

## 11. Acceso a bases de datos relacionales

### 11.1. Nociones mínimas de bases de datos relacionales

Una base de datos relacional es una estructura mucho más compleja (pero también más versátil) que un simple fichero de texto o binario. Existen ciertos conceptos que debemos conocer antes de empezar a trabajar con ellas:

- En una base de datos relacional hay varios bloques de datos, llamados "**tablas**" (por ejemplo, la tabla de "clientes", la tabla de "proveedores", la tabla de "productos"...).
- Cada tabla está formada por una serie de "**registros**" (por ejemplo, el cliente "Juan López", el cliente "Pedro Álvarez"...).
- Cada registro está formado por varios "**campos**" (de cada cliente podemos almacenar su nombre, su dirección postal, su teléfono, su correo electrónico, etc).
- Además, existen **relaciones** entre las tablas: por ejemplo una factura es para un cierto cliente, de modo que la tabla "factura" se relaciona con la tabla "cliente", y el nombre de esta relación se podría llamar "ser para" (las relaciones se suelen nombrar empleando verbos).

Vamos a aplicar estos conceptos básicos a la vez que vamos conociendo el lenguaje de consulta más habitual, llamado SQL.

### 11.2. Nociones mínimas de lenguaje SQL

El lenguaje SQL (Structured Query Language) es un lenguaje muy extendido, que permite realizar consultas a una base de datos relacional. Por eso, vamos a ver algunas de las órdenes que podemos usar para almacenar y obtener datos, y posteriormente las aplicaremos desde un programa en C# que conecte a una base de datos relacional. Cuando ya seamos capaces de guardar datos y recuperarlos, veremos alguna orden más avanzada para extraer la información o para modificarla.

#### 11.2.1. Creando la estructura

Como primer ejemplo, veremos las órdenes necesarias para crear una base de datos sencilla, que llamaremos "ejemplo1". Esta base de datos contendrá una única tabla, llamada "agenda", que contendrá algunos datos de cada uno de

nuestros amigos. Como es nuestra primera base de datos, no pretendemos que sea perfecta, sino sencilla, así que apenas guardaremos tres datos de cada amigo: el nombre, la dirección y la edad.

(**Nota:** si quieres probar ya las órdenes que vas a conocer, puedes optar por instalar algún gestor de bases de datos de código abierto, como MySQL o MariaDB; aun así, no es necesario, porque en el siguiente apartado las aplicaremos a un gestor de bases de datos pequeño y que se puede "incrustar" con una relativa facilidad en aplicaciones creadas en C# y en otros lenguajes, llamado SQLite. De hecho, la mayoría de las nociones que veamos de SQL se podrán probar también desde la consola de SQLite).

Para crear la base de datos que contendrá todo, en MySQL y MariaDB usaríamos "create database", seguido del nombre que tendrá la base de datos:

```
create database ejemplo1;
```

Si queremos empleamos la consola de SQLite, podemos optar por lanzar dicha consola indicando el nombre de la base de datos que vamos a crear:

```
sqlite3 ejemplo1.db
```

O bien, ya desde dentro de la consola, usar la orden ".open":

```
.open ejemplo1.db
```

En nuestro primer ejemplo, la base de datos almacenará una única tabla, que contendrá los datos de nuestros amigos. Por tanto, el siguiente paso será decidir qué datos concretos ("campos") guardaremos de cada amigo. Deberemos pensar también qué tamaño necesitaremos para cada uno de esos datos, porque a un gestor de bases de datos relacional habrá que dárselo bastante cuadrado, si queremos que sea eficiente. Por ejemplo, podríamos decidir lo siguiente:

nombre - texto, hasta 20 letras  
dirección - texto, hasta 40 letras  
edad - números, de hasta 3 cifras

Cada gestor de bases de datos tendrá una forma de llamar a esos tipos de datos. Por ejemplo, es habitual tener un tipo de datos llamado "**VARCHAR**" para referirnos a texto hasta una cierta longitud. Para los datos numéricos, se suele disponer de tipos de datos relacionados con la estructura interna de un ordenador, como "INT" para un entero no muy grande, o con alternativas más

cercanas al lenguaje natural, como **NUMERIC(3)** para un número de hasta 3 cifras o **NUMERIC(5,2)** para un número de hasta 5 cifras, 2 de las cuales serían decimales y 3 de las cuales estarían antes de la coma decimal.

También es altamente recomendable escoger un dato que permita distinguir un "registro" de otro. Es habitual usar para eso un "código" inventado a propósito (el código de cliente o el código de artículo, por ejemplo), o un número correlativo (como el número de factura) o un dato identificativo ya existente (como el documento nacional de identidad o el pasaporte de una persona). En nuestro, utilizamos la aproximación simplista y poco realista de dar por sentado que el nombre (incluyendo apellidos) de nuestros amigos es único. Ese campo único, que permite identificar una registro, será lo que llamaremos la "**clave primaria**" de la tabla.

Así, una forma de crear nuestra tabla sería:

```
create table personas (
  nombre varchar(20) primary key,
  direccion varchar(40),
  edad numeric (3)
);
```

### Ejercicios propuestos:

**(11.2.1.1)** Anota la orden SQL que usarías para crear una base de datos "biblioteca" con una tabla "libros", que tendrá los campos "nombre", "titulo" y "paginas". Elige tipos de datos y tamaños adecuados para cada campo.

## 11.2.2. Introduciendo datos

Para introducir datos usaremos la orden "insert into", e indicaremos el nombre de la tabla y, tras la palabra "values", los valores para los campos. Los campos de texto se deberán detallar entre comillas, y para los valores de campos numéricos no deberemos usar comillas (aunque muchos gestores de bases de datos permitirán hacerlo), así:

```
insert into personas values ('juan', 'su casa', 25);
```

Este formato nos obliga a indicar los valores para todos los campos, y exactamente en el mismo orden en que se diseñaron. Si no queremos introducir todos los datos, o queremos hacerlo en otro orden, o no recordamos el orden de creación

con certeza, tenemos otra alternativa: detallar también en la orden "insert" los nombres de cada uno de los campos que realmente vamos a introducir, así:

```
insert into personas
(nombre, direccion, edad)
values (
'pedro', 'su calle', 23
);
```

### Ejercicios propuestos:

**(11.2.2.1)** Anota las órdenes SQL que emplearías para añadir dos libros a tu base de datos, uno usando el formato abreviado y otro con el formato detallado.

### 11.2.3. Mostrando datos

Para ver los datos almacenados en una tabla usaremos el formato "SELECT campos FROM tabla". Si queremos ver **todos** los campos, lo indicaremos usando un asterisco:

```
select * from personas;
```

que, con nuestros datos, daría como resultado (en el intérprete de comandos de algunos gestores de bases de datos, como MySQL):

```
+-----+-----+-----+
| nombre | direccion | edad |
+-----+-----+-----+
| juan   | su casa   | 25   |
| pedro  | su calle  | 23   |
+-----+-----+-----+
```

Si queremos ver sólo **ciertos campos**, detallaremos sus nombres, separados por comas, como:

```
select nombre, direccion from personas;
```

y obtendríamos

```
+-----+-----+
| nombre | direccion |
+-----+-----+
| juan   | su casa   |
| pedro  | su calle  |
+-----+-----+
```

Normalmente no queremos ver todos los datos que hemos introducido, sino sólo aquellos que cumplan cierta **condición**. Esta condición se indica añadiendo un apartado WHERE a la orden "select", así:

```
select nombre, direccion from personas where nombre = 'juan';
```

que nos diría el nombre y la dirección de nuestros amigos llamados "juan":

```
+-----+-----+
| nombre | direccion |
+-----+-----+
|  juan  | su casa   |
+-----+-----+
```

En el caso de MySQL (o MariaDB), es habitual que las búsquedas sean "case insensitive" (independientes de **mayúsculas** y minúsculas), por lo que también aparecerían las personas llamadas "Juan". No será lo esperable en la mayoría de gestores de bases de datos (no ocurrirá, por ejemplo, con SQLite ni con PostgreSQL), y en ciertas circunstancias puede tampoco ocurrir en MySQL. Por eso, si queremos que la búsqueda sea independiente a mayúsculas, por lo general tendremos que ser nosotros quien se asegure (por ejemplo, convirtiendo a mayúsculas tanto el valor del campo como el dato a comparar, antes de comprobar si son iguales).

