

**Fundamentos de bases de datos  
relacionales © Ediciones Roble S. L.**

# Índice

<b>Fundamentos de bases de datos relacionales</b>	<b>3</b>
I. Introducción	3
II. Objetivos	3
III. El modelo relacional	3
3.1. Dominio, atributo e instancia	4
3.2. Claves: conceptos y tipos	5
3.3. Operaciones en el modelo relacional	7
3.4. Integridad en el modelo relacional	8
IV. SQLTESTUDIO	10
V. El lenguaje SQL	15
5.1. Creación de tablas en SQL	16
5.1.1. Tipos de datos	16
5.1.2. Definiciones por defecto	18
5.1.3. Restricciones por columna	18
5.1.4. Restricciones por tabla	19
5.1.5. Los siguientes ejemplos de creación de tablas se realizaron utilizando SQLiteStudio	19
5.1.6. Claves foráneas	23
5.2. Modificación de tablas	25
5.3. Otras operaciones sobre tablas	27
5.4. Índices	28
5.5. Consultas y recuperación de información en SQL	28
5.5.1. Consultas sobre una sola tabla	28
5.5.2. Consultas sobre más de una tabla	33
5.6. Modificación de tablas en SQL	37
5.6.1. Operaciones de actualización	37
5.6.2 Operaciones sobre tablas	40
VI. Resumen	42
VII. Caso práctico	43
<b>Recursos</b>	<b>49</b>
Enlaces de Interés	49
Bibliografía	49
Glosario.	49

# Fundamentos de bases de datos relacionales

## I. Introducción

En las últimas décadas, el mecanismo más utilizado para almacenar la información en los sistemas informáticos ha sido las bases de datos relacionales. Ofrecen un modelo estable, con variedad de herramientas de desarrollo y altos niveles de seguridad. Sin embargo, se podría decir que el elemento clave del éxito de este modelo ha sido que dispone de un lenguaje estándar para poder realizar consultas (SQL). Actualmente muchas fuentes de información se encuentran almacenadas en bases de datos relacionales.

Por otro lado, dentro del fenómeno Big Data, han surgido nuevos modelos de persistencia de datos denominados genéricamente como bases de datos NoSQL que, aunque no siguen el modelo relacional, sí guardan algunas similitudes en cuanto a sus lenguajes de consulta y conceptos usados. Es por ello que el conocimiento de las bases de datos relacionales y de SQL facilita la comprensión y aprendizaje de estos nuevos modelos.

En esta unidad se estudiará, en primer lugar, el modelo relacional que constituye el fundamento teórico en el que se asientan las bases de datos relacionales. Se presentará SQLiteStudio, una aplicación que implementa el lenguaje SQLite, sobre la cual se realizarán los ejercicios de la unidad. A continuación, se introducirán los elementos principales del lenguaje SQL que permiten a un usuario gestionar una base de datos relacional.



Se sugiere realizar los ejercicios a la par del avance de la unidad, lo que permitirá desarrollar las habilidades en SQL y bases de datos, que en la actualidad son necesarias en el área de las tecnologías de la información.

## II. Objetivos

Mostrar los principios fundamentales del modelo relacional en el que se basan las bases de datos relacionales.

Estudiar cómo diseñar, crear y explotar una base de datos relacional usando SQL.

Usar un sistema de gestión de bases de datos relacionales concreto (SQLiteStudio).

Experimentar la utilidad de una base de datos relacional como sistema de almacenamiento de los datos gestionados desde una aplicación informática.

## III. El modelo relacional

El modelo relacional es un mecanismo de representación de la información que se basa en el concepto de relación. Formalmente una relación se constituye por dos elementos:

- **Esquema.** Representa la estructura de la información, indicando qué tipo de información se va a gestionar. Un esquema, a su vez, se compone de un nombre de la relación y de un conjunto de atributos de la relación.
- **Un conjunto de instancias.** Representa la aplicación del esquema en un conjunto de datos o información concreta.

### 3.1. Dominio, atributo e instancia

#### Dominio

Un dominio representa un conjunto de valores de un tipo de datos que son atómicos —no pueden descomponerse más—; por ejemplo: el dominio de los enteros, los reales...

#### Atributo

Un atributo es el nombre que recibe un dominio en el contexto de una relación. Por ejemplo, el atributo nombre de una persona correspondería a una cadena o conjunto de caracteres.

#### Instancia

Finalmente, una instancia de una relación consiste en una tupla de valores concretos del dominio de datos asociado a los atributos de la relación. Por ejemplo, “Juan” es una cadena y representa un valor concreto para el atributo “Nombre”.



