.println("Error al cerrar el ResultSet: " +
e.getMessage());
        }

         // Cerrar PreparedStatement si está abierto
        try {
            if (pstmt != null) {
                pstmt.close();
                System.out.println("PreparedStatement cerrado correctamente.");
            }
        } catch (SQLException e) {
            System.out.println("Error al cerrar el PreparedStatement: " +
```

```
e.getMessage());
        }

        // Cerrar la conexión si está abierta
        try {
            if (con != null) {
                con.close();
                System.out.println("Conexión cerrada correctamente.");
            }
        } catch (SQLException e) {
            System.out.println("Error al cerrar la conexión: " +
e.getMessage());
        }
    }
}
```

Clase PropietariosDAO

```
package com.prog11.bbdd;

import static com.prog11.bbdd.VehiculosDAO.eliminarVehiculo;
import java.util.ArrayList;
import java.sql.*;

/**
 * Clase encargada de gestionar las operaciones CRUD relacionadas con los
 * propietarios de vehículos en la base de datos. Incluye métodos para
insertar,
 * eliminar y recuperar propietarios y sus vehículos, además de validar el NIF.
 */
public class PropietariosDAO {

    /**
     * Inserta un nuevo propietario en la base de datos si el DNI proporcionado
     * es válido.
     *
     * @param con Conexión activa a la base de datos.
     * @param nombre Nombre del propietario.
     * @param dni DNI del propietario.
     * @return int 0 si la inserción fue exitosa, -1 si ocurrió un error o el
     * DNI no es válido.
```

```
     */
    public static int insertarPropietario(Connection con, String nombre, String
dni) {
        int resultado = -1; // Valor por defecto en caso de error

        // Validar el NIF antes de intentar la inserción
        if (!validarNIF(dni)) {
            System.out.println("El DNI proporcionado no es válido.");
            return -1;
        }

        // Consulta SQL para insertar un nuevo propietario
        String queryInsert = "INSERT INTO propietarios (nombre_prop, dni_prop)
VALUES (?, ?)";

        try (PreparedStatement pstmt = con.prepareStatement(queryInsert)) {
            pstmt.setString(1, nombre);
            pstmt.setString(2, dni);
            pstmt.executeUpdate();
            System.out.println("Propietario insertado correctamente.");
            resultado = 0; // Inserción exitosa
        } catch (SQLException e) {
            System.out.println("Error al insertar propietario: " +
e.getMessage());
            System.out.println("Código de error SQL: " + e.getErrorCode());
        }

        return resultado; // 0 si se insertó correctamente, -1 si hubo error
    }

    /**
     * Elimina un propietario y todos sus vehículos asociados de la base de
     * datos.
     *
     * @param con Conexión activa a la base de datos.
     * @param dni DNI del propietario a eliminar.
     * @return int Número de propietarios eliminados (0 si no se encontró o
hubo
     * error).
     */
    public static int eliminarPropietario(Connection con, String dni) {
```

```
        int numReg = 0; // Contador de registros eliminados

        try {
            // Recuperar vehículos asociados al propietario
            ArrayList<String> vehiculos = recuperarVehiculosDePropietario(con,
dni);

            // Si hay vehículos, eliminarlos primero
            if (vehiculos != null && !vehiculos.isEmpty()) {
                System.out.println("El propietario con DNI " + dni + " tiene
los siguientes vehículos:");
                for (String vehiculo : vehiculos) {
                    System.out.println(vehiculo);
                }

                // Eliminar los vehículos del propietario
                for (String vehiculo : vehiculos) {
                    // Extraemos la matrícula del vehículo para eliminarlo
                    String matricula =
vehiculo.split(",")[0].split(":")[1].trim(); // Extraemos la matrícula
                    eliminarVehiculo(con, matricula);  // Eliminamos el
vehículo por matrícula
                }
            }