A veces no queremos comparar con un texto exacto, sino sólo con **parte del contenido** del campo (por ejemplo, porque sólo sepamos un apellido o parte del nombre de la calle). En ese caso, no compararíamos con el símbolo "igual" (=), sino que usaríamos la palabra "like", y para los fragmentos que no conozcamos usaremos el comodín "%", como en este ejemplo:

```
select nombre, direccion from personas where direccion like '%calle%';
```

que nos diría el nombre y la dirección de nuestros amigos cuya dirección contenga la palabra "calle", precedida por cualquier texto (%) y seguida por cualquier texto (%):

```
+-----+-----+
| nombre | direccion |
+-----+-----+
|  pedro  | su calle   |
+-----+-----+
```

Y, cuando esperemos obtener más de un resultado, también es habitual querer recibir los datos en un cierto orden, en vez de desordenados. Para conseguirlo, añadiremos "order by" seguido por el nombre de campo (o campos) que queremos utilizar como criterio de **ordenación**:

```
select nombre, direccion
from personas
where direccion like '%juan%'
order by nombre;
```

### Ejercicios propuestos:

(11.2.3.1) ¿Qué orden utilizarías para mostrar todos tus libros?

(11.2.3.2) ¿Cómo mostrarías todos los datos del libro "It"?

(11.2.3.3) ¿Cómo mostrarías sólo el autor del libro "It"?

(11.2.3.4) ¿Qué orden emplearías para mostrar todos los datos de los libros cuyo título empiece por "E", ordenados por título?

## 11.3. Acceso a bases de datos con SQLite

### 11.3.1. ¿Qué es SQLite?

SQLite es un gestor de bases de datos de pequeño tamaño, que emplea el lenguaje SQL para realizar las consultas, y del que existe tanto una versión en formato de fichero DLL que distribuiríamos junto al ejecutable de nuestro programa, como una versión en forma de fichero en lenguaje C que se podría integrar con los fuentes de nuestro programa... en caso de que usásemos el lenguaje C o C++ en nuestro proyecto.

Usar SQLite en vez de Microsoft SQL Server para nuestros proyectos en C# supone tener que dar un par de pasos de forma artesanal, en vez de usar un instalador totalmente automatizado, a cambio de poder distribuir nuestros proyectos de una forma muy sencilla y con un tamaño cerca a los 2 MB, frente a depender de instaladores externos y de que el usuario de nuestra aplicación tenga que tener instalado SQL Server.

Para acceder a SQLite desde C#, tenemos disponible alguna adaptación de la biblioteca original. Una de ellas es System.Data.SQLite, que se puede descargar de <http://system.data.sqlite.org/> (aunque hay tantas versiones que puede resultar abrumador), pero si quieres ganar tiempo, tienes un pequeño proyecto de ejemplo listo para usar desde Visual Studio en <http://nachocabanes.com/csharp>

### 11.3.2. Guardar datos

Con esta biblioteca, los pasos a seguir a nivel de código para crear una base de datos y guardar información en una base de datos de SQLite serían:

- Crear una conexión a la base de datos, mediante un objeto de la clase `SQLiteConnection`, en cuyo constructor indicaremos detalles como la ruta del fichero y, opcionalmente, la versión de SQLite, y si el fichero se debe crear (`new=True`) o ya existe.
- Empleando un objeto de la clase `SQLiteCommand`, detallaremos cuál es la orden SQL a ejecutar, y la lanzaremos con `ExecuteNonQuery` (en el caso de una orden como "create table", que no devuelve una lista de datos que haya que recorrer).
- `ExecuteNonQuery` devolverá la cantidad de filas afectadas, que para una introducción de un dato, debería ser 1; si nos devuelve un dato menor que uno, nos indicará que no se ha podido guardar correctamente.
- Finalmente cerraremos la conexión con `Close`.

Un fuente de consola que dé estos pasos (y que no compilará hasta que incluyas las referencias, de la forma que veremos en el siguiente apartado), podría ser:

```
// EjemploSQLite1.cs
// Ejemplo de acceso a bases de datos con SQLite (1)
// Introducción a C#, por Nacho Cabanes

using System;
// Es necesario añadir la siguiente DLL a las "referencias" del proyecto
using System.Data.SQLite;

public class EjemploSQLite1
{
    public static void Main()
    {
        Console.WriteLine("Creando la base de datos...");

        // Creamos la conexión a la BD.
        // El Data Source contiene la ruta del archivo de la BD
        SQLiteConnection conexion =
            new SQLiteConnection
                ("Data Source=ejemplo01.sqlite;Version=3;New=True;Compress=True;");
        conexion.Open();

        // Creamos la tabla
        string creacion = "create table personas ("
            + " nombre varchar(20) primary key,direccion varchar(40),"
            + " edad numeric(3));";
        SQLiteCommand cmd = new SQLiteCommand(creacion, conexion);
        cmd.ExecuteNonQuery();

        // E insertamos dos datos
        string insercion = "insert into personas values ('juan', 'su casa', 25);";
```

```

cmd = new SQLiteCommand(insercion, conexion);
int cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

insercion = "insert into personas (nombre, direccion, edad)"
    + " values ('pedro', 'su calle', 23);";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

// Finalmente, cerramos la conexion
conexion.Close();

Console.WriteLine("Creada.");
    }
}

```

### 11.3.3. Compilar un proyecto que incluya SQLite

Una vez creado nuestro proyecto de consola, deberemos comenzar por comprobar para qué versión de la "plataforma punto Net" es, mirando sus "propiedades" desde el menú Proyecto:

Configuración: N/A    Plataforma: N/A

Nombre de ensamblado:     Espacio de nombres predeterminado:

Plataforma de destino:     Tipo de salida:

☒ Generar redirecciones de enlace automáticamente


Objeto de inicio:    

**Recursos**

Especifique cómo se administrarán los recursos de la aplicación:

☒ Icono y manifiesto

Un manifiesto determina una configuración específica para una aplicación. Para insertar un manifiesto personalizado, agréguelo primero al proyecto y, después, selecciónelo en la lista siguiente.

Icono:      

Manifiesto:

☐ Archivo de recursos:    

A continuación, deberemos descargar la versión de System.Data.SQLite adecuada a este proyecto (o la más cercana; en mi caso, mi proyecto es para .Net 4.6.1 y en este momento no hay versión de System.Data.SQLite más allá de 4.6).



**Precompiled Statically-Linked Binaries for 32-bit Windows (.NET Framework 4.6)**

[sqlite-netFx46-static-binary-bundle-Win32-2015-1.0.112.0.zip](#)  
(3.76 MiB)

This binary package features the mixed-mode assembly and contains all the binaries for the x86 version of the System.Data.SQLite 1.0.112.0 (3.30.1) package. The Visual C++ 2015 Update 3 runtime for x86 is statically linked. The .NET Framework 4.6 is required.  
(sha1: 7f24958ba078ba64da01136ff42b0af47040a65a)

[sqlite-netFx46-static-binary-Win32-2015-1.0.112.0.zip](#)  
(3.97 MiB)

This binary package contains all the binaries for the x86 version of the System.Data.SQLite 1.0.112.0 (3.30.1) package. The Visual C++ 2015 Update 3 runtime for x86 is statically linked. The .NET Framework 4.6 is required.  
(sha1: eb1c187862b03411e63aaa3699820fdec93ab42b)

