



# ESCUELA POLITÉCNICA NACIONAL

**Fundamentos de Bases de Datos**

**Proyecto II Bimestre**

**Tema:** Administrador para alquiler de Vehículos

**Integrante:**

- Luis Enrique Pérez Señalín



## **Índice:**

1. Introducción
2. Descripción del proyecto
3. Diseño de la base de datos
4. Desarrollo del sistema
  - 4.1 Conexión a la base de datos.
  - 4.2 Flujo del programa
  - 4.3 Navegación por el sistema - Tabs
  - 4.4 Funciones de la ventana Registros
  - 4.5 Funciones de la ventana Actualizar datos
  - 4.6 Funciones de la ventana Insertar datos
  - 4.7 Funciones de la ventana Eliminar datos
5. Prueba del sistema.
6. Conclusiones

## 1. Introducción

Se necesita un sistema que permita manejar los registros de un local de alquiler de coches, para esto se utilizó una base de datos de SQL Server, un sistema de Windows Form en C# usando Visual Studio, extra se utilizó Git para control de versiones con el fin de mantener copias de seguridad y un seguimiento de los cambios realizados en el proyecto.

## 2. Descripción del proyecto

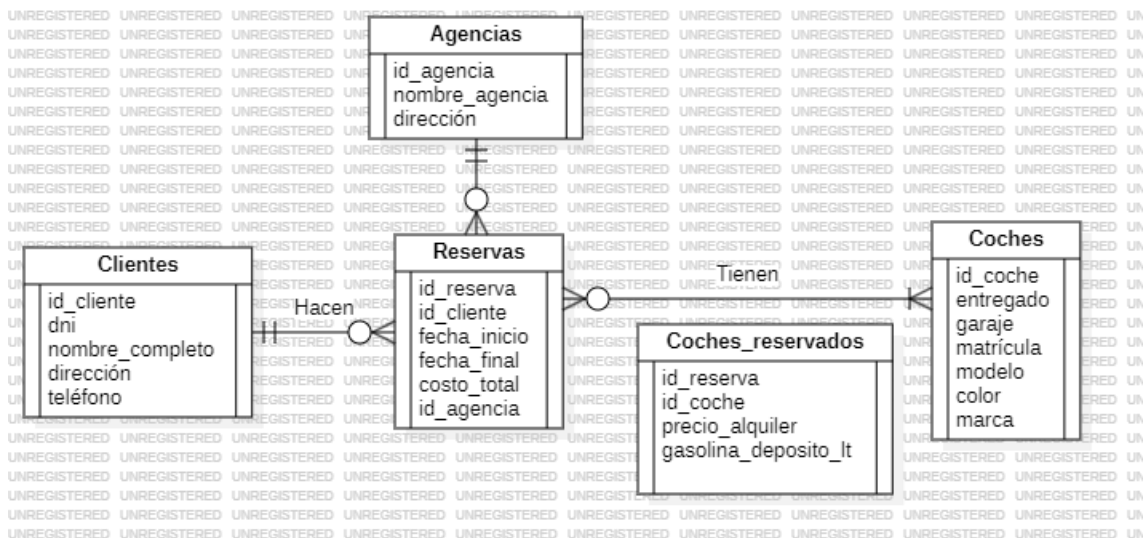
El sistema tiene como fin gestionar las reservas de coches, permitiendo ver los datos actuales en la base de datos y modificarlos.

### Funcionalidad:

- Inserción de registros.
- Actualización de registros.
- Eliminación de registros.
- Consulta de registros.

## 3. Diseño de la base de datos

Se utilizó el siguiente esquema de entidad relación en la base de datos:



En este esquema de entidad relación tenemos lo mismo que en el proyecto del primer bimestre, con 4 tablas en total, la relación entre las tablas y sus distintos atributos.

La base de datos está alojada en SQL server.

## 4. Desarrollo del sistema

### 4.1 Conexión a la base de datos.

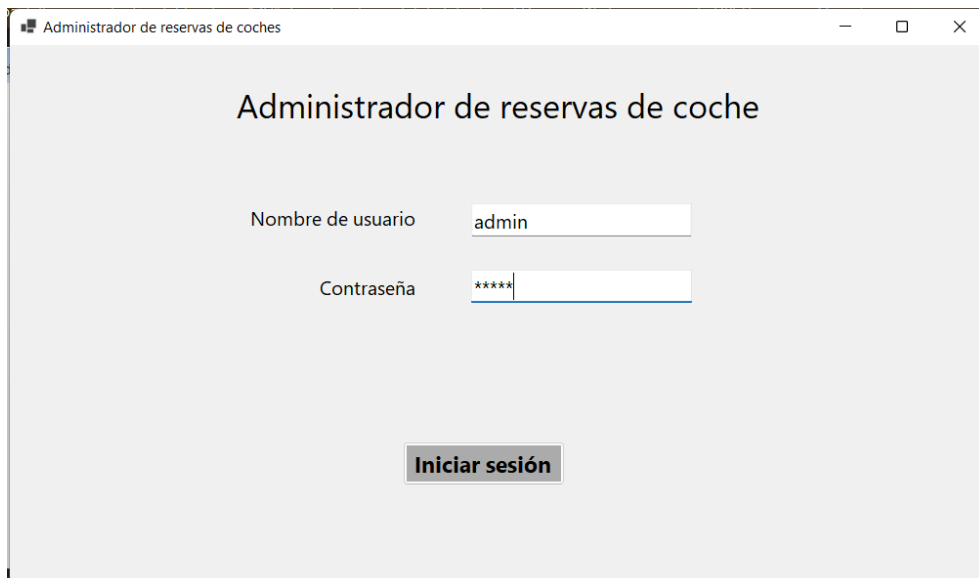
La base de datos se trabaja en Local, en el DBMS Sql Server.

```
1 // Establecer la conexión a la base de datos
2 private string connectionString = @"
3     Data Source=localhost\MSSQLSERVER03;
4     Initial Catalog=db_reservas_coches;
5     Integrated Security=True";
6 using (SqlConnection connection = new SqlConnection(connectionString))
7 {
8     try
9     {
10         connection.Open();
11         label1.Text = "Bienvenido - Estás conectado";
12         // Aquí puedes realizar otras acciones como cargar datos en controles del formulario
13     }
14     catch (Exception ex)
15     {
16         MessageBox.Show($"Error al conectar a la base de datos: {ex.Message}");
17     }
18 }
```