Por ejemplo, si se considera representar la información acerca de los empleados de una empresa mediante el modelo relacional, las instancias de la relación son tuplas de valores concretos para los atributos de la relación:

(“Juan”, “Rodríguez Rojo”, 34, “4559999F”, 30000)

(“Pedro”, “Sánchez Sánchez”, 54, “5444545E”, 45000)

(“Isabel”, “Leyva Azul”, 36, “6667733R”, 35000)

(“Jaime”, “García Redondo”, 39, “3456344T”, 39000)

En algunas ocasiones, pueden existir instancias en las que algunos atributos no tomen un valor concreto y se representan con un valor especial denominado valor nulo. El valor nulo representa un valor desconocido. En el ejemplo anterior, si se hubiera considerado el atributo “teléfono”, podría darse el caso de algún empleado que no tenga teléfono y se consignaría el valor nulo.

Algunas características sobre las relaciones y sus tuplas son:

#### Atomicidad

Los valores de los atributos deben ser atómicos, es decir, no se pueden descomponer más.

#### Unicidad de las tuplas

No pueden existir 2 tuplas con los mismos valores, dado que las instancias son un conjunto. En un conjunto solo hay elementos únicos, no repetidos.

#### Tuplas sin orden

Las tuplas no están ordenadas debido a su definición como un conjunto. En un conjunto no existe un orden entre sus elementos.

#### Atributos sin orden

En los atributos de una relación no hay un orden definido, pues se trata de un conjunto. En un conjunto no existe un orden entre sus elementos.

Se denomina grado de una relación al número de atributos que pertenecen a su esquema y cardinalidad al número de tuplas definidas en la relación. En el ejemplo anterior, el grado de la relación es 5 y su cardinalidad 4. (5 atributos y 4 tuplas).

## 3.2. Claves: conceptos y tipos

#### Superclave

Una superclave de una relación es un subconjunto de los atributos del esquema tal que no puede haber dos tuplas de la relación que tengan la misma combinación de valores para los atributos del subconjunto. Así, una superclave permite identificar las tuplas de una relación.

Toda relación tiene al menos una superclave formada por todos los atributos de su esquema —se debe al hecho de que una relación no puede tener tuplas repetidas—.



En el ejemplo anterior serían superclaves: {Nombre, Apellidos, Edad, DNI, Sueldo}, {DNI, Nombre, Apellidos}, {DNI}

**Clave candidata**

Se denomina clave candidata de una relación a una superclave de la relación que cumple que ningún subconjunto propio sea superclave. Es decir, que si se elimina algún atributo de la clave candidata ya no es superclave. Como toda relación tiene al menos una superclave, entonces tiene al menos una clave candidata.



En el ejemplo anterior, la superclave {DNI} es clave candidata. Sin embargo, la superclave {DNI, Nombre, Apellidos} no lo es, pues si se elimina, por ejemplo, el atributo "Apellidos", el conjunto {DNI, Nombre} sigue siendo superclave

**Clave primaria**

Entre todas las claves candidatas de una relación, se elige una de ellas como la clave cuyos valores se utilizarán para identificar las tuplas de una relación. Esta clave recibe el nombre de clave primaria. El resto de claves candidatas no elegidas se las denomina claves alternativas. Toda relación tiene al menos una clave primaria, dado que siempre tiene al menos una clave candidata.

Es posible que una clave candidata o una clave primaria conste de más de un atributo.



Por ejemplo, considérese una relación para representar tipos de tornillos cuyos atributos son marca, ancho y largo, de manera que una misma marca puede tener diferentes tipos de tornillos con el mismo largo y ancho. En este caso, una clave candidata estaría formada por {marca, ancho, largo}

**Clave foránea**

En general, en un contexto real, es necesario gestionar más de una relación y entre las relaciones consideradas existen vínculos o conexiones. Para modelizar estos vínculos, el modelo relacional dispone de las claves foráneas.

Una clave foránea de una relación permite establecer conexiones entre las tuplas de varias relaciones. En este sentido, una clave foránea está formada por el conjunto de atributos de una relación que referencia la clave primaria de otra relación —o incluso de la misma relación—. Dado que las claves foráneas establecen una conexión con la clave primaria que referencian, los valores de una clave foránea deben estar presentes en la clave primaria correspondiente, o bien deben ser valores nulos.



