

# Introducción a las Bases de Datos

## 1. Presentación

Hasta ahora nuestros programas trabajan con datos externos almacenados en un fichero de texto, como, por ejemplo, un fichero de equipos de fútbol. El programa debe abrir el fichero y cargar los datos en una estructura adecuada (por ejemplo, una lista de equipos) y luego realizar las operaciones que desee mediante recorridos, búsquedas, inserciones, eliminaciones de datos en la lista. Finalmente, el programa debe salvar en el fichero de texto los datos tal y como han quedado tras la sesión de trabajo.

Las bases de datos facilitan mucho el trabajo con datos externos, sobre todo cuando los datos externos son muchos, de diversos tipos y con diferentes relaciones entre ellos. Una base de datos no es más que un fichero con datos (normalmente muchos) y un sistema de gestión que permite realizar operaciones con esos datos (consultas, eliminaciones, inserciones, etc.). Podemos tener, por ejemplo, una base de datos de alumnos, profesores y asignaturas (tres entidades diferentes) y realizar operaciones sobre ellas, como por ejemplo: dime cuáles son los alumnos de una asignatura determinada, cuáles son las asignaturas impartidas por un determinado profesor, añadir una nueva asignatura, etc. Este tipo de operaciones ya están implementadas en la base de datos, de manera que el programa que use esa base de datos simplemente debe especificar el tipo de consulta que desea y procesar los resultados como estime oportuno. Con todo ello, se facilita enormemente el uso de grandes cantidades de datos.

En esta práctica vas a aprender a utilizar bases de datos muy sencillas, con una sola entidad (por ejemplo, clientes) y con consultas sencillas (por ejemplo, dime el nombre de un cliente a partir de su DNI, muéstrame todos los clientes que hay en ese momento en la base de datos, etc.). En otras asignaturas más avanzadas aprenderás a crear y usar bases de datos más complejas, con varias entidades (clientes, productos, compras, etc.) y consultas más sofisticadas (dime cuántos clientes han comprado un producto dado, o cuántas compras ha hecho un cliente en el último mes).

Para ser más concreto, con esta práctica guiada serás capaz de:

1. Crear una base de datos sencilla usando SQLite.
2. Construir consultas básicas en el lenguaje SQL, permitiéndote buscar y modificar los datos almacenados en una tabla de tu base de datos.
3. Utilizar una base de datos en tu programa Visual C#.

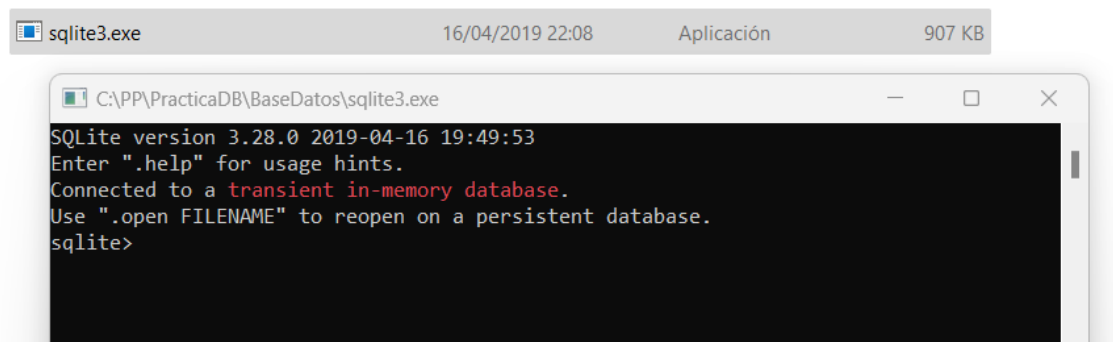
## 2. Creación de la base de datos

SQLite es un sistema de gestión de base de datos contenido en una pequeña librería en C. Esta librería proporciona un gestor de base de datos transaccional compatible con el lenguaje SQL y autocontenido, que no necesita ni de un servidor ni de configuración, con capacidad de gestionar una base de datos basada en un único fichero de datos de hasta 2 TB. SQLite es un proyecto de código libre y es una de las bases de datos embebida más popular del mundo, ideal para pequeñas

aplicaciones que no necesitan de un servidor de base de datos. SQLite no es una base de datos que siga el modelo cliente-servidor (como lo son Oracle o MySQL).

Para crear o editar una base de datos SQLite necesitamos un programa que nos permita gestionar las bases de datos de este tipo, el más simple de todos es el ejecutable de línea de comandos **sqlite3**. Este programa ocupa tan solo unos KB y no requiere de instalación, tan solo que lo copiemos en una carpeta de nuestro disco duro.