Esta se realiza dentro de la función **public MainForm()** para que al iniciar el Formulario Main, automáticamente se conecte al servidor, le damos los valores del Data Source, el nombre de la base de datos y que permita la seguridad integrada.

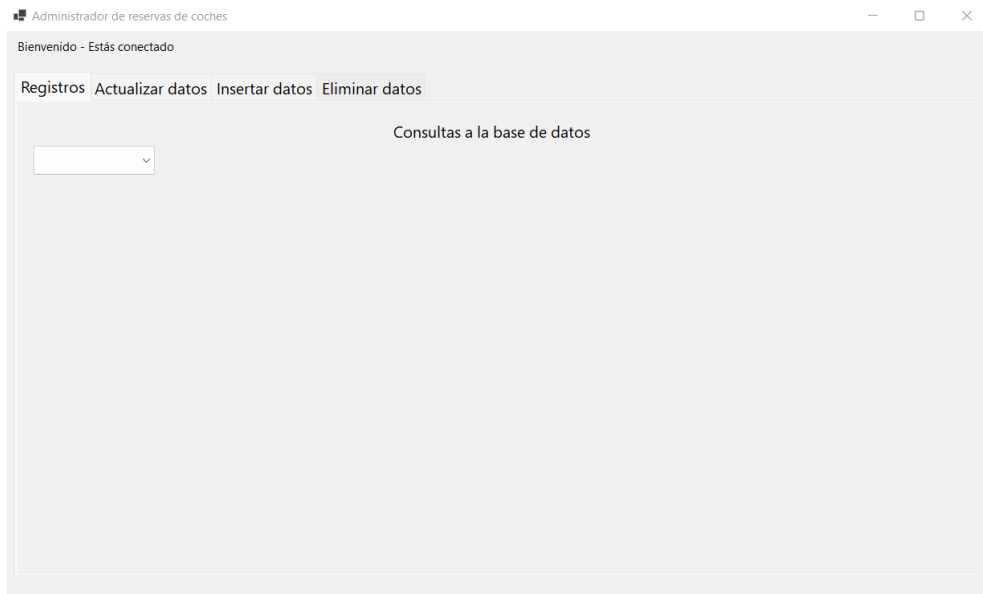
### 4.2 Flujo del programa

El programa inicia con una pantalla de Login, lo que permite añadir seguridad, este es un Form llamado LoginForm.cs.





Luego el botón oculta este Form y abre el MainForm.cs.

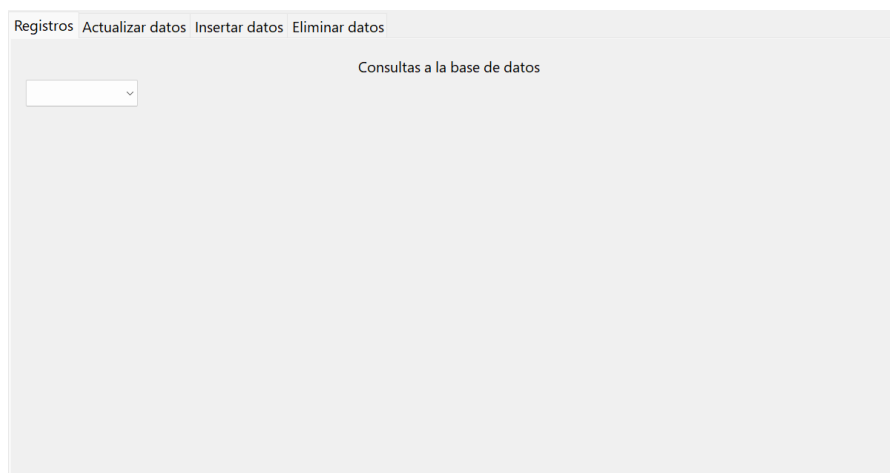


Este sistema se maneja con “Tabs”, los cuales permiten moverse entre “Registros”, “Actualizar datos”, “Insertar datos” y “Eliminar datos”.

### 4.3 Navegación por el sistema - Tabs

El sistema permite moverse entre distintas vistas por medio de los Tabs, aquí tenemos los ejemplos de cada Tabs.

- Tab - Registros.



- Tab - Actualizar datos.

Registros Actualizar datos Insertar datos Eliminar datos

Actualizar Datos

Agencias Clientes Coches Reservas

Agencias

Antiguo nombre de la agencia

Nuevo nombre de la agencia

Nueva dirección

Actualizar agencia

- Tab – Insertar datos.