**Precompiled Statically-Linked Binaries for 64-bit Windows (.NET Framework 4.6)**

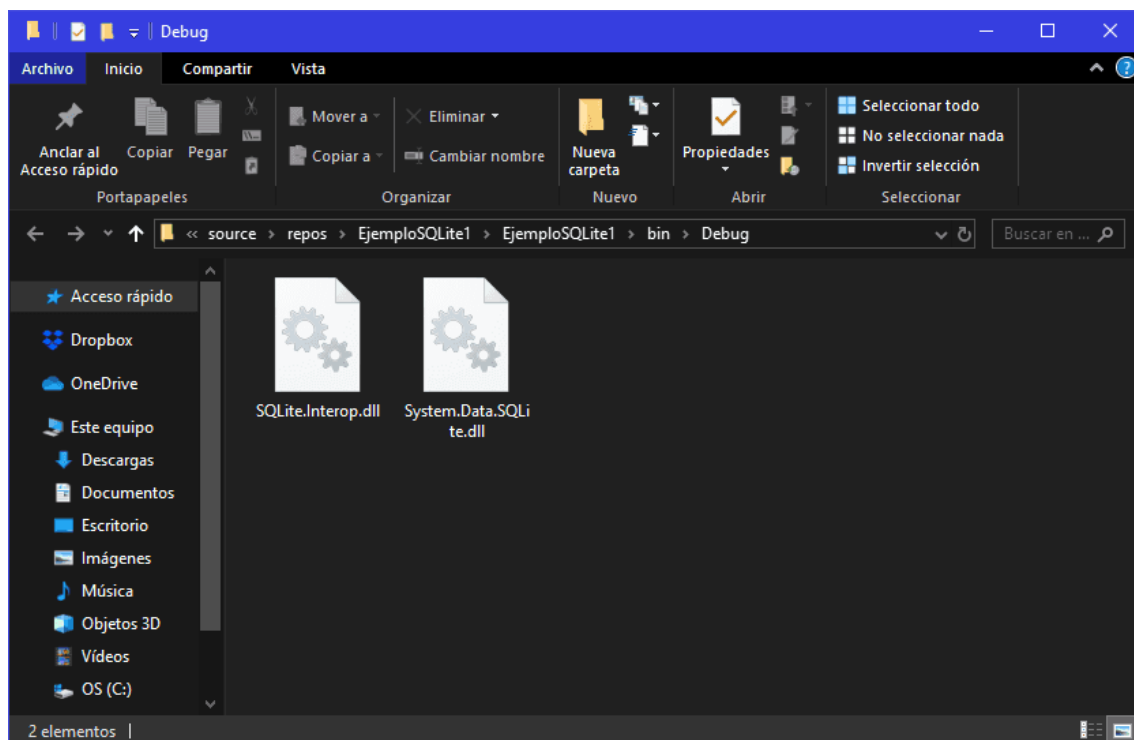
[sqlite-netFx46-static-binary-bundle-x64-2015-1.0.112.0.zip](#)  
(3.78 MiB)

This binary package features the mixed-mode assembly and contains all the binaries for the x64 version of the System.Data.SQLite 1.0.112.0 (3.30.1) package. The Visual C++ 2015 Update 3 runtime for x64 is statically linked. The .NET Framework 4.6 is required.  
(sha1: bc5782439bdd885fe2b5a0a3e03826ac52261318)

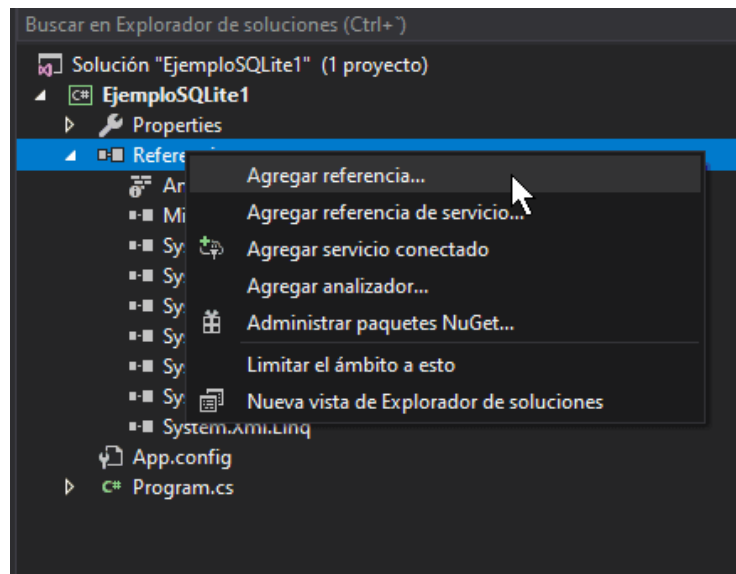
[sqlite-netFx46-static-binary-x64-2015-1.0.112.0.zip](#)  
(4.10 MiB)

This binary package contains all the binaries for the x64 version of the System.Data.SQLite 1.0.112.0 (3.30.1) package. The Visual C++ 2015 Update 3 runtime for x64 is statically linked. The .NET Framework 4.6 is required.  
(sha1: 01db81071c3399f6d86a306538c498097a82d561)

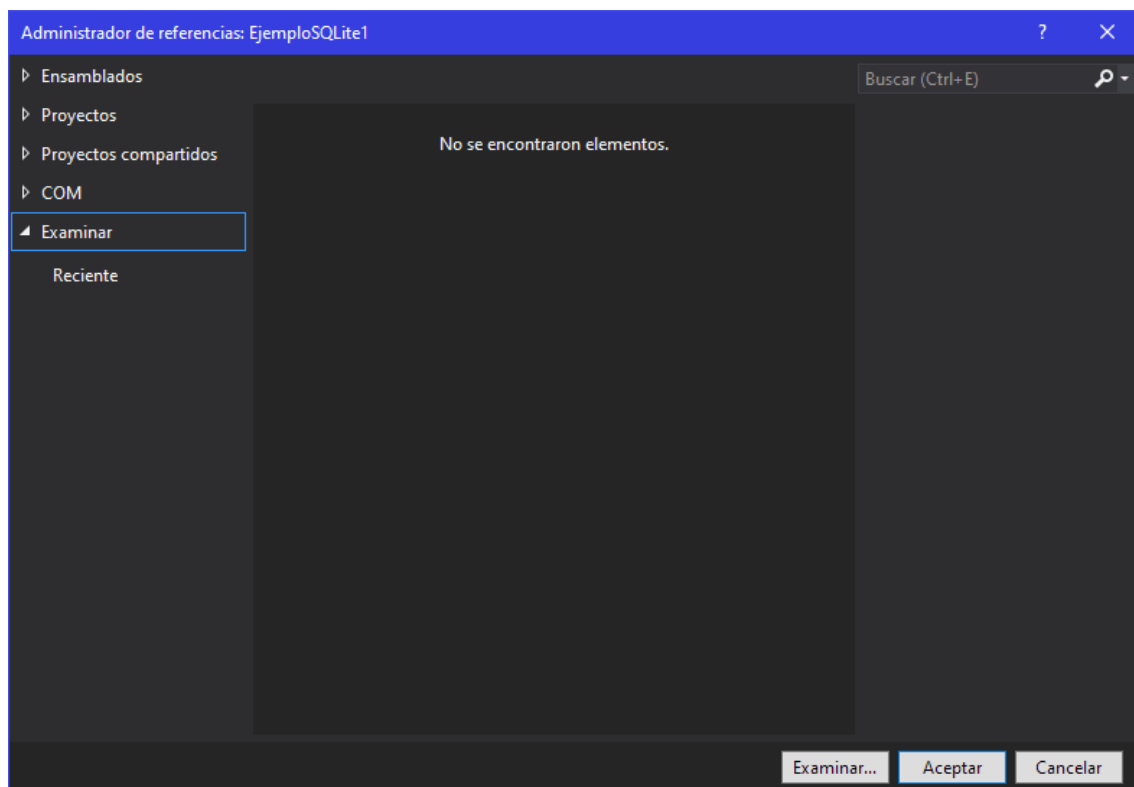
Por ejemplo, yo he usado el fichero "sqlite-netFx46-binary-x64-2015-1.0.112.0.zip", que es la versión de 64 bits para .Net 4.6. De ese fichero ZIP, habrá que extraer "System.Data.SQLite.dll" y "SQLite.Interop.dll", y copiarlos en la carpeta bin/debug de nuestro proyecto (en mi caso, C:\Users\Nacho\source\repos\EjemploSQLite1)