1. En la carpeta donde vayas a realizar la práctica, crea una carpeta para la base de datos (ej. `C:\...\PP\PracticaDB\BaseDatos`) y copia en ella el ejecutable **sqlite3.exe** (disponible en Atenea junto a este enunciado). Una vez tengas el ejecutable en tu disco duro, ejecútalo haciendo doble-clic sobre él. Se te abrirá un terminal de Windows con el intérprete de comandos de SQLite:



Escribe ahí el siguiente comando:

```
.open practica.db
```

Acabas de crear el fichero *practica.db* que va a contener tu **base de datos**.

2. Ahora necesitamos **crear la tabla** que va a almacenar nuestros datos. Para ello, dentro de *sqlite3*, ejecuta el siguiente comando SQL:

```
CREATE TABLE clientes (DNI varchar primary key, nombre varchar, edad integer);
```

Este comando que acabas de introducir es una sentencia SQL que ha creado una tabla de nombre *clientes* que contiene tres columnas *DNI*, *nombre* y *edad* del tipo de datos respectivamente texto, texto y número entero. La palabra clave *primary key* le indica al gestor de base de datos que esa columna de la tabla es nuestra clave, que solo podrá contener valores únicos y que es la que va a utilizar del gestor de base de datos para indexar la tabla.

3. SQLite3 permite comandos de gestión (típicamente empiezan con “.”). Por ejemplo, puedes ver la descripción de la tabla *clientes* que has creado con el comando:

```
.schema clientes
```

4. Ahora vamos a **insertar** datos a la tabla *clientes*. Para ello usaremos el comando SQL INSERT:

```
INSERT INTO clientes VALUES ('22222222T', 'enric', 25);  
INSERT INTO clientes VALUES ('44444444T', 'pepe', 35);
```



Habrás observado que los valores de texto, como el DNI o el nombre, se escriben entre **comillas simples** (típicamente en la tecla a la derecha del 0 en el teclado). Algunos procesadores de texto suelen cambiar las “comillas rectas” (') por “comillas tipográficas” ('). Lo mismo ocurre con las **comillas dobles**. Esto hará que si usas copiar-y-pegar para escribir tu código, este de error al ejecutarse. Si esto sucede, **reescribe las comillas correctamente en tu código**.

5. Ahora que ya tenemos datos en la tabla, podemos **consultarlos** ejecutando el comando:

```
SELECT * FROM clientes;
```



El comando *SELECT* es, al igual que *CREATE*, una instrucción con la que formamos una sentencia SQL. **Cuando introduzcas una sentencia SQL esta tiene que acabar siempre en un “;”**. Por el contrario, todos los comandos que introduces en *sqlite3* y que empiezan por un “.” son comandos propios del gestor de base de datos. Estos comandos no son SQL y no se acaban nunca en “;” (al no ser comandos estandarizados, cada gestor de base de datos utiliza sus propios comandos).

6. Ahora abandona el programa *sqlite3* con el comando:

```
.quit
```

Los datos insertados habrán quedado almacenados en el fichero *practica.db*.

Otros comandos que te pueden ser útiles de *sqlite3* son:

*.tables* => muestra todas las tablas que contiene la base de datos

*.open DB\_FILE* => carga la base de datos de un fichero de nombre *DB\_FILE*. Si el fichero no existe, se crea una nueva base de datos vacía.

*.save DB\_FILE* => guarda la base de datos con la que estamos trabajando en un fichero de nombre *DB\_FILE*.

*.help* => permite consultar la lista de comando de *sqlite3* (no la sintaxis SQL).

### 3. Introducción a *Structured Query Language (SQL)*

El lenguaje de consulta estructurado (SQL) es un lenguaje de programación de "alto nivel" para acceder a los datos operativos almacenados en una base de datos relacional.

En el lenguaje SQL, el comando SELECT permite realizar búsquedas desde una base de datos.

```
SELECT * FROM clientes;
```

Esta consulta devuelve todos los datos almacenados en una tabla. Si sólo deseas recuperar algunos campos de cada entrada, por ejemplo "DNI" y "nombre", ejecuta esta consulta:

```
SELECT DNI, nombre FROM clientes;
```

Con la cláusula WHERE puedes limitar el SELECT a un subconjunto de entradas de la tabla que coincidan con un criterio determinado. La cláusula ORDER BY te permite ordenar los resultados devueltos por un atributo ascendente o descendente.