Registros Actualizar datos Insertar datos Eliminar datos

Insertar Datos

Agencias Clientes Coches Reservas

Agencias

Nombre de la agencia

Dirección

Crear Agencia

- Tab Eliminar datos.

Registros Actualizar datos Insertar datos Eliminar datos

Eliminar Datos

Agencias Clientes Coches Reservas

Agencias

Nombre de la agencia

Eliminar agencia

*Cada uno de estos Tabs tienen otros Tabs para moverse entre las distintas tablas de la base de datos, esto nos permite un flujo claro de como el programa se debe utilizar*

#### 4.4 Funciones de la ventana Registros

La ventana Registro tiene como función principal permitir la visualización de los datos registrados en la base de datos de todas las tablas.

Utilizando un Dropbox que tiene como opciones: Agencias, Clientes, Coches y Reservas, se realiza la consulta a la base de datos y el resultado se muestra en un **DataGridView**.

Para realizar una consulta a la base de datos y ver el resultado basta con seleccionar una opción en el Dropbox.

##### 1. Consultar las Agencias.

Consultas a la base de datos

Agencias ▾

	Nombre de la Agencia	Dirección
▶	Agencia Central	Calle Principal 123
	Agencia Norte	Avenida Norte 456
	Agencia Sur	Boulevard del Sur 789
	Agencia Este	Plaza del Este 101
	Agencia Oeste	Avenida Oeste 651

##### 2. Consultar los clientes.

Consultas a la base de datos

Clientes ▾

	Nombre completo	Dirección	DNI	Número de telefono	Avalador	DNI del Avalador
▶	Juan Perez	Calle Falsa 123	12345678	987654321	Juan Perez	12345678
	Ana Gómez	Avenida Siempreviva 742	87654321	123456789	Ana Gómez	87654321
	Carlos Ruiz	Boulevard de los Sueños Rotos 101	11223344	456123789	Juan Perez	12345678
	Laura Fuentes	Plaza de la Luna 202	44332211	789123456	Carlos Ruiz	11223344
	Enrique Perez	Quito, Floresta	7706546	989570200	Ana Gómez	87654321
	Karina Arichavala	Quito, Floresta	98769875	99576540	Enrique Perez	7706546



### 3. Consultar los coches.

Consultas a la base de datos

Coches ▾

	ID	Garaje	Matricula	Modelo	Color	Marca	entregado
▶	1	Garaje A	ABC1234D	Model S	Rojo	Tesla	<input checked="" type="checkbox"/>
	2	Garaje B	XYZ5678F	Mustang	Azul	Ford	<input type="checkbox"/>
	3	Garaje C	LMN9101G	Civic	Negro	Honda	<input checked="" type="checkbox"/>
	4	Garaje D	JKL1122H	Corolla	Blanco	Toyota	<input type="checkbox"/>
	6	Garaje Z	EFD1234S	Model R	Blanco	Tesla	<input checked="" type="checkbox"/>

### 4. Consultar las reservas

Consultas a la base de datos

Reservas ▾

	ID	Nombre del cliente	DNI	Fecha inicio	Fecha de fin	Costo total	Agencia
▶	1	Juan Perez	12345678	1/6/2023 10:00	10/6/2023 10:00	500,00	Agencia Central
	2	Ana Gómez	87654321	5/6/2023 12:00	15/6/2023 12:00	750,00	Agencia Norte
	3	Carlos Ruiz	11223344	10/6/2023 9:00	20/6/2023 9:00	300,00	Agencia Sur
	4	Laura Fuentes	44332211	15/6/2023 14:00	25/6/2023 14:00	450,00	Agencia Este
	6	Enrique Perez	7706546	10/8/2023 9:00	13/8/2023 9:00	1800,00	Agencia Central



## Ejecutar consultas.

Para ejecutar las consultas utilizamos la siguiente función:

```
1 private DataTable query_sql(string query)
2 {
3     using (SqlConnection connection = new SqlConnection(connectionString))
4     {
5         try
6         {
7             connection.Open();
8
9             SqlDataAdapter adapter = new SqlDataAdapter(query, connection);
10            DataTable dataTable = new DataTable();
11            adapter.Fill(dataTable);
12
13            return dataTable;
14        }
15        catch (Exception ex)
16        {
17            MessageBox.Show($"Error al cargar los datos: {ex.Message}");
18            return null;
19        }
20    }
21 }
```

Vamos a extraer el valor del dropBox y en base a eso guardaremos la query en una variable que luego ejecutaremos y por último mostraremos en el DataGridView.

```
1 private void comboBox_table_selected_SelectedIndexChanged(object sender, EventArgs e)
2 {
3     string table = comboBox_table_selected.Text;
4     string query = "";
5     if (table == "Clientes")
6     {
7
8     }
9     if (table == "Reservas")
10    {
11
12    }
13    if (table == "Agencias")
14    {
15
16    }
17    if (table == "Coches")
18    {
19
20    }
21    DataTable query_table = query_sql(query);
22    // Asigna los datos al DataGridView
23    dataGridView1.DataSource = query_table;
24 }
25
```

Esta es la función que nos da el DropBox para ejecutar, cuando se selecciona una opción en el Dropbox

Estás son las consultas para cada una de las opciones:

- Clientes:

```
1 SELECT c1.nombre_completo AS 'Nombre completo',
2 c1.dni AS DNI, c1.direccion AS Dirección,
3 c1.telefono AS 'Número de telefono',
4 c2.nombre_completo AS Avalador, c2.dni AS 'DNI del Avalador'
5 FROM clientes c1
6 LEFT JOIN clientes c2 ON c1.avalador = c2.id_cliente;";
```

- Reservas:

```
1 SELECT c1.nombre_completo AS 'Nombre completo',
2 c1.dni AS DNI, c1.direccion AS Dirección,
3 c1.telefono AS 'Número de telefono',
4 c2.nombre_completo AS Avalador, c2.dni AS 'DNI del Avalador'
5 FROM clientes c1
6 LEFT JOIN clientes c2 ON c1.avalador = c2.id_cliente;
```

- Agencias:

```
1 SELECT agencias.nombre_agencia AS 'Nombre de la Agencia',
2 agencias.direccion AS Dirección
3 FROM agencias;
```

- Reservas:

```
1 SELECT coches.id_coche AS ID, coches.garaje AS Garaje,
2 coches.matricula AS Matricula, coches.modelo AS Modelo,
3 coches.color AS Color, coches.marca AS Marca, coches.entregado
4 FROM coches;
```

## 4.5 Funciones de la ventana Actualizar datos

En esta ventana tenemos otro Tab de 4 opciones, una para cada tabla, cada una de estas tiene una interfaz distinta donde pondremos distintos datos para realizar la modificación.

La función `update_query` es utilizada para ejecutar las consultas SQL para actualizar los datos. Recibe los parámetros “query” y “message”.

```
1 public void update_query(string query, string message)
2 {
3     try
4     {
5         query_sql(query);
6         MessageBox.Show($"{message} actualizada con éxito");
7     }
8     catch (Exception ex)
9     {
10        MessageBox.Show($"Error al actualizar los datos: {ex.Message}");
11    }
12 }
```

## 1. Actualizar Agencia.

Registros Actualizar datos Insertar datos Eliminar datos

Actualizar Datos

Agencias Clientes Coches Reservas

Agencias

Antiguo nombre de la agencia

Nuevo nombre de la agencia

Nueva dirección

Actualizar agencia

En actualizar datos tenemos los campos vistos en la pantalla, usamos el campo “antiguo nombre de la agencia” para poder identificar la agencia que se va a actualizar.

Este es el código para la actualización:

```
1 private void button_update_agencia_Click(object sender, EventArgs e)
2 {
3     string agencia_old_name = update_agencias_old_name.Text;
4     string agencia_new_name = update_agencias_new_name.Text;
5     string agencia_new_address = update_agencias_new_address.Text;
6
7     string query = $"
8         UPDATE agencias
9         SET nombre_agencia = '{agencia_new_name}', direccion = '{agencia_new_address}'
10        WHERE nombre_agencia = '{agencia_old_name}';
11    ";
12    update_query(query, "Agencia");
13    update_agencias_old_name.Text = "";
14    update_agencias_new_name.Text = "";
15    update_agencias_new_address.Text = "";
16 }
```

Los nombres de los text\_box son lo más descriptivos posibles, tienen primero el nombre del tab ej.: “update”, “insert”, “delete” y el nombre de la tabla “agencias”, “clientes”, “reservas”, “vehículos” y por último el campo que hace referencia.

## 2. Actualizar clientes.

Registros Actualizar datos Insertar datos Eliminar datos

Actualizar Datos

Agencias Clientes Coches Reservas

Clientes

DNI

Nuevo nombre completo

Nueva dirección

Nuevo telefono

Nuevo DNI Avalador

Actualizar Cliente

Para actualizar los datos se usa el DNI del cliente, porque los nombres se pueden repetir, luego se añaden los campos nuevos, no se puede cambiar el DNI porque su caso sería extraño, una vez llenados los datos se ejecuta la función **button\_update\_client\_Click**.

```
1 private void button_update_client_Click(object sender, EventArgs e)
2 {
3     string dni = update_client_dni.Text;
4     string name = update_client_new_name.Text;
5     string address = update_client_new_address.Text;
6     float phone = float.Parse(update_client_new_phone.Text);
7     string avalador_dni = update_client_new_dni_avalador.Text;
8
9     string query = $"
10     UPDATE clientes
11     SET
12         nombre_completo = '{name}',
13         direccion = '{address}',
14         telefono = {phone},
15         avalador = (SELECT id_cliente FROM clientes WHERE dni = '{avalador_dni}')
16     WHERE dni = '{dni}';
17 ";
18     update_query(query, "cliente");
19
20     update_client_dni.Text = "";
21     update_client_new_name.Text = "";
22     update_client_new_address.Text = "";
23     update_client_new_phone.Text = "";
24     update_client_new_dni_avalador.Text = "";
25 }
```

Se utiliza la misma función de **update\_query** para ejecutar la sentencia SQL y para obtener el id del "avalador" realizamos una subconsulta utilizando el dni del avalador, de esta manera no se necesita conocer el id del avalador, sino solo el DNI.