Por ejemplo, considérense las relaciones EMPLEADOS y DESPACHOS que permiten modelizar la información acerca de un empleado de una empresa y sobre el despacho en el que se encuentran los empleados. Para la primera relación se han considerado los atributos: DNI, Nombre, Apellidos, DNIJefe, PlantaDespacho, NumDespacho y para la segunda relación se consideran los atributos: Planta, Número, Plazas. En la relación EMPLEADOS, la clave primaria podría ser {DNI} y en la relación DESPACHOS la clave primaria podría ser {Planta, Número}.

Téngase en cuenta lo siguiente:

El conjunto de atributos {PlantaDespacho, NumDespacho} de la relación EMPLEADOS constituye una clave foránea que se refiere a la clave primaria de la relación DESPACHOS y que indica para cada empleado el despacho donde trabaja.

El atributo DNIJefe de la relación EMPLEADOS es una clave foránea que se refiere a la clave primaria de la misma relación que indica, para cada empleado, quién es su jefe.

El número de atributos de una clave foránea y de la clave primaria a la que referencia deben ser iguales.

Debe ser posible establecer una correspondencia entre los atributos de una clave foránea y los atributos de la clave primaria a la que referencia.

Los dominios de los atributos de la clave foránea deben coincidir con los dominios de los atributos de la clave primaria a la que referencian. En el ejemplo anterior, la clave foránea {PlantaDespacho, NumDespacho} y la clave primaria {Planta, Número} cumplen estas propiedades.

Un atributo de una relación podría formar parte tanto de la clave primaria como de una clave foránea de la relación.

### 3.3. Operaciones en el modelo relacional

Las operaciones del modelo relacional deben permitir manipular la información representada en las relaciones. En este sentido se definen dos tipos de operaciones:

**Actualización**

**Actualización.** Permite modificar la información representada en una relación. Pueden ser de tres tipos:

- Inserción para añadir nuevas tuplas a una relación.
- Borrado para eliminar tuplas de una relación.
- Modificación para alterar los valores de una tupla de la relación.

**Consulta**

**Consulta.** Permite recuperar la información representada en las relaciones y que se encuentra almacenada en las tuplas.

Para implementar estas operaciones se han definidos lenguajes relacionales.

### 3.4. Integridad en el modelo relacional

Para mantener la integridad y la consistencia de la información que se representa en una relación se define un conjunto de reglas de integridad:

**Unicidad de la clave primaria**

Establece que toda clave primaria de una relación no debe tener valores repetidos. Se debe garantizar el cumplimiento de esta regla en todas las inserciones y modificaciones que afecten a atributos que pertenezcan a la clave primaria de la relación.

**Entidad de la clave primaria**

Establece que los atributos de la clave primaria de una relación no pueden tener valores nulos. Se debe garantizar el cumplimiento de esta regla en todas las inserciones y modificaciones que afecten a atributos que pertenezcan a la clave primaria de la relación.

**Integridad referencial**

Establece que todos los valores que toma una clave foránea deben ser nulos o valores que existen en la clave primaria que referencia.



## Operaciones

Se debe tener en cuenta en las siguientes operaciones:

1

Inserciones en una relación que tenga una clave foránea.

2

Modificaciones que afecten a atributos que pertenezcan a la clave foránea de una relación.

3

Borrados en relaciones referenciadas por otras relaciones.

4

Modificaciones que afecten a atributos que pertenezcan a la clave primaria de una relación referenciada por otras relaciones.

## Políticas de actuación

Existen dos políticas de actuación para mantener la **integridad referencial** cuando se realiza alguna de las operaciones anteriormente señaladas:

1

Rechazar cualquier operación que provoque el incumplimiento de alguna de las reglas anteriores.

2

Llevar a cabo la operación y realizar posteriormente aquellas acciones necesarias para que las relaciones mantengan la integridad. Es posible aplicarlo:

- Cuando se borra una tupla que tiene una clave primaria referenciada.
- Cuando se modifican los valores de los atributos de la clave primaria de una tupla que es referenciada.

Para los casos anteriores existen 3 políticas de actuación:

1

Restricción. Consiste en no aceptar la operación de actualización siempre que afecte a una clave primaria referenciada por una clave foránea.

2

Actualización en cascada. Se permite la operación de actualización y se lleva las mismas operaciones sobre las tuplas que las referencian en otras relaciones.

3

Anulación. Se permite la operación de actualización y se ponen valores nulos a los atributos de la clave foránea de las tuplas que la referencian.