7. Ejecuta las siguientes **consultas** y observa los resultados:

```
SELECT * FROM clientes;
SELECT DNI, nombre FROM clientes;
SELECT * FROM clientes WHERE edad>18 ORDER BY edad ASC;
SELECT * FROM clientes WHERE edad>18 ORDER BY edad DESC;
SELECT * FROM clientes WHERE edad>18 AND edad<30;
SELECT * FROM clientes WHERE edad<18 OR edad>30;
SELECT * FROM clientes WHERE DNI='111111111H';
SELECT * FROM clientes WHERE nombre LIKE 'j%';
SELECT * FROM clientes WHERE DNI LIKE '%H';
SELECT count(*) FROM clientes;
SELECT max(edad) FROM clientes;
```

El lenguaje de manipulación de datos (DML) consta de tres instrucciones que te permiten añadir nuevos datos, modificarlos y eliminar los datos almacenados en una base de datos:

- INSERT: añade nuevos registros a una tabla.
- UPDATE: modifica los valores de los atributos de los registros ya almacenados en una tabla.
- DELETE: elimina registros de una relación.

8. Ejecuta las siguientes instrucciones y observa cómo **modifican** los datos almacenados en tu tabla (ejecuta "SELECT \* FROM clientes;" después de cada una):

```
INSERT INTO clientes VALUES ('22222222T', 'enric', 25);
UPDATE clientes SET nombre='jordi' WHERE DNI='22222222T';
UPDATE clientes SET edad=edad+1 WHERE DNI='22222222T';
DELETE FROM clientes WHERE DNI='22222222T' and edad<18;
DELETE FROM clientes WHERE DNI='22222222T';
```

El lenguaje de definición de datos (DDL) es la parte de SQL que se utiliza para crear una base de datos. También es la herramienta que se utiliza para modificar la estructura de una base de datos existente o destruirla después de que ya no se necesite. El DDL consta de tres declaraciones:

- CREATE: crea una nueva tabla.
- ALTER: modifica la estructura de una tabla existente.
- DROP: borra una tabla existente.

Un ejemplo sería la instrucción que has usado previamente para crear la tabla "clientes":

```
CREATE TABLE clientes (DNI varchar primary key, nombre varchar, edad integer);
```

#### 4. Instalación del conector de la base de datos

Para poder utilizar una base de datos desde un lenguaje de programación necesitamos que el driver de acceso a la base de datos para ese lenguaje esté instalado. La función de este driver o conector es ofrecer un interfaz de comunicación común entre las clases que tiene el lenguaje de programación para el acceso a base de datos y los diferentes gestores de base de datos, de tal manera que con una misma API el programador pueda utilizar cualquier base de datos. En nuestro caso el conector de SQLite no viene por defecto con la instalación de .net por lo que debemos instalarlo nosotros.

Para instalar el conector de SQLite, copia el fichero **System.Data.SQLite.dll** en la carpeta donde estás realizando la práctica (ej. en C:\...\PP\PracticaDB\BaseDatos). Más adelante, deberás añadir este fichero como referencia a tu proyecto.

#### 5. Preparación de una clase para operar con la base de datos

Para trabajar con la base de datos, nuestro programa debe tener una clase en la que programaremos los diferentes tipos de consultas que queremos hacer (cada tipo de consulta que queramos hacer será un método de la clase).

9. Crea una aplicación de Visual C# con un proyecto de tipo **Biblioteca de clases (.NET Framework)** que se llame *GestionClientes*. En esa biblioteca de clases vamos a tener una clase que se llamará *Gestion*.

Para agregar al proyecto el conector de SQLite, pulsa con el botón derecho del ratón en las referencias del proyecto y selecciona "Agregar referencia...". Una vez en la ventana del gestor de referencias, selecciona el botón "Examinar..." (ver Figura 1), navega hasta la carpeta donde descargaste la DLL del punto 4 (**System.Data.SQLite.dll**) y agrégala.