### 3. Actualizar coches.

Registros Actualizar datos Insertar datos Eliminar datos

Actualizar Datos

Agencias Clientes Coches Reservas

Coches

Id Coche

Nueva matrícula

Nuevo Garaje

Nuevo modelo

Nuevo color

Nueva marca

Actualizar Vehículo

Para actualizar la tabla de coches se utiliza el id de coche para que se pueda modificar la matrícula del vehículo, después de llenar todos los campos se ejecuta la función **button\_update\_car\_Click**.

```
1 private void button_update_car_Click(object sender, EventArgs e)
2 {
3     string id_coche = update_car_id.Text;
4     string garaje = update_car_new_garage.Text;
5     string matricula = update_car_new_matricula.Text;
6     string model = update_car_new_model.Text;
7     string color = update_car_new_color.Text;
8     string marca = update_car_new_marca.Text;
9
10    string query = $"
11        UPDATE coches
12        SET
13            garaje = '{garaje}',
14            matricula = '{matricula}',
15            modelo = '{model}',
16            color = '{color}',
17            marca = '{marca}'
18        WHERE id_coche = {id_coche};
19    ";
20    update_query(query, "cliente");
21    update_car_id.Text = "";
22    update_car_new_garage.Text = "";
23    update_car_new_matricula.Text = "";
24    update_car_new_model.Text = "";
25    update_car_new_color.Text = "";
26    update_car_new_marca.Text = "";
27 }
```

#### 4. Actualizar reservas

Registros
Actualizar datos
Insertar datos
Eliminar datos

Actualizar Datos

Agencias
Clientes
Coches
Reservas

Reservas

ID reserva

DNI cliente

Fecha inicio
yy/mm/dd hh:mm:ss

Fecha final
yy/mm/dd hh:mm:ss

Costo total \$

Nombre de agencia

Actualizar Reserva

Para actualizar las reservas se utiliza como identificador el id, el cuál es usuario puede obtener de la parte de registros, con el fin de poder modificar todos los campos de la reserva que se desea actualizar, se utiliza la función **button\_update\_reservas\_Click**.

```

1 private void button_update_reservas_Click(object sender, EventArgs e)
2 {
3     string id_reserva = update_reservas_id.Text;
4     string dni_cliente = update_reservas_new_dni_client.Text;
5     string begin_date = update_reservas_new_begin_date.Text;
6     string final_date = update_reservas_new_end_date.Text;
7     string cost = update_reservas_new_cost.Text;
8     string agencia_name = update_reservas_new_agencia_name.Text;
9
10    string query = $"
11        UPDATE reservas
12        SET
13            id_cliente = (SELECT id_cliente FROM clientes WHERE dni = '{dni_cliente}'),
14            fecha_inicio = '{begin_date}',
15            fecha_final = '{final_date}',
16            costo_total = '{cost}',
17            id_agencia = (SELECT id_agencia FROM agencias WHERE nombre_agencia = '{agencia_name}')
18        WHERE id_reserva = {id_reserva};
19    ";
20    update_query(query, "Reserva");
21
22    update_reservas_id.Text = "";
23    update_reservas_new_dni_client.Text = "";
24    update_reservas_new_begin_date.Text = "";
25    update_reservas_new_end_date.Text = "";
26    update_reservas_new_cost.Text = "";
27    update_reservas_new_agencia_name.Text = "";
28 }

```

#### 4.6 Funciones de la ventana Insertar datos

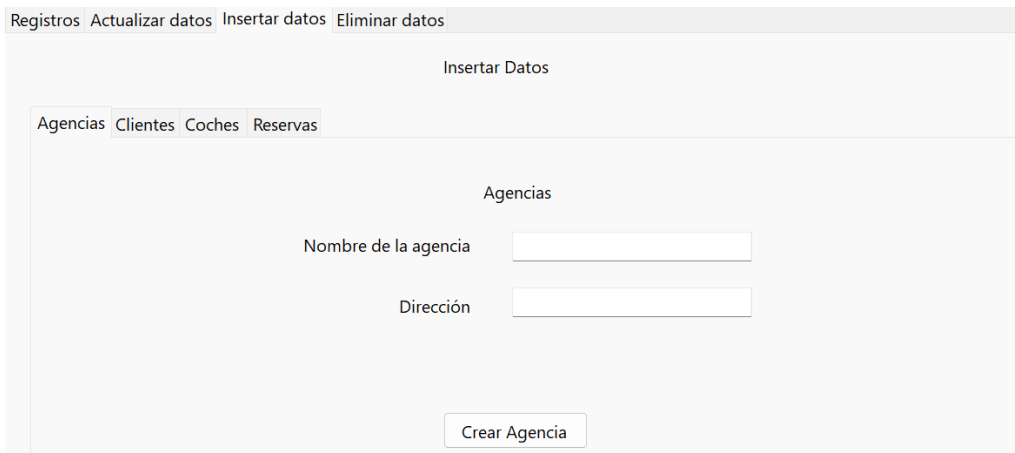
Esta ventana tiene como función insertar nuevos datos dentro de las tablas correspondientes en la base de datos. En esta ventana contamos con las mismas 4 pestañas que tiene “Actualizar datos”, para poder seleccionar que dato vamos a crear.