### Integridad del dominio


Establece que todos los valores no nulos para un determinado atributo deben ser del dominio declarado para dicho atributo y que los operadores que se pueden aplicar sobre los valores dependen del dominio de estos valores..

## IV. SQLITESTUDIO

Para poner en práctica el lenguaje SQL del siguiente tema, se va a utilizar SQLiteStudio, un sistema de gestión de bases de datos relacionales que puede descargarse en:

SQLite Studio: <https://sqlitestudio.pl/index.rvt?act=download> (figura 4.1.).

 [Download Windows binary](#)  
Version 3.2.1  
(33.2 MB)

 **SQLite Studio**

Navigation: [About](#) [Gallery](#) [Download](#) [Changelog](#) [Forum](#) [Wiki](#) [Bugs & Ideas](#) [The Wall](#) [Contact](#) [Links](#)

**Lastest stable release (3.2.1):**

Distribution	Platform	Size	Version	Link
Windows (portable)	32-bit	27.5MB	3.2.1	<a href="#">SQLiteStudio-3.2.1.zip</a>
Windows (installer)	32-bit	33.3MB	3.2.1	<a href="#">InstallSQLiteStudio-3.2.1.exe</a>
Linux (portable)	64-bit	30.0MB	3.2.1	<a href="#">sqlitestudio-3.2.1.tar.xz</a>
Linux (installer)	64-bit	46.1MB	3.2.1	<a href="#">InstallSQLiteStudio-3.2.1</a>
MacOSX (portable)	64-bit (x86_64)	30.9MB	3.2.1	<a href="#">SQLiteStudio-3.2.1.dmg</a>
MacOSX (installer)	64-bit (x86_64)	25.5MB	3.2.1	<a href="#">InstallSQLiteStudio-3.2.1.dmg</a>
Sources (zip)	Independent	9.8MB	3.2.1	<a href="#">sqlitestudio-3.2.1.zip</a>
Sources (tar.gz)	Independent	9.0MB	3.2.1	<a href="#">sqlitestudio-3.2.1.tar.gz</a>

NOTE:  
Binary distribution can be downloaded either as installer, which will install SQLiteStudio at specified path (for Windows it will also create file associations and create Start menu entries), or as a portable package (files without "installer" in their name), which you just download, decompress and run on spot.

[SHA-256 checksums...](#)  
[All files - click here...](#)  
[Old, unsupported versions \(2.x.x\) - click here...](#)

**News** [RSS](#)

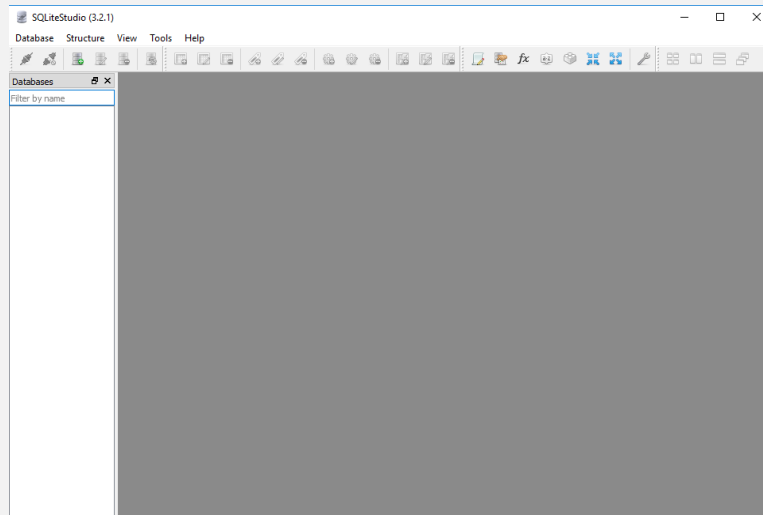
**3.2.1 released!**  
2018-07-27  
Soon after 3.2.0 few major problems appeared, which are addressed in 3.2.1. One was inability to start application under Linux, second was missing Image previewer plugin in binary packages. Both problems were caused by incorrect packaging scripts. There was also one more smaller bug regarding data exporting (see Changelog for more details). If you're upgrading automatically from 3.2.0, after you're done, you need to run the "UpdateSQLiteStudio" from application's folder, pick "Add or remove component" option, on next page expand the tree and mark "Image editor/viewer plugin" and proceed with installation. If you're installing/downloading 3.2.1 from homepage (and not through the auto-update), you will already have this plugin installed.

Figura 4.1. Zona de descarga de SQLiteStudio.