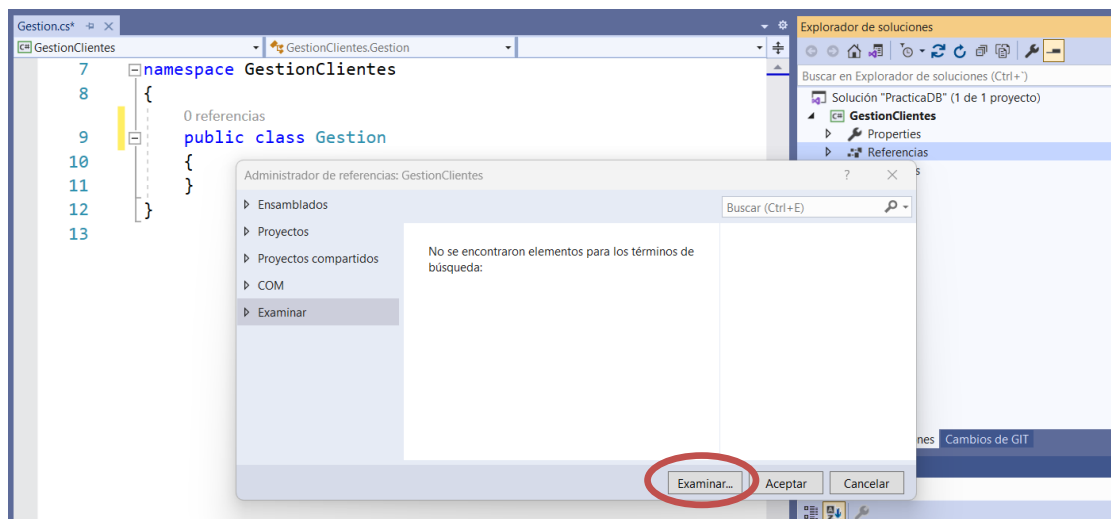


Figura 1

Observa la figura 2. En la clase *Gestion* hemos añadido dos *using* necesarios para trabajar con la base de datos (*System.Data* y *System.Data.SQLite*). También hemos añadido un conector para acceder a la base de datos (*cnx*) y hemos empezado a escribir el código de un método de la clase que será el que habrá que ejecutar para abrir la base de datos (*Iniciar*). En las referencias del proyecto aparece la DLL que hemos agregado antes (*System.Data.SQLite.dll*).

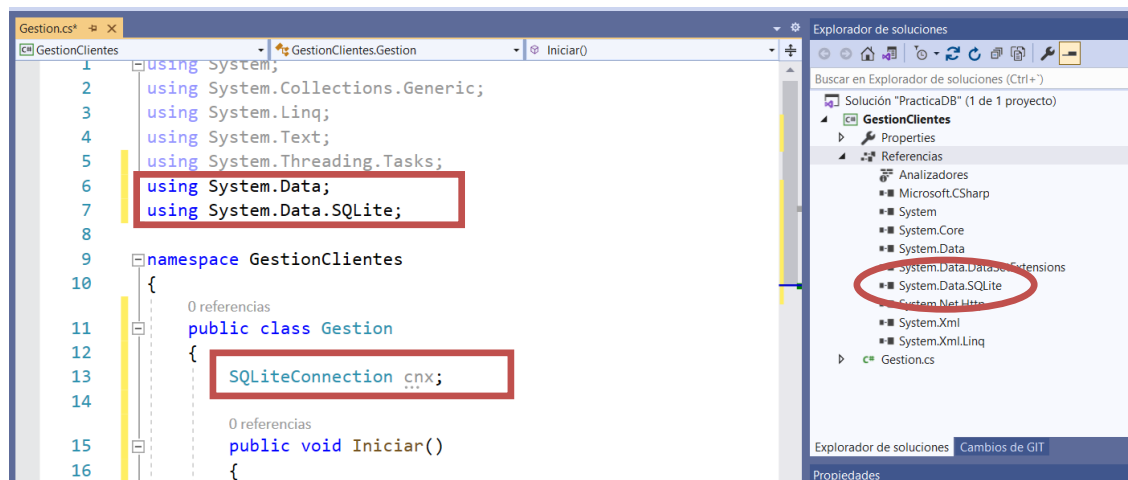


Figura 2

10. El código que debes escribir en el método *Iniciar* es el siguiente:

```
public void Iniciar()
{
    string dataSource = "Data Source=..\..\..\..\BaseDatos\\basedatos.db";
    cnx = new SQLiteConnection(dataSource);
    cnx.Open();
}
```

Observa que al crear la conexión debes indicar el camino completo para que encuentre la base de datos. En el código de ejemplo se indica la ruta relativa suponiendo que la carpeta “BaseDatos” se encuentra en la misma carpeta que la carpeta “GestionClientes” (como se muestra en la Figura 3). Corrige la ruta convenientemente en función de la ubicación de tus ficheros, o reorganiza tus carpetas convenientemente (en ese caso, vuelve a añadir la referencia a la DLL, ya que también habrá cambiado de sitio).

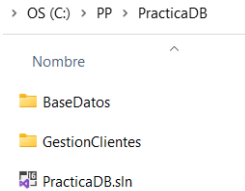


Figura 3

También puedes indicar la ruta absoluta, como, por ejemplo:

```
"Data Source=C:\\PP\\PracticaDB\\BaseDatos\\basedatos.db";
```

Pero ten presente que si mueves el código (a otro ordenador u otra ubicación dentro de tu ordenador), la ruta dejará de ser válida y tendrás que corregirla.