Para esto tenemos la función **insert\_query**.

```
1 public void insert_query(string query, string message)
2 {
3     try
4     {
5         query_sql(query);
6         MessageBox.Show($"{message} creada con éxito");
7     }
8     catch (Exception ex)
9     {
10        MessageBox.Show($"Error al crear los datos: {ex.Message}");
11    }
12 }
```

A continuación, vamos a revisar cada una de las pestañas con las funciones que tienen:

##### 1. Insertar Agencia



Esta pestaña solo tiene los campos de nombre de la agencia y dirección que se utilizan para insertar una nueva agencia, se utiliza la función **insert\_agencia\_Click**.

```
1 private void insert_agencia_Click(object sender, EventArgs e)
2 {
3     string agencia_name = insert_agencia_name.Text;
4     string agencia_address = insert_agencia_address.Text;
5
6     string query = @"
7         INSERT INTO agencias (nombre_agencia, direccion)
8         VALUES ('{agencia_name}', '{agencia_address}')";
9     insert_query(query, "Agencia");
10    insert_agencia_name.Text = "";
11    insert_agencia_address.Text = "";
12 }
```

## 2. Insertar Clientes.

Registros Actualizar datos Insertar datos Eliminar datos

Insertar Datos

Agencias Clientes Coches Reservas

Clientes

DNI

Nombre completo

dirección

telefono

DNI Avalador

Guardar Cliente

Utilizamos todos los campos necesarios para clientes, el único campo que se utiliza para una **subconsulta** es “DNI Avalador” por el cuál se va a obtener el id del avalador y se insertará el cliente con dicho valor en el campo id avalador, utilizamos la función **button\_insert\_cliente\_Click**.

```
1 private void button_insert_cliente_Click(object sender, EventArgs e)
2 {
3     string dni = insert_cliente_dni.Text;
4     string name = insert_cliente_name.Text;
5     string address = insert_cliente_address.Text;
6     float phone = float.Parse(insert_cliente_phone.Text);
7     string avalador_dni = insert_cliente_avalador.Text;
8
9     string query = @"
10         INSERT INTO clientes (dni, nombre_completo, direccion, telefono, avalado, avalador)
11         VALUES
12         ('{dni}','{name}','{address}','{phone}','{1}',
13         (SELECT id_cliente FROM clientes WHERE dni = '{avalador_dni}'))";
14     insert_query(query, "cliente");
15
16     insert_cliente_dni.Text = "";
17     insert_cliente_name.Text = "";
18     insert_cliente_address.Text = "";
19     insert_cliente_phone.Text = "";
20     insert_cliente_avalador.Text = "";
21 }
```

En el campo avalador se necesita el id del avalador, por lo que se utiliza el dni del avalador para obtener el valor de su id en la línea 13 para que no se necesite mostrar los id de los clientes, sino solo los dni.



### 3. Insertar Coches.

Registros Actualizar datos Insertar datos Eliminar datos

Insertar Datos

Agencias Clientes Coches Reservas

Coches

Matrícula

Garaje

Modelo

Color

Marca

Guardar Vehículo

Utilizamos todos los valores necesarios para crear un nuevo coche en la tabla de Coches, en este caso no utilizamos las subconsultas, y el valor de disponible está en 1 automáticamente, utilizamos la función **button2\_Click\_1** debido a una mala generación automática de la función.

```
1 private void button2_Click_1(object sender, EventArgs e)
2 {
3     string matricula = insert_coches_matricula.Text;
4     string garaje = insert_coches_garaje.Text;
5     string modelo = insert_coches_modelo.Text;
6     string color = insert_coches_color.Text;
7     string marca = insert_coches_marca.Text;
8
9     string query = @"
10         INSERT INTO coches (garaje, matricula, modelo, color, marca, entregado)
11         VALUES ('{garaje}','{matricula}','{modelo}','{color}','{marca}','{1}')";
12
13     insert_query(query, "Coche");
14     insert_coches_matricula.Text = "";
15     insert_coches_garaje.Text = "";
16     insert_coches_modelo.Text = "";
17     insert_coches_color.Text = "";
18     insert_coches_marca.Text = "";
19 }
```

La función utiliza todos los valores dados y pone en la sentencia SQL, solo el valor de “entregado” se pone en 1 por defecto.

#### 4. Insertar Reservas.

Registros Actualizar datos Insertar datos Eliminar datos

Insertar Datos

Agencias Clientes Coches Reservas

Reservas

DNI cliente

Fecha inicio  yy/mm/dd hh:mm:ss

Fecha final  yy/mm/dd hh:mm:ss

Costo total \$

Nombre de agencia

Crear Reserva

Utilizamos todos los campos para insertar los datos excepto los campos de “DNI cliente” y “Nombre de agencia”, los cuales usamos para hacer una **subconsulta** para obtener el id del cliente y de la agencia y crear la reserva utilizando estos valores, para crear esta reserva utilizamos la función **button\_insert\_reserva\_Click**.

```
1 private void button_insert_reserva_Click(object sender, EventArgs e)
2 {
3     string dni_cliente = insert_reserva_dni_cliente.Text;
4     string begin_date = insert_reserva_begin_date.Text;
5     string final_date = insert_reserva_final_date.Text;
6     string cost = insert_reserva_costo.Text;
7     string agencia_name = insert_reserva_agencia_name.Text;
8
9     string query = $"
10         INSERT INTO reservas (id_cliente, fecha_inicio, fecha_final, costo_total, id_agencia)
11         VALUES (
12             (SELECT id_cliente FROM clientes WHERE dni = '{dni_cliente}'),
13             '{begin_date}',
14             '{final_date}',
15             '{cost}',
16             (SELECT id_agencia FROM agencias WHERE nombre_agencia = '{agencia_name}')
17         );
18     ";
19     insert_query(query, "Reserva");
20
21     insert_reserva_dni_cliente.Text = "";
22     insert_reserva_begin_date.Text = "";
23     insert_reserva_final_date.Text = "";
24     insert_reserva_costo.Text = "";
25     insert_reserva_agencia_name.Text = "";
26 }
```

Podemos ver las **subconsultas** usando “dni\_cliente” y “agencia\_name” en las líneas 12 y 16 para obtener los id de cliente y agencia e incluirlos en los datos para crear una reserva.