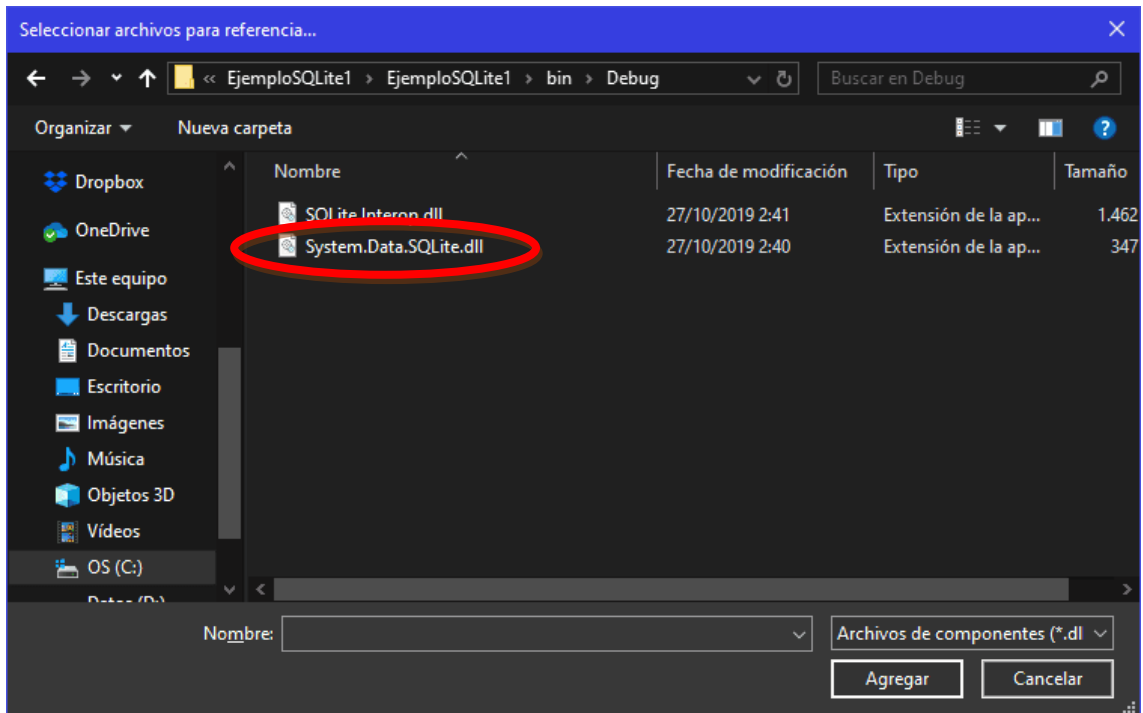
Finalmente, deberemos añadir el fichero "System.Data.SQLite.dll" a las "referencias" del proyecto, desde el panel de Explorador de Soluciones:



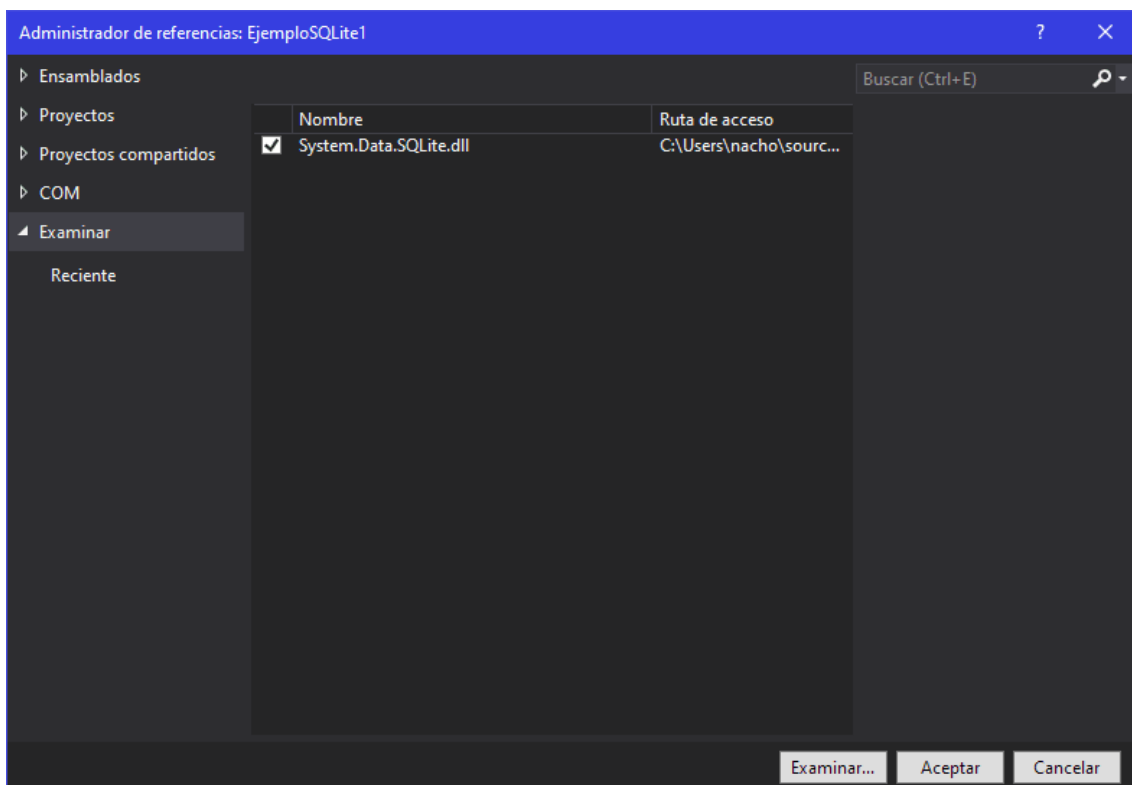
Como no es una referencia propia del sistema, sino un fichero que hemos preparado nosotros, deberemos buscarlo con la opción de "Examinar":



Y seleccionar (solamente) el fichero "System.Data.SQLite.dll" de la carpeta bin/debug de nuestro proyecto:

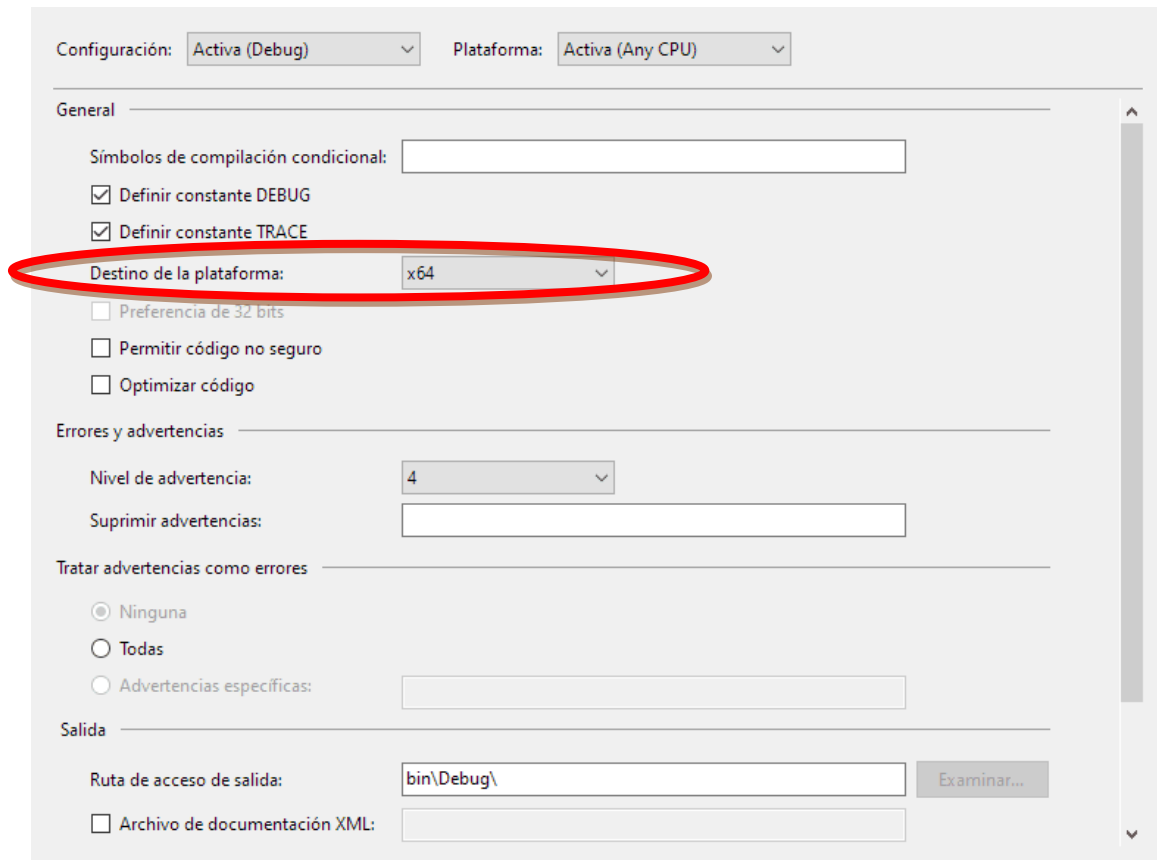


De modo que nuestra lista de referencias debería quedar así:



Y entonces ya podemos lanzar el proyecto. Si obtenemos una excepción "Bad image format exception" será porque estamos usando una DLL de 64 bits con un programa de 32 bits o viceversa. Lo podríamos solucionar cambiando las

propiedades del proyecto, en la pestaña "Compilación", indicando que la plataforma de destino no sea "Any CPU" sino "x64":



Con esos pasos, debería aparecer un fichero llamado "ejemplo01.sqlite" en la carpeta bin/debug del proyecto, y podríamos permitir que **cualquier otra persona** utilizase nuestro programa en su ordenador simplemente entregándole el contenido de dicha carpeta (nuestro ejecutable, los dos ficheros DLL y, según el caso, el fichero de datos).

### @@11.3.4. Obtener datos

Para **mostrar los datos**, el primer y último paso que debería dar nuestro programa serían casi iguales, pero no los pasos intermedios, porque deberemos preparar un "lector" e ir pidiendo los datos de uno en uno:

- Crear una conexión a la base de datos, indicando en este caso que el fichero ya existe (new=false).
- Con un objeto de la clase SQLiteCommand detallaremos cuál es la orden SQL a ejecutar, y la lanzaremos con ExecuteReader.

- Con "Read", leeremos cada dato (devuelve un bool que indica si se ha conseguido leer correctamente), y accederemos a los campos de cada dato como parte de un array: dato[0] será el primer campo, dato[1] será el segundo y así sucesivamente.
- Finalmente cerraremos la conexión con Close:

Vamos a crear un nuevo proyecto y seguiremos pasos similares los del proyecto anterior, excepto por un par de cambios:

- Copiaremos ambos ficheros DLL en su carpeta bin/debug
- Copiaremos a esa carpeta también el fichero de datos (ejemplo1.sqlite) que habíamos generado con el proyecto anterior.
- Añadiremos System.Data.SQLite.dll a las referencias del proyecto, pero, como buena costumbre y para evitar errores difíciles de rastrear, asegurándonos de que lo tomamos de la carpeta bin/debug de este proyecto, no del proyecto anterior.
- En caso de ser necesario, forzaremos a que nuestro proyecto sea de 64 bits.

```
// EjemploSQLite2.cs
// Ejemplo de acceso a bases de datos con SQLite (2)
// Introducción a C#, por Nacho Cabanes

using System;
// Es necesario añadir la siguiente DLL a las "referencias" del proyecto
using System.Data.SQLite;

public class EjemploSQLite2
{
    public static void Main()
    {
        // Creamos la conexión a la BD
        // El Data Source contiene la ruta del archivo de la BD
        SQLiteConnection conexion =
            new SQLiteConnection
                ("Data Source=ejemplo01.sqlite;Version=3;New=False;Compress=True;");
        conexion.Open();

        // Lanzamos la consulta y preparamos la estructura para leer datos
        string consulta = "select * from personas";
        SQLiteCommand cmd = new SQLiteCommand(consulta, conexion);
        SQLiteDataReader datos = cmd.ExecuteReader();

        // Leemos los datos de forma repetitiva
        while (datos.Read())
        {
            string nombre = Convert.ToString(datos[0]);
            int edad = Convert.ToInt32(datos[2]);
            // Y los mostramos
            System.Console.WriteLine("Nombre: {0}, edad: {1}",
                nombre, edad);
        }

        // Finalmente, cerramos la conexión
        conexion.Close();
    }
}
```

```
}  
}
```

Que mostraría:

Nombre: juan, edad: 25  
Nombre: pedro, edad: 23

Nota: en este segundo ejemplo, mostrábamos dato[0] (el primer dato, el nombre) y dato [2] (el tercer dato, la edad) pero no el dato intermedio, dato[1] (la dirección).

### Ejercicios propuestos:

**(11.3.1)** Crea una versión de la base de datos de ficheros (ejemplo 04\_06a) que realmente guarde en una base de datos (de SQLite) los datos que maneja. Deberá guardar todos antes de terminar cada sesión de uso, y volverlos a cargar al comienzo de la siguiente.

**(11.3.2)** Crea la base de datos "biblioteca" que habías planificado anteriormente (en el ejercicio 11.2.1.1), Debe permitir añadir libros, ver todos los libros existentes y buscar libros que contengan un cierto texto en el título o en el autor.

## 11.4. Un poco más de SQL: varias tablas

### 11.4.1. La necesidad de varias tablas

No vamos a ver conceptos de diseño de base de datos, pero aun así, conviene al menos tener presente que puede haber varios motivos por los que nos interese trabajar con más de una tabla.

Por una parte, podemos tener bloques de información claramente distintos. Por ejemplo, en una base de datos que guarde la información de una empresa, tendremos datos como los artículos que distribuimos y los clientes que nos los compren, y estos dos bloques de información son tan distintos que no deberían guardarse en una misma tabla.

Por otra parte, habrá ocasiones en que veamos que los datos, a pesar de que se podrían clasificar dentro de un mismo "bloque de información" (tabla), serían redundantes: existiría gran cantidad de datos repetitivos, y esto puede dar lugar a dos problemas:

- Espacio desperdiciado.

- Posibilidad de errores al introducir los datos, lo que daría lugar a inconsistencias:

Veamos un ejemplo:

```
+-----+-----+-----+
| nombre | direccion | ciudad  |
+-----+-----+-----+
| juan   | su casa   | alicante |
| alberto| calle uno | alicante |
| pedro  | su calle  | alicantw |
+-----+-----+-----+
```

En primer lugar, si en vez de repetir "alicante" en cada una de esas fichas (registros), utilizásemos un código de ciudad, por ejemplo "a", gastaríamos menos espacio (en este ejemplo, 7 bytes menos en cada ficha).

Por otra parte, y más importante todavía que el espacio desperdiciado: hemos introducido mal uno de los datos: en la tercera ficha no hemos indicado "alicante", sino "allicantw", de modo que si hacemos consultas sobre personas de Alicante, la última de ellas no aparecería. Al teclear menos, es también más difícil cometer este tipo de errores.

A cambio, necesitaremos una segunda tabla, en la que guardemos los códigos de las ciudades, y el nombre al que corresponden (por ejemplo: si códigoDeCiudad = "a", la ciudad es "alicante").