Presta especial atención a la línea de código dónde especificas la **ubicación del fichero** de tu base de datos SQLite (“Data Source=...”), puesto que si te equivocas de directorio el programa no va a dar ningún error de conexión a la base de datos, si no que va crear un fichero de base de datos vacío en la ubicación indicada por “Data Source”. Luego cuando intentes acceder a una tabla que no existirá en el nuevo fichero de datos te saltará una excepción.

11. Vamos a necesitar también un método para cerrar la base de datos:

```
public void Cerrar()  
{  
    cnx.Close();  
}
```

12. Y ahora vamos a añadir un método para obtener todos los datos que tiene la base de datos en ese momento. El código de ese método es el siguiente:

```
public DataTable GetClientes()  
{  
    DataTable dt = new DataTable();  
  
    string sql = "SELECT * FROM clientes";  
    SQLiteDataAdapter adp = new SQLiteDataAdapter(sql, cnx);  
    adp.Fill(dt);  
  
    return dt;  
}
```

El método (que se llama *GetClientes*) es una función que retorna un objeto de la clase *DataTable*. Lo más importante es que observes la forma en que se especifica la consulta que queremos hacer en la base de datos.

Las consultas se realizan, como ya hemos visto en el punto 3, en un lenguaje que se llama SQL. La consulta "SELECT \* FROM clientes" se interpreta como "Selecciona todos los registros de la tabla clientes".

El resultado de la consulta (en este caso, todos los clientes) queda almacenado en el objeto *dt* (de la clase *DataTable*) que se retorna como resultado de la función. Después veremos cómo debe usar ese objeto el programa que use la clase que estamos construyendo.

Si quieres hacer otros tipos de consultas simplemente tienes que cambiar la frase que especifica la consulta en lenguaje SQL. El resto del código es el mismo. Más adelante veremos otros ejemplos de consultas en SQL.

Ya tenemos lista la clase con la que podemos hacer tres operaciones: abrir la base de datos, obtener todos los clientes de esa base y cerrar la base de datos.

## 6. Realizar consultas y mostrar resultados desde una aplicación de consola

Observa el siguiente código que utiliza la clase *Gestion* creada en la sección anterior para mostrar todos los registros de la base de datos:

```
using System.Data;
using GestionClientes;

namespace DbConsola
{
    class Program
    {
        static void Main(string[] args)
        {
            // Crear un objeto de la clase Gestion
            Gestion mi_base = new Gestion();

            // Abrir la conexión con la base de datos
            mi_base.Iniciar();

            // Obtener todos los registros de la base de datos y mostrarlos
            DataTable dt = mi_base.GetClientes();
            Console.WriteLine("Clientes en la base de datos:");
            for (int i = 0; i < dt.Rows.Count; i++)
            {
                Console.WriteLine("DNI:      {0}", dt.Rows[i]["DNI"]);
                Console.WriteLine("Nombre: {0}", dt.Rows[i]["nombre"]);
                Console.WriteLine("Edad:   {0}", dt.Rows[i]["edad"]);
                Console.WriteLine();
            }

            // Cerrar la conexión
            mi_base.Cerrar();

            Console.ReadKey();
        }
    }
}
```



El programa empieza creando un objeto de la clase *Gestion* (*mi\_base*). A continuación, invoca al método *Iniciar* para establecer la conexión con la base de datos. A partir de ese momento, ya podemos utilizar los métodos que hayamos programado para realizar consultas u otras operaciones con la base de datos.

En este ejemplo, se invoca al método *GetClientes*, que nos devuelve un *DataTable* con todos los registros de la base de datos. Para mostrar el *DataTable*, iteramos fila a fila, sabiendo que la propiedad *dt.Rows.Count* indica el número de filas de la tabla *dt*. Para acceder a una celda determinada de la tabla, debemos indicar el índice de la fila y el índice o nombre de la columna. Por ejemplo, *dt.Rows[0]["DNI"]* haría referencia a la columna "DNI" de la primera fila de la tabla.

Por último, cerramos la conexión invocando el método *Cerrar*.

13. Añade a tu solución un proyecto del tipo **Aplicación de Consola (.NET Framework)** llamado *DbConsola*. Recuerda añadir como referencia el proyecto *GestionClientes* (que contiene la clase *Gestion*) y establecer el nuevo proyecto como proyecto de inicio. Copia en código anterior en el *Program.cs* y comprueba su funcionamiento

## 7. Un formulario para realizar consultas y mostrar los resultados

Añadiremos ahora a la aplicación un proyecto de tipo **Formularios de Windows (.NET Framework)** que usará la clase *Gestion* para acceder a la base de datos y mostrará al usuario los resultados de las consultas.