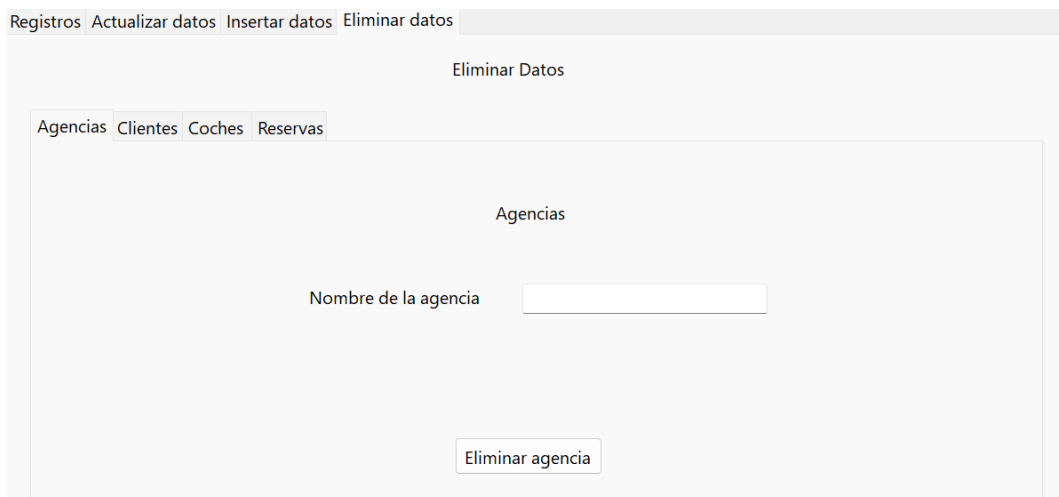
#### 4.7 Funciones de la ventana Eliminar datos

Para eliminar datos en la base de datos utilizamos la pestaña “Eliminar datos”, todas las funciones que se ejecutan dentro de esta ventana utilizan la función **delete\_query**.

```
1 public void delete_query(string query, string message)
2 {
3     try
4     {
5         query_sql(query);
6         MessageBox.Show($"{message} eliminada con éxito");
7     }
8     catch (Exception ex)
9     {
10        MessageBox.Show($"Error al eliminar los datos: {ex.Message}");
11    }
12 }
```

A continuación, veremos las pestañas de cada tabla:

##### 1. Eliminar Agencias.



Para eliminar una agencia solo se necesita el campo “nombre de la agencia” para que al utilizar la función **button\_delete\_agencia\_Click** podamos eliminar la agencia deseada.

```
1 private void button_delete_agencia_Click(object sender, EventArgs e)
2 {
3     string agencia_name = delete_agencia_name.Text;
4
5     string query = $"DELETE FROM agencias WHERE nombre_agencia = '{agencia_name}'";
6     delete_query(query, "Agencia");
7 }
```

La función obtiene el nombre de la agencia de la interfaz y ejecuta la sentencia SQL para eliminar la agencia de la base de datos.

## 2. Eliminar Clientes.

Registros Actualizar datos Insertar datos Eliminar datos

Eliminar Datos

Agencias Clientes Coches Reservas

Clientes

DNI

Nombre Completo

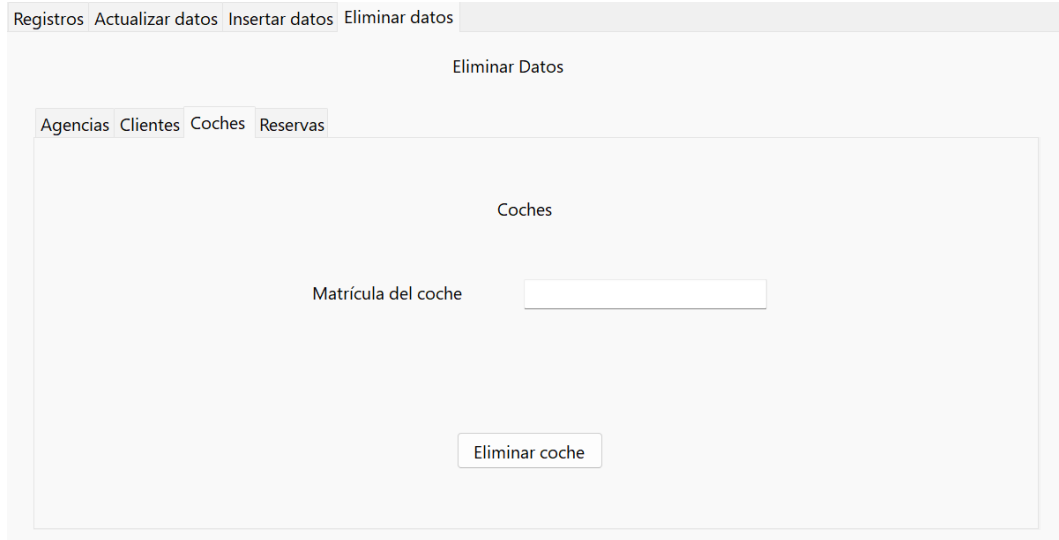
Eliminar cliente

Para eliminar un cliente solo se usa el DNI y el Nombre completo, esto con el fin de tener una confirmación de que sea el cliente indicado y no se copió erróneamente el DNI del cliente que se desea eliminar. Usamos la función **delete\_clientes\_Click**.

```
1 private void delete_clientes_Click(object sender, EventArgs e)
2 {
3     string cliente_dni = delete_dni_cliente.Text;
4     string cliente_name = delete_name_cliente.Text;
5
6     string query = @"
7         DELETE FROM clientes WHERE dni = {cliente_dni}
8         AND nombre_completo = '{cliente_name}';";
9
10    delete_query(query, "Cliente");
11    delete_name_cliente.Text = "";
12    delete_dni_cliente.Text = "";
13 }
```