            // Luego eliminamos al propietario
            String queryDelete = "DELETE FROM propietarios WHERE dni_prop = ?";
            try (PreparedStatement pstmtDelete =
con.prepareStatement(queryDelete)) {
                pstmtDelete.setString(1, dni);
                numReg = pstmtDelete.executeUpdate();
            }

        } catch (SQLException e) {
            System.out.println("Error al eliminar propietario: " +
e.getMessage());
            System.out.println("Código de error SQL: " + e.getErrorCode());
            return 0; // Si hubo un error, devolvemos 0
        }

        // Imprimir mensaje según el resultado
```

```
        if (numReg > 0) {
            System.out.println("El propietario con DNI " + dni + " y sus
vehículos han sido eliminados.");
        } else {
            System.out.println("No se encontró un propietario con el DNI " +
dni);
        }

        return numReg; // Devolvemos el número de registros eliminados
    }

    /**
     * Recupera todos los vehículos asociados a un propietario a partir de su
     * DNI.
     *
     * @param con Conexión activa a la base de datos.
     * @param dni DNI del propietario.
     * @return ArrayList<String> Lista con la información de los vehículos del
     * propietario o null si no tiene vehículos o hubo un error.
     */
    public static ArrayList<String> recuperarVehiculosDePropietario(Connection
con, String dni) {
        ArrayList<String> vehiculos = new ArrayList<>();

        // Consulta SQL para recuperar los vehículos asociados al propietario
        String query = "SELECT mat_veh, marca_veh, kms_veh, precio_veh FROM
vehiculos WHERE id_prop = (SELECT id_prop FROM propietarios WHERE dni_prop =
?)";

        try (PreparedStatement stmt = con.prepareStatement(query)) {
            stmt.setString(1, dni); // Establecer el DNI del propietario

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

            // Verificar si hay vehículos asociados al propietario
            while (rs.next()) {
                String matricula = rs.getString("mat_veh");
                String marca = rs.getString("marca_veh");
                int kms = rs.getInt("kms_veh");
                float precio = rs.getFloat("precio_veh");
```

```
                // Añadir los datos del vehículo a la lista
                String vehiculo = "Matrícula: " + matricula + ", Marca: " +
marca + ", Kms: " + kms + ", Precio: " + precio;
                vehiculos.add(vehiculo);
            }

            // Si no se encuentran vehículos, devolver null
            if (vehiculos.isEmpty()) {
                return null;
            }
        } catch (SQLException e) {
            System.out.println("Error al recuperar vehículos:");
            System.out.println("Código de error: " + e.getErrorCode());
            System.out.println("Mensaje: " + e.getMessage());
            return null; // Retornar null si hubo un error en la consulta
        }

        return vehiculos; // Devolver la lista de vehículos
    }

    /**
     * Valida si un NIF es correcto, verificando la letra y el número.
     *
     * @param nif El NIF a validar.
     * @return boolean True si el NIF es válido, false si es inválido.
     */
    private static boolean validarNIF(String nif) {
        boolean valido = true;

        char letra_calculada;
        char letra_leida;
        int dni_leido;

        if (nif == null) {
            valido = false;
        } else if (nif.length() < 8 || nif.length() > 9) {
            valido = false;
        } else {
            letra_leida = extraerLetraNIF(nif);  // Extraemos la letra
            dni_leido = extraerNumeroNIF(nif);  // Extraemos el número
```

```
            letra_calculada = calcularLetraNIF(dni_leido);  // Calculamos la
letra

            if (letra_leida == letra_calculada) {
                valido = true; // Si la letra coincide, el NIF es válido
            } else {
                valido = false; // La letra no coincide, el NIF es inválido
            }
        }

        return valido; // Devolver la validez del NIF
    }

    /**
     * Extrae la letra del NIF (la última posición del string).
     *
     * @param nif El NIF del cual extraer la letra.
     * @return char La letra del NIF.
     */
    private static char extraerLetraNIF(String nif) {
        return nif.charAt(nif.length() - 1);  // La letra es el último carácter
    }