SQLiteStudio es un sistema de gestión de bases de datos relacionales ligero y abierto, basado en SQLite, que implementa un subconjunto del estándar SQL. Su ventaja principal es poder mantener datos de una aplicación de forma sencilla y segura, todo en un archivo pequeño que puede ser leído por aplicaciones externas. Se puede consultar más información sobre SQLite en la siguiente dirección: <http://www.sqlite.org/lang.html>



Descargue la versión portable, desempaquete en la ruta preferida, pulse en el archivo SQLiteStudio.exe y aparecerá la interface principal (figura 4.2.):



**Figura 4.2.** Interface principal de SQLiteStudio.

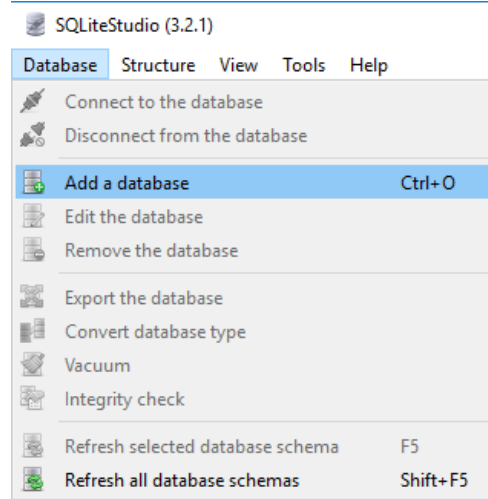
Para ilustrar algunas de las funcionalidades de SQLiteStudio, se va a considerar una base de datos con el fin de realizar los ejercicios del siguiente tema. En primer lugar, hay que crear la base de datos:

**1**

Se pulsa sobre "Database".

## 2

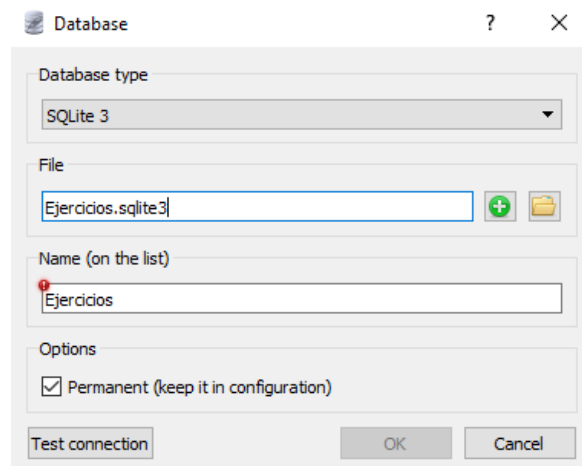
En el desplegable que aparece se selecciona “Add database” (figura 4.3.).



**Figura 4.3.** Figura Add database

## 3

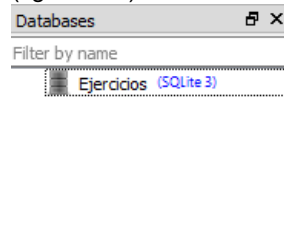
Aparece un formulario (figura 4.4.) en el que se debe indicar el tipo de base de datos (sección “Database type”), donde se ubicará la base de datos (sección “File”) y el nombre de la base de datos (sección “name”). Se elige el tipo “SQLite 3”. Se pulsa sobre el símbolo + para crear el archivo de la base de datos, aparece un navegador para ir a la carpeta donde se quiere guardar el archivo y se crea un archivo denominado “Ejercicios.sqlite3”.