### 11.4.2. Las claves primarias

Como ya habíamos adelantado, generalmente será necesario tener algún dato que nos permita distinguir de forma clara los datos que tenemos almacenados. Por ejemplo, el nombre de una persona no es único: pueden aparecer en nuestra base de datos varios usuarios llamados "Juan López". Si son nuestros clientes, debemos saber cuál es cuál, para no cobrar a uno de ellos un dinero que corresponde a otro. Eso se suele solucionar guardando algún dato adicional que sí sea único para cada cliente, como puede ser el Documento Nacional de Identidad, o el Pasaporte. Si no hay ningún dato claro que nos sirva, en ocasiones añadiremos un "código de cliente", inventado por nosotros, o algo similar.

Se llama "claves candidatas" a estos datos (porque quizá haya uno o quizá varios) que puede servir para distinguir claramente unas "fichas" (registros) de otras.

Llamaremos "clave primaria" a la clave candidata que escojamos en un caso concreto, para una cierta tabla (por ejemplo, el "código de cliente").

### 11.4.3. Enlazar varias tablas usando SQL

Vamos a crear la tabla de ciudades, que guardará el nombre de cada una de ellas y su código. Este código será el que actúe como "clave primaria", para distinguir otra ciudad. Por ejemplo, hay una ciudad llamado "Toledo" en España, pero también otra en Argentina, otra en Uruguay, dos en Colombia, una en Ohio (Estados Unidos)... el nombre claramente no es único, así que podríamos usar códigos como "te" para Toledo de España, "ta" para Toledo de Argentina y así sucesivamente.

La forma de crear la tabla con esos dos campos y con esa clave primaria sería:

```
create table ciudades (
  codigo varchar(3) primary key,
  nombre varchar(30)
);
```

O bien, un segundo formato alternativo, que será el que utilizaríamos si la clave primaria estuviera formada por más de un campo, sería:

```
create table ciudades (
  codigo varchar(3),
  nombre varchar(30),
  primary key (codigo)
);
```

Mientras que la tabla de personas sería casi igual al ejemplo anterior, pero añadiendo un nuevo dato: el código de la ciudad

```
create table personas (
  nombre varchar(20) primary key,
  direccion varchar(40),
  edad numeric(3),
  codciudad varchar(3)
);
```

Para introducir datos, el hecho de que exista una clave primaria no supone ningún cambio, salvo por el hecho de que no se nos permitiría introducir dos ciudades con el mismo código (según la plataforma que estemos empleando, puede ocurrir que obtengamos un valor 0 como resultado de la orden de inserción, que nos avisaría de que se han guardado 0 datos -ninguno-, o incluso puede saltar una



excepción que interrumpa el programa, por lo que deberemos comprobar si es el caso e incluir los try-catch pertinentes):

```
insert into ciudades values ('a', 'alicante');
insert into ciudades values ('b', 'barcelona');
insert into ciudades values ('m', 'madrid');

insert into personas values ('juan', 'su casa', 25, 'a');
insert into personas values ('pedro', 'su calle', 23, 'm');
insert into personas values ('alberto', 'calle uno', 22, 'b');
```

Cuando queremos mostrar datos de varias tablas a la vez, deberemos hacer unos pequeños cambios en las órdenes "select" que hemos visto:

- En primer lugar, indicaremos varios nombres después de "FROM" (los de cada una de las tablas que necesitemos).
- Además, puede ocurrir que tengamos campos con el mismo nombre en distintas tablas (por ejemplo, el nombre de una persona y el nombre de una ciudad), y en ese caso deberemos escribir el nombre de la tabla antes del nombre del campo.

Por eso, una consulta básica sería algo parecido (sólo parecido) a:

```
select personas.nombre, direccion, ciudades.nombre from personas, ciudades;
```

Pero esto todavía tiene problemas: estamos combinando TODOS los datos de la tabla de personas con TODOS los datos de la tabla de ciudades, de modo que obtenemos  $3 \times 3 = 9$  resultados:

nombre	direccion	nombre
juan	su casa	alicante
pedro	su calle	alicante
alberto	calle uno	alicante
juan	su casa	barcelona
pedro	su calle	barcelona
alberto	calle uno	barcelona
juan	su casa	madrid
pedro	su calle	madrid
alberto	calle uno	madrid

Pero esos datos no son reales: si "juan" vive en la ciudad de código "a", sólo debería mostrarse junto al nombre "alicante". Nos falta indicar esa condición: "el

código de ciudad que aparece en la persona debe ser el mismo que el código que aparece en la ciudad". La forma más sencilla (pero no la única) de conseguirlo es:

```
select personas.nombre, direccion, ciudades.nombre
from personas, ciudades
where personas.codciudad = ciudades.codigo;
```

Ésta será la forma en que trabajaremos normalmente. El resultado de esta consulta sería:

```
+-----+-----+-----+
| nombre | direccion | nombre  |
+-----+-----+-----+
| juan   | su casa   | alicante |
| alberto | calle uno | barcelona |
| pedro  | su calle  | madrid   |
+-----+-----+-----+
```

Ese sí es el resultado correcto. Cualquier otra consulta que implique las dos tablas deberá comprobar que los dos códigos coinciden. Por ejemplo, para ver qué personas viven en la ciudad llamada "madrid", haríamos:

```
select personas.nombre, direccion, edad
from personas, ciudades
where ciudades.nombre='madrid'
and personas.codciudad = ciudades.codigo;
```

```
+-----+-----+-----+
| nombre | direccion | edad |
+-----+-----+-----+
| pedro  | su calle  | 23 |
+-----+-----+-----+
```

El orden de las condiciones, por supuesto, no es importante de cara al resultado, por lo que también podríamos haber escrito:

```
select personas.nombre, direccion, edad
from personas, ciudades
where personas.codciudad = ciudades.codigo
and ciudades.nombre='madrid';
```

Y para saber las personas que hay en ciudades que comiencen con la letra "b", usaríamos "like":

```
select personas.nombre, direccion, ciudades.nombre
```

```
from personas, ciudades
where ciudades.nombre like 'b%'
and personas.codciudad = ciudades.codigo;
```

```
+-----+-----+-----+
| nombre | direccion | nombre |
+-----+-----+-----+
| alberto | calle uno | barcelona |
+-----+-----+-----+
```

Si en nuestra tabla puede haber algún dato que se repita, como la dirección, podemos pedir un listado sin duplicados, usando la palabra "distinct":

```
select distinct direccion from personas;
```

### Ejercicios propuestos:

**(11.4.3.1)** Anota cómo sería una nueva versión de la base de datos "biblioteca" en la que haya una tabla para los libros y otra diferente para los autores. Supondremos que cada libro tiene sólo un autor, de modo que en la tabla "libro" aparecería el "código de autor", para enlazar ambas tablas.