    /**
     * Extrae el número del NIF (sin la letra).
     *
     * @param nif El NIF del cual extraer el número.
     * @return int El número del NIF.
     */
    private static int extraerNumeroNIF(String nif) {
        return Integer.parseInt(nif.substring(0, nif.length() - 1));  // Todo
menos la última letra
    }

    /**
     * Calcula la letra del NIF en base al número utilizando un algoritmo
     * específico.
     *
     * @param dni El número del NIF.
     * @return char La letra correspondiente al número.
     */
```

```
    private static char calcularLetraNIF(int dni) {
        char[] letras = {'T', 'R', 'W', 'A', 'G', 'M', 'Y', 'F', 'P', 'D', 'X',
'B', 'N', 'J', 'Z', 'S', 'Q', 'V', 'H', 'L', 'C', 'K', 'E'};
        return letras[dni % 23];  // La letra se calcula con el módulo 23
    }
}
```

Clase VehiculosDAO

```
package com.prog11.bbdd;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

/**
 * Clase encargada de gestionar las operaciones CRUD (Crear, Leer, Actualizar,
Eliminar)
 * relacionadas con los vehículos en la base de datos. Incluye métodos para
insertar,
 * eliminar, actualizar vehículos y recuperar información sobre ellos, ya sea
por matrícula, marca o propietario.
 */
public class VehiculosDAO {

    /**
     * Inserta un nuevo vehículo en la base de datos.
     *
     * @param con Conexión activa a la base de datos.
     * @param matricula Matrícula del vehículo.
     * @param marca Marca del vehículo.
     * @param kms Kilómetros recorridos por el vehículo.
     * @param precio Precio del vehículo.
     * @param descripcion Descripción del vehículo.
     * @param idProp ID del propietario del vehículo.
     * @return int 0 si la inserción fue exitosa, -1 si hubo un error.
     */
    public static int insertarVehiculo(Connection con, String matricula, String
marca, int kms, float precio, String descripcion, int idProp) {
```

```
        int resultado = -1;  // Por defecto, si hay error, se devuelve -1

        String queryInsert = "INSERT INTO vehiculos (mat_veh, marca_veh,
kms_veh, precio_veh, desc_veh, id_prop) VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement stmtInsert = con.prepareStatement(queryInsert))
{
            stmtInsert.setString(1, matricula);     // Establecer matrícula
            stmtInsert.setString(2, marca);         // Establecer marca
            stmtInsert.setInt(3, kms);              // Establecer kilómetros
            stmtInsert.setFloat(4, precio);         // Establecer precio
            stmtInsert.setString(5, descripcion);   // Establecer descripción
            stmtInsert.setInt(6, idProp);           // Establecer id_prop

            int filasAfectadas = stmtInsert.executeUpdate();
            if (filasAfectadas > 0) {
                // Si la inserción fue exitosa
                System.out.println("El vehículo con matrícula " + matricula + "
ha sido insertado correctamente.");
                return 0;  // Devolvemos 0 si la inserción fue exitosa
            } else {
                // En caso de que no se haya insertado ninguna fila
                System.out.println("No se pudo insertar el vehículo.");
            }
        } catch (SQLException e) {
            System.out.println("Error al insertar vehículo: " +
e.getMessage());
        }

        return -1; // Retorna -1 si hubo error
    }

    /**
     * Elimina un vehículo de la base de datos usando su matrícula.
     *
     * @param con Conexión activa a la base de datos.
     * @param matricula Matrícula del vehículo a eliminar.
     * @return int 0 si la eliminación fue exitosa, -1 si hubo un error.
     */
    public static int eliminarVehiculo(Connection con, String matricula) {
        int resultado = -1;  // Por defecto, si hay error, se devuelve -1
```