**Figura 4.4.** Formulario de configuración.

## 4

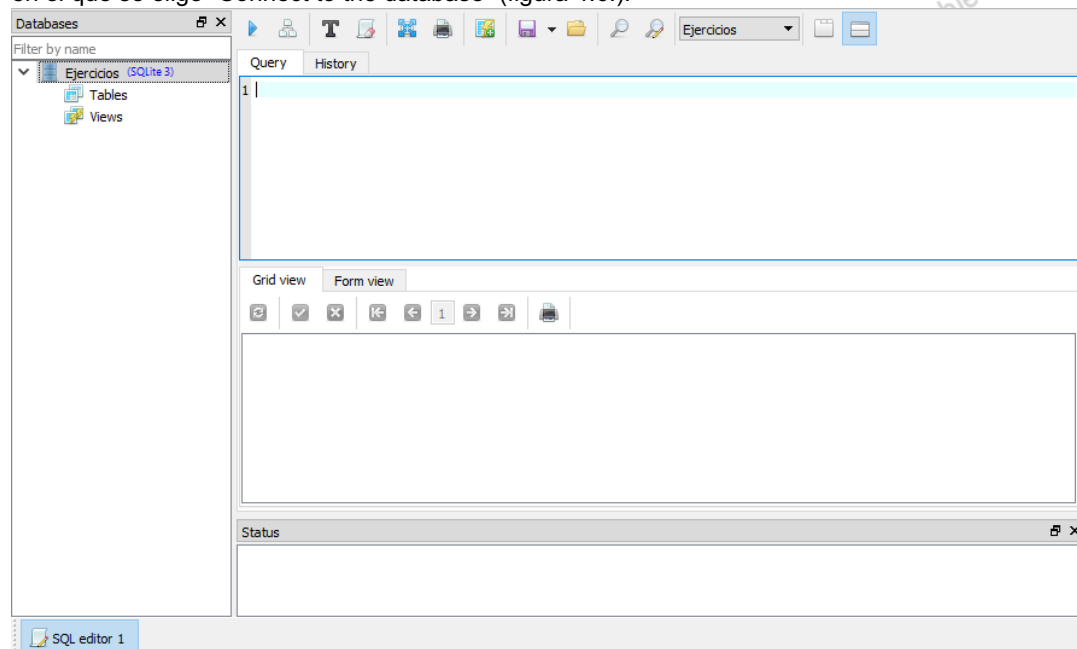
Cuando se pulsa sobre “OK”, la nueva base de datos aparece en el marco izquierdo de la interface principal (figura 4.5.).



**Figura 4.5.** Base de datos creada.

## 5

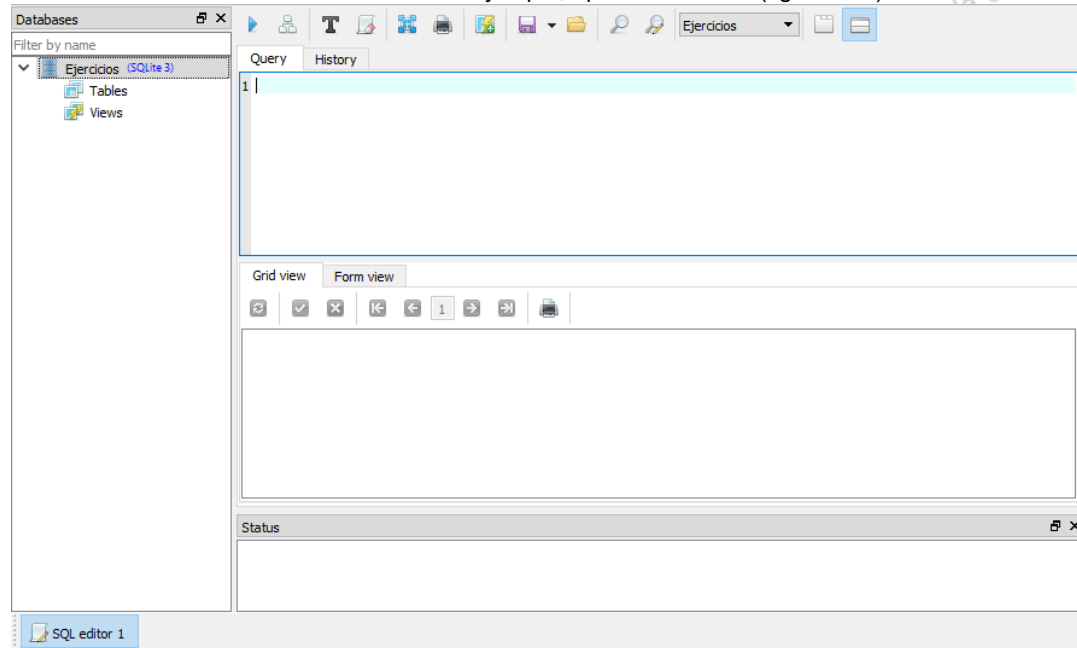
Para poder usar la base de datos, se selecciona con el ratón y, con botón derecho, aparece un desplegable en el que se elige “Connect to the database” (figura 4.6.).



**Figura 4.6.** Se pulsa sobre “Connect to the database”.

6

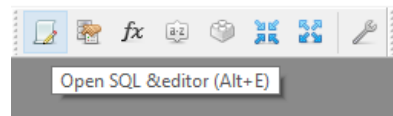
Una vez que se ha conectado con la base de datos, en el marco izquierdo aparecen listadas las tablas y vistas asociadas a la base de datos. En el ejemplo, aparecen vacías (figura 4.7.).