14. Añade ese nuevo proyecto (que se llamará *InterfazDB*) a la aplicación, e incorpora al formulario principal un botón y un componente del tipo *DataGridView*, al que llamaremos *vistaClientes*. En la figura 4 puedes ver el diseño de este nuevo formulario:

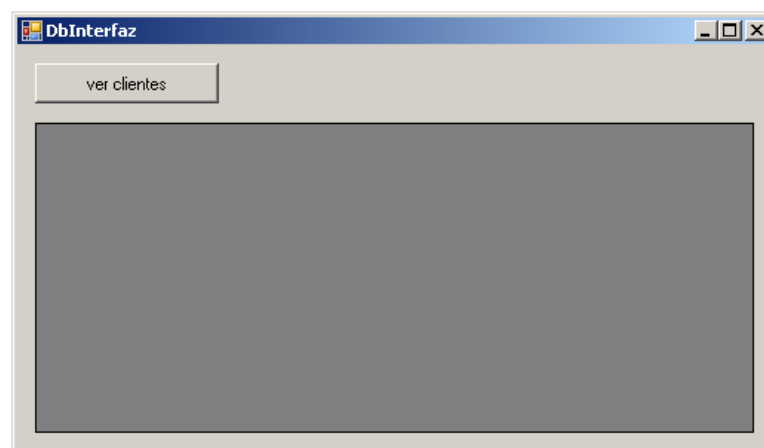


Figura 4

15. En el código del formulario, declaramos un objeto *mi\_base* de la clase *Gestion* (ver figura 5). Para poder usar la clase en el código del formulario debes añadir *using GestionClientes*. Recuerda agregar la referencia a la librería de clases y establecer el proyecto de inicio.

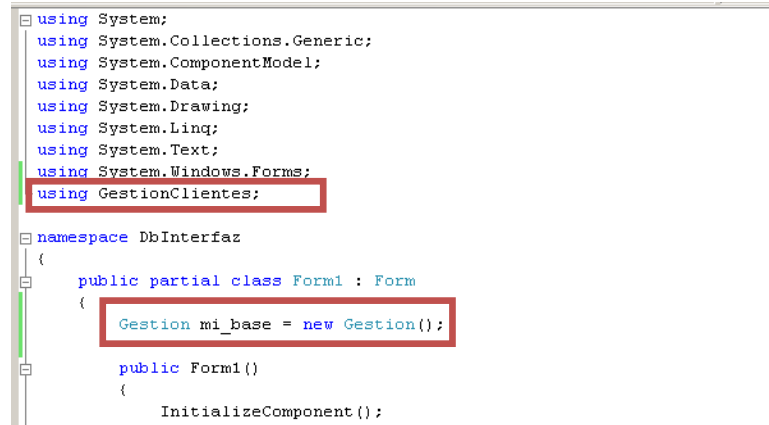


Figura 5

16. En el tratamiento del evento *Load* del formulario daremos la orden de abrir la base de datos:

```

private void Form1_Load(object sender, EventArgs e)
{
    mi_base.Iniciar();
}

```

17. En el tratamiento del evento *Click* del botón ordenaremos la consulta a la base de datos y mostraremos el resultado en el *DataGridView*:

```

private void button1_Click(object sender, EventArgs e)
{
    DataTable consulta = mi_base.GetClientes();
    vistaClientes.DataSource = consulta;
    vistaClientes.Refresh();
}

```

Tal y como muestra el código, se trata simplemente de llamar al método que realiza la consulta y que devuelve un objeto de la clase *DataTable*. Ese objeto se carga directamente sobre el *DataGridView*.

18. Finalmente, no olvides añadir una llamada al método que habíamos preparado para cerrar la base de datos en el tratamiento del evento *FormClosing*.

```

private void Form1_FormClosing(object sender, EventArgs e)
{
    mi_base.Cerrar();
}

```

19. Ejecuta el código y observa el resultado. Aparecerá algo así como lo que muestra la figura 6. Si el programa no hace nada, comprueba que los métodos anteriores estén correctamente asignados con los eventos correspondientes (en el icono del rayo de la ventana de propiedades).

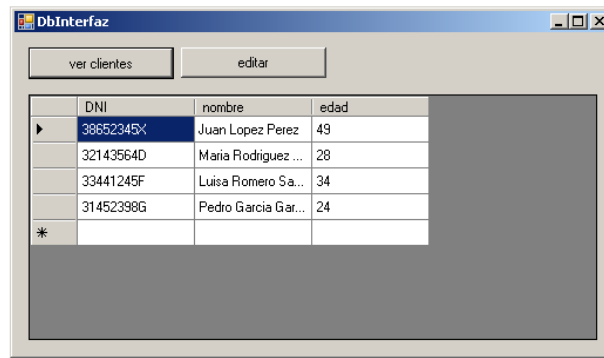


Figura 6

Ya sabes cómo trabajar con un *DataGridView*. Puedes eliminar por ejemplo las barras de desplazamiento que aparecen abajo y a la derecha, quitar la columna de la izquierda que no hace nada, y ajustar el tamaño de las columnas para que todos los datos se vean bien.