```
        // SQL para eliminar el vehículo con la matrícula proporcionada
        String queryDelete = "DELETE FROM vehiculos WHERE mat_veh = ?";

        try (PreparedStatement stmtDelete = con.prepareStatement(queryDelete))
{
            stmtDelete.setString(1, matricula);  // Establecer la matrícula del
vehículo a eliminar

            int filasAfectadas = stmtDelete.executeUpdate();
            if (filasAfectadas > 0) {
                // Si la eliminación fue exitosa
                System.out.println("El vehículo con matrícula " + matricula + "
ha sido eliminado correctamente.");
                return 0;  // Devolvemos 0 si la eliminación fue exitosa
            } else {
                // Si no se ha eliminado ningún registro (probablemente porque
el vehículo no existe)
                System.out.println("No se encontró el vehículo con matrícula "
+ matricula + ".");
                return -1;  // Si no se encuentra el vehículo, devolvemos -1
            }
        } catch (SQLException e) {
            // Si ocurre algún error (por ejemplo, si la base de datos lanza
una excepción por no encontrar el vehículo)
            System.out.println("Error al eliminar el vehículo: " +
e.getMessage());
            return -1;  // Si hubo un error, devolvemos -1
        }
    }

    /**
     * Actualiza el propietario de un vehículo mediante su matrícula.
     *
     * @param con Conexión activa a la base de datos.
     * @param matricula Matrícula del vehículo.
     * @param nuevoIdProp Nuevo ID del propietario.
     * @return int 0 si la actualización fue exitosa, -1 si hubo un error o el
vehículo no existe.
     */
    public static int actualizarPropietarioVehiculo(Connection con, String
matricula, int nuevoIdProp) {
```

```
        int resultado = -1;  // Por defecto, si hay error, se devuelve -1

        // SQL para actualizar el propietario del vehículo con la matrícula
proporcionada
        String queryUpdate = "UPDATE vehiculos SET id_prop = ? WHERE mat_veh =
?";

        try (PreparedStatement stmtUpdate = con.prepareStatement(queryUpdate))
{
            // Establecemos los parámetros de la consulta de actualización
            stmtUpdate.setInt(1, nuevoIdProp);  // Establecemos el nuevo ID del
propietario
            stmtUpdate.setString(2, matricula); // Establecemos la matrícula
del vehículo

            // Ejecutamos la consulta UPDATE
            int filasAfectadas = stmtUpdate.executeUpdate();
            if (filasAfectadas > 0) {
                // Si la actualización fue exitosa
                System.out.println("El propietario del vehículo con matrícula "
+ matricula + " ha sido actualizado correctamente.");
                resultado = 0;  // La actualización fue exitosa
            } else {
                // Si no se ha actualizado ningún registro (es decir, el
vehículo no existe)
                System.out.println("El vehículo con matrícula " + matricula + "
no existe.");
                resultado = -1;  // Si no existe el vehículo, devolvemos -1
            }
        } catch (SQLException e) {
            // Si ocurre algún error al ejecutar la consulta
            System.out.println("Error al actualizar el propietario del
vehículo: " + e.getMessage());
            resultado = -1;  // Si hubo un error, devolvemos -1
        }

        return resultado;  // Retorna 0 si la operación fue exitosa, -1 si hubo
error
    }

    /**
```

```
     * Recupera todos los vehículos y sus propietarios.
     *
     * @param con Conexión activa a la base de datos.
     * @return ArrayList<String> Lista de vehículos con sus detalles.
     */
    public static ArrayList<String> recuperarVehiculos(Connection con) {
        ArrayList<String> listaVehiculosRecuperar = new ArrayList<>();

        // Consulta SQL para obtener los vehículos y sus propietarios
        String query = "SELECT v.mat_veh, v.marca_veh, v.kms_veh, v.precio_veh,
v.desc_veh, p.nombre_prop "
                + "FROM vehiculos v "
                + "JOIN propietarios p ON v.id_prop = p.id_prop";

        try (PreparedStatement stmt = con.prepareStatement(query)) {
            ResultSet rs = stmt.executeQuery();