**Figura 4.7.** Recursos asociados con la base de datos.

7

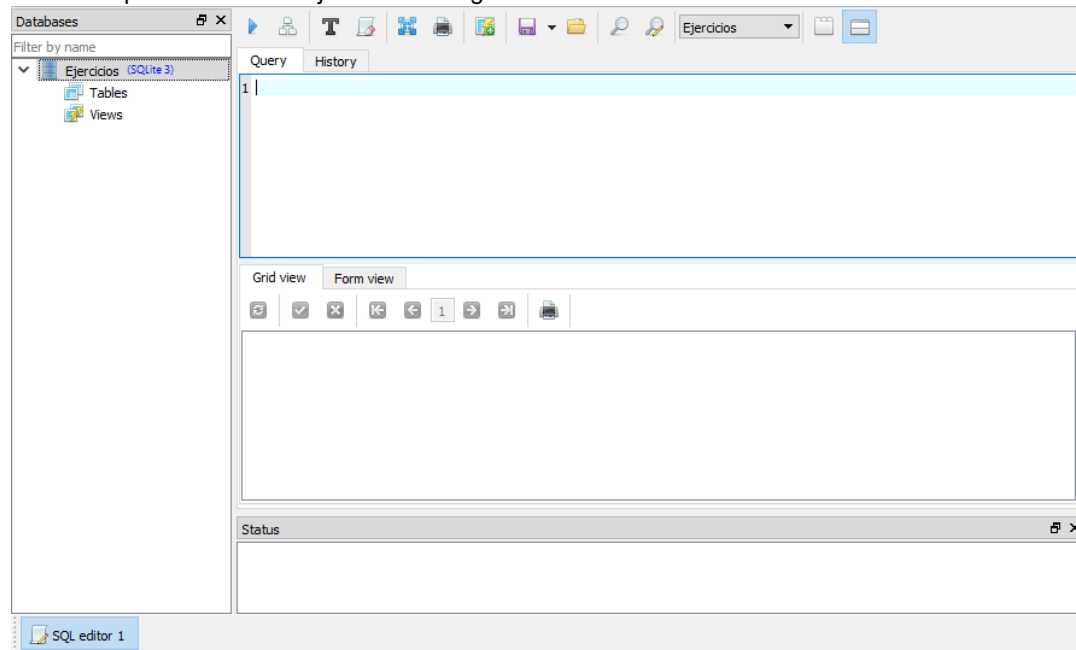
Para interactuar con la base de datos, se debe abrir el editor de SQL pulsando sobre el icono en forma de lápiz (figura 4.8).



**Figura 4.8.** Acceso al editor de SQL.

8

El editor de SQL se muestra en la parte derecha de la interface principal (figura 4.9.). Dicho editor será necesario para realizar los ejercicios del siguiente tema.



**Figura 4.9.** Editor de SQL.

## V. El lenguaje SQL

Las bases de datos relacionales se basan en el modelo relacional para almacenar y estructurar la información. SQL es un lenguaje relacional que permite manipular bases de datos relacionales.

Existen otro tipo de bases de datos no relacionales que se basan en otros modelos de datos diferentes al modelo relacional. Este tipo de bases de datos se denominan genéricamente como bases de datos NoSQL.

SQL es un lenguaje estándar ANSI/ISO de definición, manipulación y control de bases de datos relacionales. Se caracteriza por:

1

Es un lenguaje declarativo basado en el álgebra relacional.

2

Está soportado por la mayoría de los sistemas relacionales comerciales.

3

Se puede utilizar de manera interactiva o embebido en un programa.

En el modelo relacional se estructura la información en base a los conceptos:

1

Relación.

2

Atributos.

3

Tuplas.

En SQL se consideran conceptos similares, pero con una nomenclatura diferente:

1

Tablas.

2

Columnas.

3

Filas.

A continuación, se van a estudiar los elementos principales de lenguaje SQL.

## 5.1. Creación de tablas en SQL

Para crear una tabla, se utiliza la sentencia CREATE TABLE:

```
CREATE TABLE nombre_tabla (definición_columna[, definición_columna...]
```

```
[, restricciones_tabla]);
```

La definición de una columna consta del **nombre de la columna**, un **tipo de datos predefinido**, un **conjunto de definiciones por defecto** y **restricciones de columna**.

A continuación, se explica cada elemento.

### 5.1.1. Tipos de datos

Los principales tipos de datos predefinidos en SQL que pueden asociarse a una columna aparecen en la figura 4.10.