La función ejecuta la sentencia SQL usando los dos valores extraídos de la interfaz, y al final limpia los valores puestos en la interfaz.

### 3. Eliminar coches.



En la interfaz solo tomamos el valor de “Matrícula de coche” porque es único para cada coche, la razón de no usar el id es para no tener que mostrarlo en la parte de registros con el fin de que sea más fácil la gestión y no depende de un número asignado en orden, sino de uno **Lógico** como el número de matrícula del coche. Usamos la función **button\_delete\_coche\_Click** para la eliminación en la base de datos.

```
1 private void button_delete_coche_Click(object sender, EventArgs e)
2 {
3     string matricula_coche = delete_coche_matricula.Text;
4
5     string query = $"DELETE FROM coches WHERE matricula = '{matricula_coche}'";
6     delete_query(query, "Registro del Coche");
7     delete_coche_matricula.Text = "";
8 }
```

La función toma el valor del número de la matrícula del coche para usarlo en la sentencia SQL y eliminarlo de la base de datos.

#### 4. Eliminación de Reservas.

Registros Actualizar datos Insertar datos Eliminar datos

Eliminar Datos

Agencias Clientes Coches Reservas

Reservas

DNI cliente

Fecha inicio  yy/mm/dd hh:mm:ss

Nombre de agencia

Eliminar reserva

Al momento de crear la interfaz para eliminar las reservas no se pensaba implementar el id de Reservas para que se usen datos útiles, en este caso se utilizó el DNI cliente, la fecha de inicio y el nombre de la agencia, porque no podría coincidir estos 3 valores en otras Reservas.

Para poder eliminar la reserva se necesita el "id\_cliente" la "fecha de inicio" y "id\_agencia", para esto realizamos una **subconsulta** dentro de nuestra sentencia SQL. Usamos la función **button3\_Click** porque se generó automáticamente.

```
1 private void button3_Click(object sender, EventArgs e)
2 {
3     string dni_cliente = delete_reserva_dni_cliente.Text;
4     string begin_date = delete_reserva_begin_date.Text;
5     string agencia_name = delete_reserva_agencia_name.Text;
6
7     string query = $"
8         DELETE FROM reservas
9         WHERE
10             id_cliente = (SELECT id_cliente FROM clientes WHERE dni = '{dni_cliente}') AND
11             fecha_inicio = '{begin_date}' AND
12             id_agencia = (SELECT id_agencia FROM agencias WHERE nombre_agencia = '{agencia_name}')
13     ";
14
15     delete_query(query, "Reserva");
16     delete_reserva_dni_cliente.Text = "";
17     delete_reserva_begin_date.Text = "";
18     delete_reserva_agencia_name.Text = "";
19 }
```

En la sentencia podemos observar las **subconsultas** en la línea 10 y 12 para el "id\_cliente" e "id\_agencia", utilizando los valores de dni y nombre de la agencia extraídos de la interfaz del usuario



## 5. Prueba del sistema.

Para las pruebas del sistema se realizó una grabación corta donde se hace uso de cada una de las funcionalidades como : “Consultar tabla”, “Insertar datos”, “Actualizar datos” y “Eliminar datos”.

Por medio de este link se puede visualizar la prueba en tiempo real del sistema.

[Presentación Proyecto 2B Fundamentos de base de datos - Enrique Pérez.mp4](#)

## 6. Conclusiones

Como conclusiones después del desarrollo, tenemos como principal:

1. Creación del proyecto:  
El proyecto tomó tiempo y se tuvo que dedicar bastante a resolver problemas con las consultas SQL y las ejecuciones, pero se logró realizar correctamente
2. Uso del sistema:  
Aunque el sistema fue realizado con una finalidad puramente funcional sigue siendo un sistema bastante intuitivo y capaz de permitir una gestión clara de un sistema de ventas o cualquier tipo de sistema comercial de forma aceptable.
3. Oportunidad de mejora:  
El sistema tiene muchas oportunidades de mejora, tanto visuales como de interfaz de navegación, también se puede mejorar bastantes operaciones para que el manejador de este sistema sea capaz de usarlos de una manera mucho más rápido, como añadiendo funciones al momento de hacer clic en cada registro y demás.
4. Aprendizaje  
Gracias a este proyecto se aprendió bastante el uso de C# y las posibles formas de crear estos sistemas de forma sencilla y rápida.

### Código del proyecto.

El proyecto realizado se adjunta como un .zip donde contiene todo lo que Visual studio creo para poder funcionar correctamente, se recuerda que la base de datos es distinto del sistema pero se adjunta el schema.sql para crear las tablas y el insert.sql para los datos de prueba.