Tal y como acabamos de ver, nuestras consultas a la base de datos devolverán un objeto de la clase *DataTable* que puede visualizarse fácilmente en un *DataGridView*. La cuestión es entonces realizar correctamente la consulta SQL.

## 8. Más consultas SQL de selección

A continuación, tienes más consultas SQL que puedes probar.

20. Simplemente modifica el código de la consulta en el método *GetClientes* de la clase *Gestion* que has construido y observa el resultado que se muestra en el *DataGridView*.

```
"SELECT * FROM clientes ORDER BY edad"
```

Esta consulta devuelve un *DataTable* con todos los clientes, pero ordenados por orden ascendente de la edad.

```
"SELECT nombre FROM clientes WHERE edad=34"
```

Esta consulta devuelve sólo el nombre de los clientes cuya edad es 34.

21. Intenta ahora adivinar qué hacen las consultas siguientes. Luego pruébalas a ver si has acertado:

```
"SELECT nombre FROM clientes WHERE edad > 20 AND nombre LIKE 'J%'"
"SELECT nombre FROM clientes WHERE edad = (SELECT max(edad) FROM clientes)"
```

## 9. Consultas SQL de no selección

Es común que además de consultar la información que tenemos almacenada en la base de datos nos interese modificarla, por ejemplo, añadiendo un nuevo registro, eliminando un registro ya existente o bien modificando el valor de un campo.

Intenta ahora adivinar qué hacen las consultas siguientes:

```
"INSERT INTO clientes VALUES ('12345558D', 'Antonio Marquez Linares',72)"  
"UPDATE clientes SET edad=24 WHERE DNI='12345558D'"  
"DELETE FROM clientes WHERE edad=24"
```

22. Ahora carga con el gestor *sqlite3* la base de datos que has creado para la práctica y observa que sucede cuando ejecutas las sentencias SQL anteriores.

Efectivamente, se trata de comandos que no hacen ninguna selección de información de la base de datos, sino que hacen modificaciones en la base de datos. Por tanto, no tiene sentido visualizar en un *DataGridView* la información seleccionada porque no hay información seleccionada. Estos comandos se tratan de una manera diferente a los comandos de tipo SELECT.

Fíjate en el siguiente código:

```
public int ponEdad(string dni, int edad)  
{  
    string sql = "UPDATE clientes SET edad =" + edad + " WHERE DNI='" + dni + "'";  
  
    SQLiteCommand command = new SQLiteCommand(sql, cnx);  
    int affectedRows = command.ExecuteNonQuery();  
  
    if (affectedRows == 1)  
        return 0;  
    else  
        return -1;  
}
```

El método *ponEdad* modifica la edad del cliente con un DNI determinado; devuelve 0 si lo ha modificado, -1 en caso contrario. La sentencia clave es *command.ExecuteNonQuery()*, que realiza la operación SQL en la base de datos (una modificación de la base de datos y no una consulta) y devuelve como resultado el número de filas de la base de datos que han sido afectadas por la operación. Podemos usar ese valor para ver si todo ha ido bien.

En el siguiente apartado vamos utilizar el código que acabamos de ver en un ejemplo completo que combina una selección con una modificación de un registro de la base de datos.

## 10. Cómo acceder a la información de un DataTable

No siempre queremos visualizar en un *DataGridView* el resultado de una consulta. Por ejemplo, imagina que queremos hacer un formulario que pida al usuario el DNI de un cliente, muestre su nombre y edad y además incremente en la base de datos la edad.

El formulario debe tener el aspecto que muestra la figura 7:

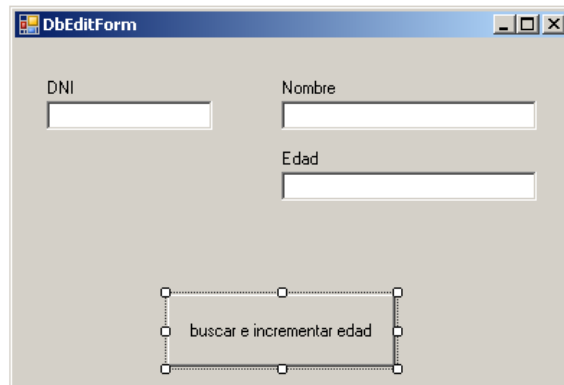


Figura 7

En ese formulario, el cuadrado de texto en el que el usuario debe escribir el DNI se llama *DNIIn*, y las *label* en las que se mostrará el resultado se llaman *nombre* y *edad*.