## 11.4.4. Varias tablas con SQLite desde C#

Vamos a crear un fuente de C# simple pero repetitivo, que ponga a prueba los ejemplos anteriores. Como antes, necesitaremos incluir los dos ficheros DLL y la referencia (pero no el fichero de datos, vamos a crear datos nuevos), así como quizá indicar que se trata de un proyecto de 64 bits.

```
// EjemploSQLite3.cs
// Ejemplo de acceso a bases de datos con SQLite (3)
// Introducción a C#, por Nacho Cabanes

using System;
// Es necesario añadir la siguiente DLL a las "referencias" del proyecto
using System.Data.SQLite;

public class EjemploSQLite3
{
    public static void Crear()
    {
        Console.WriteLine("Creando la base de datos...");

        // Creamos la conexion a la BD.
        // El Data Source contiene la ruta del archivo de la BD
        SQLiteConnection conexion =
            new SQLiteConnection
                ("Data Source=ejemplo02.sqlite;Version=3;New=True;Compress=True;");
        conexion.Open();

        // Creamos las tablas
        Console.WriteLine(" Creando la tabla de ciudades");
```

```

string creacion = "create table ciudades ( "
    + " codigo varchar(3), nombre varchar(30),"
    + " primary key (codigo));";
SQLiteCommand cmd = new SQLiteCommand(creacion, conexion);
cmd.ExecuteNonQuery();

Console.WriteLine(" Creando la tabla de personas");
creacion = "create table personas ( "
    + " nombre varchar(20), direccion varchar(40),"
    + " edad numeric(3), codciudad varchar(3), "
    + " primary key (nombre));";
cmd = new SQLiteCommand(creacion, conexion);
cmd.ExecuteNonQuery();

// E insertamos datos
Console.WriteLine(" Introduciendo ciudades");
string insercion = "insert into ciudades values "
    + "('a', 'alicante');";
cmd = new SQLiteCommand(insercion, conexion);
int cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

// Lo siguiente es repetitivo, sería deseable
// crear una función auxiliar
insercion = "insert into ciudades values "
    + "('b', 'barcelona');";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

insercion = "insert into ciudades values "
    + "('m', 'madrid');";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

Console.WriteLine(" Introduciendo personas");
insercion = "insert into personas values "
    + "('juan', 'su casa', 25, 'a');";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

insercion = "insert into personas values "
    + "('pedro', 'su calle', 23, 'm');";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

insercion = "insert into personas values "
    + "('alberto', 'calle uno', 22, 'b');";
cmd = new SQLiteCommand(insercion, conexion);
cantidad = cmd.ExecuteNonQuery();
if (cantidad < 1)
    Console.WriteLine("No se ha podido insertar");

// Finalmente, cerramos la conexion
conexion.Close();

Console.WriteLine("Base de datos creada.");
}

```

```

public static void Mostrar()
{
    // Creamos la conexion a la BD
    // El Data Source contiene la ruta del archivo de la BD
    SQLiteConnection conexion =
        new SQLiteConnection
            ("Data Source=ejemplo02.sqlite;Version=3;New=False;Compress=True;");
    conexion.Open();

    // Lanzamos la consulta y preparamos la estructura para leer datos
    string consulta = "select personas.nombre, direccion, ciudades.nombre "
        + "from personas, ciudades where personas.codciudad = ciudades.codigo;";
    SQLiteCommand cmd = new SQLiteCommand(consulta, conexion);
    SQLiteDataReader datos = cmd.ExecuteReader();

    Console.WriteLine("Datos:");
    // Leemos los datos de forma repetitiva
    while (datos.Read())
    {
        // Y los mostramos
        Console.WriteLine(" {0} - {1} - {2}",
            Convert.ToString(datos[0]), Convert.ToString(datos[1]),
            Convert.ToString(datos[2]));
    }

    // Finalmente, cerramos la conexion
    conexion.Close();
}

public static void Main()
{
    Crear();
    Mostrar();
}

```

Su resultado sería:

Creando la base de datos...

Creando la tabla de ciudades

Creando la tabla de personas

Introduciendo ciudades

Introduciendo personas

Base de datos creada.

Datos:

juan - su casa - alicante

pedro - su calle - madrid

alberto - calle uno - barcelona

**Nota:** el código anterior tiene fragmentos repetitivos, que se deberían modularizar en un programa real, por ejemplo creando funciones auxiliares.

Como ya habíamos anticipado, en SQLite, cierto tipo de errores darán lugar a que se lance una **excepción** y se interrumpa el programa. Ocurrirá al intentar crear una tabla que ya existe y también al intentar guardar un dato cuya clave primaria

ya se ha utilizado. Por ejemplo, si el nombre fuera clave primaria de la tabla "personas", sería más fiable plantear la inserción así:

```
Console.WriteLine("Insertando datos...");
string insercion;
int cantidad;
try
{
    insercion = "insert into personas values "
                + "('juan', 'su casa', 25, 'a');";
    cmd = new SQLiteCommand(insercion, conexion);
    cantidad = cmd.ExecuteNonQuery();
    if (cantidad == 1)
        Console.WriteLine("Dato de Juan insertado ");
}
catch (Exception)
{
    Console.WriteLine("Dato de Juan no insertado (¿clave duplicada?)");
}
```

### Ejercicios propuestos:

**(11.4.4.1)** Mejora el ejercicio 11.3.1 para que, además del nombre del fichero y su tamaño, guarde una categoría (por ejemplo, "utilidad" o "vídeo"). Estas categorías estarán almacenadas en una segunda tabla.

**(11.4.4.2)** Implementa la nueva versión de la base de datos "biblioteca", con una tabla para libros y otra para autores. Guarda varios datos prefijados de ejemplo.

## 11.5. Borrado y modificación de datos

Podemos **borrar** los datos que cumplen una cierta condición. La orden es "delete from", y con "where" indicamos las condiciones que se deben cumplir, de forma similar a como hacíamos en la orden "select":

```
delete from personas where nombre = 'juan';
```

Esto borraría todas las personas llamadas "juan" que estén almacenadas en la tabla "personas".

**Cuidado:** si no se indica la parte de "where", no se borrarían los datos que cumplen una condición, sino TODOS los datos que existan en esa tabla.

Para **modificar** datos de una tabla, el formato habitual es "update tabla set campo=nuevoValor where condicion".

Por ejemplo, si hemos escrito "Alberto" en minúsculas ("alberto"), lo podríamos corregir con:

```
update personas set nombre = 'Alberto' where nombre = 'alberto';
```

Y si queremos corregir todas las edades para sumarles un año se haría con

```
update personas set edad = edad+1;
```

(al igual que habíamos visto para "select" y para "delete", si no indicamos la parte del "where", los cambios se aplicarán a todos los registros de la tabla).

### Ejercicios propuestos:

**(11.5.1)** Crea una versión del ejercicio 11.3.1 que no guarde todos los datos al salir, sino que actualice con cada nueva modificación: inserte los nuevos datos inmediatamente, permita borrar un registro (reflejando los cambios inmediatamente) y modificar los datos de un registro (ídem).

**(11.5.2)** A partir de la base de datos "biblioteca" con dos tablas, crea un programa de gestión que permita añadir autores, añadir libros, ver todos los libros (incluyendo el nombre del autor), buscar libros por un fragmento del título, buscar autores por un fragmento del nombre, modificar autores, modificar libros.

## 11.6. Operaciones matemáticas con los datos

Desde SQL podemos realizar operaciones a partir de los datos antes de mostrarlos. Por ejemplo, podemos mostrar cuál era la edad de una persona hace un año, con

```
select edad-1 from personas;
```

Y podemos dar un "nombre ficticio" (un "**alias**") a esos campos calculados:

```
select edad-1 as edadAnterior from personas;
```

Los operadores matemáticos que podemos emplear son los habituales en cualquier lenguaje de programación, ligeramente ampliados: + (suma), - (resta y negación), \* (multiplicación), / (división). La división calcula el resultado con decimales; si queremos trabajar con números enteros, también tenemos los operadores DIV (división entera) y MOD (resto de la división):

```
select 5/2, 5 div 2, 5 mod 2;
```

Darí­a como resultado

```
+-----+-----+-----+
| 5/2    | 5 div 2 | 5 mod 2 |
+-----+-----+-----+
| 2.5000 |        2 |        1 |
+-----+-----+-----+
```

Tambi3n podemos aplicar ciertas funciones matemáticas **a todo un conjunto de datos** de una tabla (esto se conoce como "funciones de agregaci3n"). Por ejemplo, podemos saber cuál es la edad mäs baja de entre las personas que tenemos en nuestra base de datos, haríamos:

```
select min(edad) from personas;
```

Las "funciones de agregaci3n" mäs habituales son:

- min = m3nimo valor
- max = m3ximo valor
- sum = suma de los valores
- avg = media de los valores
- count = cantidad de valores

La forma mäs com3n de usar "count" ser3 pidiendo con "count(\*)" que se nos muestren todos los datos que cumplen una condici3n. Por ejemplo, podríamos saber cuántas personas tienen una direcci3n que comience por la letra "s", así:

```
select count(*) from personas where direccion like 's%';
```

### Ejercicios propuestos:

**(11.6.1)** Añade al gestor de "biblioteca" esta funcionalidad: cuando se busquen libros que contengan un cierto texto, se mostrar3 antes la cantidad de datos que cumplen esa condici3n.