            // Recorremos el ResultSet y formateamos los datos de los vehículos
            while (rs.next()) {
                String matricula = rs.getString("mat_veh");
                String marca = rs.getString("marca_veh");
                int kms = rs.getInt("kms_veh");
                float precio = rs.getFloat("precio_veh");
                String descripcion = rs.getString("desc_veh");
                String propietario = rs.getString("nombre_prop");

                // Formateamos la información del vehículo
                String vehiculoInfo = "Matrícula: " + matricula
                        + ", Marca: " + marca
                        + ", Kms: " + kms
                        + ", Precio: " + precio
                        + ", Descripción: " + descripcion
                        + ", Propietario: " + propietario;

                // Añadimos el vehículo a la lista
                listaVehiculosRecuperar.add(vehiculoInfo);
            }

            // Mostramos los vehículos recuperados
            if (!listaVehiculosRecuperar.isEmpty()) {
                System.out.println("Lista de vehículos:");
```

```
                for (String vehiculo : listaVehiculosRecuperar) {
                    System.out.println(vehiculo);
                }
            } else {
                System.out.println("No se encontraron vehículos.");
            }

        } catch (SQLException e) {
            System.out.println("Error al recuperar los vehículos: " +
e.getMessage());
        }

        return listaVehiculosRecuperar;  // Devolvemos la lista de vehículos
(vacía si no se encuentran)
    }

    /**
     * Recupera los vehículos de una marca específica y sus propietarios.
     *
     * @param con Conexión activa a la base de datos.
     * @param marca Marca de los vehículos a recuperar.
     * @return ArrayList<String> Lista de vehículos de la marca especificada.
     */
    public static ArrayList<String> recuperarVehiculosPorMarca(Connection con,
String marca) {
        ArrayList<String> listaVehiculosMarca = new ArrayList<>();

        // Consulta SQL para obtener los vehículos de la marca proporcionada y
sus propietarios
        String query = "SELECT v.mat_veh, v.marca_veh, v.kms_veh, v.precio_veh,
v.desc_veh, p.nombre_prop "
                + "FROM vehiculos v "
                + "JOIN propietarios p ON v.id_prop = p.id_prop "
                + "WHERE v.marca_veh = ?";  // Filtramos por la marca

        try (PreparedStatement stmt = con.prepareStatement(query)) {
            stmt.setString(1, marca);  // Establecemos la marca en la consulta

            ResultSet rs = stmt.executeQuery();

            // Recorremos el ResultSet y formateamos los datos de los vehículos
```

```
            while (rs.next()) {
                String matricula = rs.getString("mat_veh");
                String marcaVeh = rs.getString("marca_veh");
                int kms = rs.getInt("kms_veh");
                float precio = rs.getFloat("precio_veh");
                String descripcion = rs.getString("desc_veh");
                String propietario = rs.getString("nombre_prop");

                // Formateamos la información del vehículo y propietario
                String vehiculoInfo = "Matrícula: " + matricula
                        + ", Marca: " + marcaVeh
                        + ", Kms: " + kms
                        + ", Precio: " + precio
                        + ", Descripción: " + descripcion
                        + ", Propietario: " + propietario;

                // Añadimos el vehículo a la lista
                listaVehiculosMarca.add(vehiculoInfo);
            }
            // Mostrar la lista de vehículos
            if (listaVehiculosMarca.isEmpty()) {
                System.out.println("No se encontraron vehículos de la marca " +
marca + ".");
            } else {
                System.out.println("Vehículos de la marca " + marca + ":");
                for (String vehiculo : listaVehiculosMarca) {
                    System.out.println(vehiculo);
                }
            }

        } catch (SQLException e) {
            System.out.println("Error al recuperar los vehículos por marca: " +
e.getMessage());
        }

        return listaVehiculosMarca;  // Devolvemos la lista de vehículos (vacía
si no se encuentran)
    }

    /**
     * Recupera los vehículos de un propietario específico utilizando su DNI.
```