Ahora debemos añadir a la clase *Gestion* un método para obtener el cliente con un DNI determinado. El código de ese método es el siguiente:

```
public DataTable buscarCliente(string dni)
{
    DataTable dt = new DataTable();

    string sql = "SELECT * FROM clientes where DNI='" + dni + "'";
    SQLiteDataAdapter adp = new SQLiteDataAdapter(sql, cnx);
    adp.Fill(dt);

    return dt;
}
```

También tenemos que añadir un método para asignar una edad al cliente que tiene un DNI determinado. El código es el que hemos visto en el apartado 8:

```
public int ponEdad(string dni, int edad)
{
    string sql = "UPDATE clientes SET edad =" + edad + " WHERE DNI='" + dni + "'";

    SQLiteCommand command = new SQLiteCommand(sql, cnx);
    int affectedRows = command.ExecuteNonQuery();

    if (affectedRows == 1)
        return 0;
    else
        return -1;
}
```

El código para el tratamiento del evento correspondiente al botón del formulario es el siguiente:

```

private void cmdSearch_Click(object sender, EventArgs e)
{
    DataTable consulta = mi_base.buscarCliente(inDni.Text);

    if (consulta.Rows.Count == 0) {
        MessageBox.Show("No hay ningún cliente con ese DNI");
    }
    else if (consulta.Rows.Count > 1) {
        MessageBox.Show("Hay más de un cliente con ese DNI");
    }
    else {
        outNom.Text = consulta.Rows[0][1].ToString();
        outEdat.Text = consulta.Rows[0][2].ToString();
    }

    int r = mi_base.ponEdad(inDni.Text, Convert.ToInt32(outEdat.Text) + 1);

    if (r == 0) {
        MessageBox.Show("Se ha modificado un registro");
    }
    else {
        MessageBox.Show("Error");
    }
}

```

Primero buscamos los clientes con el DNI introducido por el cliente y nos aseguramos de que hay solo un cliente con ese DNI. En el caso de que sólo haya un cliente, en el *DataTable* consulta tenemos sólo una fila de datos (la fila 0) y en sucesivas columnas de esa fila tenemos el DNI, nombre y edad del cliente. Podemos acceder a la información como si fuese una matriz (en fila 0 columna 1 está el nombre y en la fila 0 columna 2 está la edad). Después llamamos al método para fijar la edad, pasándole la edad incrementada.

Finalmente, no olvides añadir donde estimes oportuno una llamada a los métodos que habíamos preparado para abrir y cerrar la base de datos.

## 11. Errores comunes

Estas son algunas de las fuentes de error más frecuentes al trabajar con bases de datos:

- En las secuencias SQL, las cadenas deben estar entre comillas simples, los números no.
- No defines nombres de tablas o columnas que contengan espacios o caracteres especiales.
- Las instrucciones SQL deben completarse con un punto y coma (;) en el editor, pero este carácter no es necesario cuando se escriben las consultas en C#.
- No olvides llamar al método *Iniciar* cada vez que crees una nueva instancia de objeto de *Gestion*. Es necesario abrir la conexión a la base de datos antes de poder ejecutar una operación SQL.
- Sitúa el conector a la base de datos SQLite dentro de una carpeta de la solución. Con esto conseguirás hacer tu programa más portable, ya que podrás mover el ejecutable a otro ordenador sin tener que instalar el driver de la base de datos para ejecutarlo.
- De igual forma, sitúa el archivo de tu base de datos dentro de una carpeta de la solución para moverlo con tu proyecto. Revisa la ruta relativa desde la carpeta bin\Debug de la aplicación.

## 12.Ejercicio

Construye una aplicación que trabajará con una base de datos de clientes, de acuerdo con las siguientes especificaciones:

- Los clientes deben tener los siguientes atributos: DNI, nombre, edad y saldo (un número real).
- La aplicación debe utilizar los formularios necesarios para realizar de manera cómoda y amigable las siguientes operaciones con la base de datos:
  1. Mostrar todos los clientes en orden de mayor a menor edad.
  2. Añadir un cliente nuevo
  3. Eliminar un cliente identificado por su DNI
  4. Mostrar los datos del cliente que tenga mayor saldo
  5. Mostrar los clientes cuya edad esté entre dos valores que introducirá el usuario
  6. Sumar una cantidad al saldo de un cliente identificado por su DNI
  7. Mostrar el saldo acumulado de todos los clientes