## 11.7. Grupos

Puede ocurrir que no nos interese un 3nico valor agrupado para todos los datos (el total, la media, la cantidad de datos), sino el resultado para un grupo de datos. Por ejemplo: saber no s3lo la cantidad de clientes que hay registrados en nuestra base de datos, sino tambi3n la cantidad de clientes que viven en cada ciudad.



La forma de obtener subtotales es creando grupos con la orden "group by", y entonces pidiendo una valor agrupado (count, sum, avg, ...) para cada uno de esos grupos. Por ejemplo, en nuestra tabla "personas", podríamos saber cuántas personas aparecen de cada edad, con:

```
select count(*), edad from personas group by edad;
```

que daría como resultado

count(*)	edad
1	22
1	23
1	25

Pero podemos llegar más allá: podemos no trabajar con todos los grupos posibles, sino sólo con los que cumplen alguna condición.

La condición que se aplica a los grupos no se indica con "where", sino con "having" (que se podría traducir como "los que tengan..."). Un ejemplo:

```
select count(*), edad from personas group by edad having edad > 24;
```

que mostraría

count(*)	edad
1	25

En el lenguaje SQL existe mucho más que lo que hemos visto aquí, pero para nuestro uso desde C# y SQLite debería ser suficiente.

### Ejercicios propuestos:

**(11.7.1)** Crea una versión del ejercicio 11.4.1 que permita saber cuántos ficheros hay pertenecientes a cada categoría.

**(11.7.2)** Añade al gestor de "biblioteca" la posibilidad de saber los nombre de los autores de los que tenemos más de un libro.

## 11.8. Un ejemplo completo con C# y SQLite

Vamos a crear un pequeño ejemplo que, para una única tabla, permita añadir datos, mostrar todos ellos o buscar los que cumplan una cierta condición:

```
// AgendaSQLite.cs
// Ejemplo de acceso a bases de datos con SQLite: agenda
// Introducción a C#, por Nacho Cabanes

using System;
using System.IO; // Para File.Exists

// Es necesario añadir la siguiente DLL a las "referencias" del proyecto
using System.Data.SQLite;

public class AgendaSQLite
{
    static SQLiteConnection conexion;

    // Constructor
    public AgendaSQLite()
    {
        AbrirBD();
    }

    // Abre la base de datos, o la crea si no existe
    private void AbrirBD()
    {
        if (!File.Exists("agenda.sqlite"))
        {
            // Si no existe, creamos la base de datos
            conexion = new SQLiteConnection (
                "Data
Source=agenda.sqlite;Version=3;New=True;Compress=True;");
            conexion.Open();

            // Y creamos la tabla
            string creacion = "CREATE TABLE persona "
                + "(nombre VARCHAR(30), direccion VARCHAR(40), "
                + " edad INT );";
            SQLiteCommand cmd = new SQLiteCommand(creacion, conexion);
            cmd.ExecuteNonQuery();
        }
        else
        {
            // Si ya existe, abrimos
            conexion = new SQLiteConnection(
                "Data
Source=agenda.sqlite;Version=3;New=False;Compress=True;");
            conexion.Open();
        }
    }

    public bool InsertarDatos(string nombre, string direccion, int edad)
    {
        string insercion; // Orden de insercion, en SQL
        SQLiteCommand cmd; // Comando de SQLite
        int cantidad; // Resultado: cantidad de datos
```

```

try
{
    insercion = "INSERT INTO persona " +
        "VALUES ('"+nombre+"', '"+direccion+"', '"+edad+"');";
    cmd = new SQLiteCommand(insercion, conexion);
    cantidad = cmd.ExecuteNonQuery();
    if (cantidad < 1)
        return false; // Si no se ha podido insertar
}
catch (Exception e)
{
    return false; // Si no se ha podido insertar (codigo repetido?)
}
return true; // Si todo ha ido bien, devolvemos true
}

// Leer todos los datos y devolverlos en un string de varias líneas
public string LeerTodosDatos()
{
    // Lanzamos la consulta y preparamos la estructura para leer datos
    string consulta = "select * from persona";
    string resultado = "";
    SQLiteCommand cmd = new SQLiteCommand(consulta, conexion);
    SQLiteDataReader datos = cmd.ExecuteReader();
    // Leemos los datos de forma repetitiva
    while (datos.Read())
    {
        resultado += Convert.ToString(datos[0]) + " - "
            + Convert.ToString(datos[1]) + " - "
            + Convert.ToInt32(datos[2]) + "\n";
    }
    return resultado;
}

// Leer todos los datos y devolverlos en un string de varias líneas
public string LeerBusqueda(string texto)
{
    // Lanzamos la consulta y preparamos la estructura para leer datos
    string consulta = "select * from persona where nombre like '%"
        + texto + "%' or direccion like '%" + texto + "%'";
    string resultado = "";
    SQLiteCommand cmd = new SQLiteCommand(consulta, conexion);
    SQLiteDataReader datos = cmd.ExecuteReader();
    // Leemos los datos de forma repetitiva
    while (datos.Read())
    {
        resultado += Convert.ToString(datos[0]) + " - "
            + Convert.ToString(datos[1]) + " - "
            + Convert.ToInt32(datos[2]) + "\n";
    }
    return resultado;
}

// Destructor: cierra la conexión a la base de datos
// Según la versión de Visual Studio y de SQLite, puede ser innecesario
// Si provoca una excepción, elimínalo
~AgendaSQLite()
{
    conexion.Close();
}

```

```

    }

}

// -----

public class pruebaSQLite01
{
    public static void Main()
    {
        AgendaSQLite agenda = new AgendaSQLite();
        string opcion;
        do
        {
            Console.WriteLine("Escoja una opción...");
            Console.WriteLine("1.- Añadir");
            Console.WriteLine("2.- Ver todos");
            Console.WriteLine("3.- Buscar");
            Console.WriteLine("0.- Salir");

            opcion = Console.ReadLine();

            switch (opcion)
            {
                case "1":
                    Console.Write("Nombre? ");
                    string n = Console.ReadLine();
                    Console.Write("Dirección? ");
                    string d = Console.ReadLine();
                    Console.Write("Edad? ");
                    int e = Convert.ToInt32(Console.ReadLine());
                    agenda.InsertarDatos(n, d, e);
                    break;
                case "2":
                    Console.WriteLine(agenda.LeerTodosDatos());
                    break;
                case "3":
                    Console.Write("Texto a buscar? ");
                    string txt = Console.ReadLine();
                    Console.WriteLine(agenda.LeerBusqueda(txt));
                    break;
            }
        } while (opcion != "0");
    }
}

```

### Ejercicios propuestos:

**(11.8.1)** Amplía este ejemplo (AgendaSQLite), para que se pueda borrar un dato a partir de su nombre (que debe coincidir exactamente).

**(11.8.2)** Amplía el ejemplo 11.8.1, para que se pueda modificar un registro.

**(11.8.3)** Amplía el ejemplo 11.8.2, para que permita exportar los datos a un fichero de texto (que contenga el nombre, la dirección y la edad correspondientes a cada registro en líneas separados), o bien se pueda importar datos desde un fichero de texto (añadiendo al final de los existentes).

**(11.8.4)** Amplía el ejemplo 11.8.3 con una tabla de ciudades, que se deberá pedir (y mostrar) de forma independiente al resto de la dirección.

## 11.9. ¿Y desde un entorno gráfico?

En la mayoría de los casos, cuando trabajamos desde un entorno "de ventanas" (en un apéndice de este texto verás las nociones básicas sobre cómo crearlos), tendremos a nuestra disposición ciertos componentes visuales que nos permitan "navegar" por los datos de forma muy simple.

Por ejemplo, en Windows Forms existe un "DataGridView", que nos permite recorrer los datos en una vista de tabla, y poder hacer modificaciones sobre ellos (lo comentaremos brevemente en dicho apéndice al final del texto):