```
     *
     * @param con Conexión activa a la base de datos.
     * @param dni DNI del propietario de los vehículos a recuperar.
     * @return ArrayList<String> Lista de vehículos pertenecientes al
propietario con el DNI especificado.
     */
    public static ArrayList<String> recuperarVehiculosPorPropietario(Connection
con, String dni) {
        ArrayList<String> listaVehiculos = new ArrayList<>();

        // Consulta SQL para obtener los vehículos asociados al propietario
cuyo DNI coincide
        String query = "SELECT v.mat_veh, v.marca_veh, v.kms_veh, v.precio_veh,
v.desc_veh, p.nombre_prop "
                + "FROM vehiculos v "
                + "JOIN propietarios p ON v.id_prop = p.id_prop "
                + "WHERE p.dni_prop = ?";  // Filtramos por el DNI del
propietario

        try (PreparedStatement stmt = con.prepareStatement(query)) {
            stmt.setString(1, dni);  // Usamos el DNI para filtrar el
propietario
            ResultSet rs = stmt.executeQuery();

            // Recorremos el ResultSet y formateamos los datos de cada vehículo
            while (rs.next()) {
                String matricula = rs.getString("mat_veh");
                String marca = rs.getString("marca_veh");
                int kms = rs.getInt("kms_veh");
                float precio = rs.getFloat("precio_veh");
                String descripcion = rs.getString("desc_veh");
                String propietario = rs.getString("nombre_prop");

                String vehiculoInfo = "Matrícula: " + matricula
                        + ", Marca: " + marca
                        + ", Kms: " + kms
                        + ", Precio: " + precio
                        + ", Descripción: " + (descripcion != null ?
descripcion : "No disponible")
                        + ", Propietario: " + (propietario != null ?
propietario : "No disponible");
```

```
                listaVehiculos.add(vehiculoInfo);
            }

            // Si la lista está vacía, se imprime un mensaje, de lo contrario
se muestran los vehículos
            if (listaVehiculos.isEmpty()) {
                System.out.println("El propietario con DNI " + dni + " no tiene
vehículos.");
            } else {
                System.out.println("Vehículos del propietario con DNI " + dni +
":");
                for (String vehiculo : listaVehiculos) {
                    System.out.println(vehiculo);
                }
            }
        } catch (SQLException e) {
            System.out.println("Error al recuperar los vehículos del
propietario: " + e.getMessage());
        }

        return listaVehiculos;  // Devuelve la lista (vacía si no se
encontraron vehículos)
    }
}
```

Clase Prog11_Principal

```
package com.prog11.princ;

import com.prog11.bbdd.ConnectionDB;
import com.prog11.bbdd.PropietariosDAO;
import com.prog11.bbdd.VehiculosDAO;
import static com.prog11.bbdd.VehiculosDAO.recuperarVehiculosPorPropietario;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

/**
```

```
 * Clase principal del programa que se encarga de gestionar vehículos y
 * propietarios. La clase realiza operaciones sobre la base de datos, tales
como
 * insertar vehículos y propietarios, actualizar propietarios de vehículos,
 * listar vehículos, eliminar vehículos y propietarios, y recuperar vehículos
 * asociados a una marca o propietario.
 */
public class Prog11_Principal {

    /**
     * Método principal que ejecuta una serie de operaciones sobre la base de
     * datos. Realiza la inserción de vehículos y propietarios, actualización
de
     * propietario de vehículo, eliminación de vehículos y propietarios, y
     * consultas para listar vehículos por diferentes criterios.
     *
     * @param args Argumentos de la línea de comandos (no utilizados en este
     * caso).
     */
    public static void main(String[] args) {

        // Abrir la conexión a la base de datos
        Connection con = ConnectionDB.openConnection();
        PreparedStatement pstmt = null;
        ResultSet rs = null;

        // Insertar varios vehículos
        // Inserta tres vehículos con distintos atributos
        int insertarvehiculo1 = VehiculosDAO.insertarVehiculo(con, "1234XYZ",
"Mercedes", 25000, 22000.00f, "Mercedes Benz Clase A, en excelente estado", 2);
        int insertarvehiculo2 = VehiculosDAO.insertarVehiculo(con, "5678ABC",
"BMW", 35000, 25000.00f, "BMW Serie 3, casi nuevo", 3);
        int insertarvehiculo3 = VehiculosDAO.insertarVehiculo(con, "9876DEF",
"Audi", 40000, 30000.00f, "Audi A4, en muy buen estado", 3);

        // Insertar varios propietarios
        // Inserta tres propietarios con sus respectivos datos
        int insertarpropietario1 = PropietariosDAO.insertarPropietario(con,
"Francisco Rebollo", "06599725J");
        int insertarpropietario2 = PropietariosDAO.insertarPropietario(con,
"Carmen García", "63961340L");
```

```
        int insertarpropietario3 = PropietariosDAO.insertarPropietario(con,
"Manolo Trucha", "48005699F");

        // Listar todos los vehículos
        // Recupera todos los vehículos de la base de datos
        ArrayList<String> listaVehiculos =
VehiculosDAO.recuperarVehiculos(con);

        // Actualizar el propietario de un vehículo
        // Actualiza el propietario de un vehículo con matrícula "1234ABC"
        String matriculaVehiculo = "1234ABC";  // Matrícula del vehículo
        int nuevoIdPropietario = 1;  // Nuevo ID del propietario (debe ser un
ID válido)
        int resultadoActualizacion =
VehiculosDAO.actualizarPropietarioVehiculo(con, matriculaVehiculo,
nuevoIdPropietario);

        // Listar todos los vehículos después de la actualización
        ArrayList<String> listaVehiculos2 =
VehiculosDAO.recuperarVehiculos(con);

        // Eliminar un vehículo que exista
        // Elimina un vehículo con la matrícula "1234XYZ"
        String matriculaEliminar = "1234XYZ"; // Matrícula del vehículo a
eliminar
        int resultadoEliminacion = VehiculosDAO.eliminarVehiculo(con,
matriculaEliminar);

        // Eliminar un vehículo que no exista
        // Intenta eliminar un vehículo con matrícula "4321XYZ" (no existe)
        String matriculaEliminar2 = "4321XYZ"; // Matrícula del vehículo a
eliminar
        int resultadoEliminacion2 = VehiculosDAO.eliminarVehiculo(con,
matriculaEliminar2);

         // Listar todos los vehículos después de la eliminación
        ArrayList<String> listaVehiculos3 =
VehiculosDAO.recuperarVehiculos(con);

        // Listar los vehículos de una marca
        // Recupera todos los vehículos de la marca "Audi"
```

```
        String marcaBuscar = "Audi";
        ArrayList<String> listaVehiculosMarca =
VehiculosDAO.recuperarVehiculosPorMarca(con, marcaBuscar);

        // Listar todos los vehículos de un propietario
        // Recupera los vehículos del propietario con DNI "87654321B"
        String dni = "87654321B";
        ArrayList<String> vehiculos = recuperarVehiculosPorPropietario(con,
dni);

        // Eliminar un propietario con vehículos asociados
        // Elimina al propietario con DNI "48005699F" y sus vehículos asociados
        String dniPropietario = "48005699F";  // DNI del propietario a eliminar
        int registrosEliminados = PropietariosDAO.eliminarPropietario(con,
dniPropietario);

        // Eliminar un propietario sin vehículos asociados
        // Elimina al propietario con DNI "63961340L" que no tiene vehículos
asociados
        String dniPropietario2 = "63961340L";  // DNI del propietario a
eliminar
        int registrosEliminados2 = PropietariosDAO.eliminarPropietario(con,
dniPropietario2);

        // Cerrar la conexión a la base de datos
        ConnectionDB.closeConnection(con, pstmt, rs);

    }
}
```