Tipos de datos predefinidos	
Tipos de datos	Descripción
CHARACTER (longitud)	Cadenas de caracteres de longitud fija.
CHARACTER VARYING (longitud)	Cadenas de caracteres de longitud variable.
BIT (longitud)	Cadenas de bits de longitud fija.
BIT VARYING (longitud)	Cadenas de bits de longitud variables.
NUMERIC (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimales con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
INTEGER	Números enteros.
SMALLINT	Números enteros pequeños.
REAL	Números con coma flotante con precisión predefinida.
FLOAT (precisión)	Números con coma flotante con la precisión especificada.
DOUBLE PRECISION	Números con coma flotante con más precisión predefinida que la del tipo REAL.
DATE	Fechas. Están compuestas de: YEAR año, MONTH mes, DAY día.
TIME	Horas. Están compuestas de HOUR hora, MINUT minutos, SECOND segundos.
TIMESTAMP	Fechas y horas. Están compuestas de YEAR año, MONTH mes, DAY día, HOUR hora, MINUT minutos, SECOND segundos.

Figura 4.10. Restricciones por tabla.

Para trabajar con el tiempo, se usa la siguiente nomenclatura:

1
YEAR (0001..9999)
2
MONTH (01..12)
3
DAY (01..31)
4
HOUR (00..23)
5
MINUT (00..59)
6
SECOND (00..59.precisión)



- Una columna fecha\_nacimiento podría ser del tipo DATE y tomar el valor '1978-12-25'.
- Una columna inicio\_pelicula podría ser del tipo TIME y tomar el valor '17:15:00.000000'.
- Una columna entrada\_clase podría ser de tipo TIMESTAMP y tomar el valor '1998-7-8 9:30:05'.

### 5.1.2. Definiciones por defecto

La opción **def\_defecto** permite especificar valores por omisión mediante la sentencia **DEFAULT (literal|función|NULL)** donde:

Si se elige la opción **NULL**, indica que la columna debe admitir valores nulos.

Si se elige la opción **literal**, señala que la columna tomará el valor indicado por el literal.

Si se elige la opción **función**, se indicará alguna de las funciones siguientes (tabla 3.1.).

Tabla 4.1. Funciones para definir valores por defecto.

Función	Descripción
{USER CURRENT_USER}	Identificador del usuario actual
SESSION_USER	Identificador del usuario de esta sesión
SYSTEM_USER	Identificador del usuario del sistema operativo
CURRENT_DATE	Fecha actual
CURRENT_TIME	Hora actual
CURRENT_TIMESTAMP	Fecha y hora actuales

### 5.1.3. Restricciones por columna

Cuando se define una columna, además de especificar su nombre y tipo, se pueden establecer un conjunto de restricciones que siempre se tienen que cumplir (Tabla 4.2.):

Tabla 4.2. Restricciones de columna.

Restricciones de columna	
Restricción	Descripción
NOT NULL	La columna no puede tener valores nulos
UNIQUE	La columna no puede tener valores repetidos. Es una clave alternativa
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria
REFERENCES tabla [(columna)]	La columna es la clave foránea de la columna de la tabla especificada
CHECK (condiciones)	La columna debe cumplir las condiciones especificadas



A las restricciones se les puede poner un nombre de la siguiente manera, por ejemplo:

[CONSTRAINT nombre\_restricción] CHECK (condiciones).

#### 5.1.4. Restricciones por tabla

Cuando se han definido las columnas de una tabla, a continuación, se pueden especificar restricciones sobre toda la tabla, que siempre se deberán cumplir (Tabla 4.3.):

Tabla 4.3. Restricciones por tabla.

Restricciones de tabla	
Restricción	Descripción
UNIQUE (columna [, columna...])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa
PRIMARY KEY (columna [, columna...])	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa
FOREIGN KEY (columna [, columna...]) REFERENCES tabla [(columna2 [,columna2...])]	El conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa
CHECK (condiciones)	La tabla debe cumplir las condiciones especificadas



A las restricciones se les puede poner un nombre de la siguiente manera, por ejemplo:

[CONSTRAINT nombre\_restricción] CHECK (condiciones).

#### 5.1.5. Los siguientes ejemplos de creación de tablas se realizaron utilizando SQLiteStudio

**Tabla sucursal**

- Nombre de sucursal de hasta 15 caracteres, llave primaria.
- Ciudad de localización de hasta 20 caracteres, no nulo, no repetible.
- Activos, numérico con precisión 12 y escala dos, valor por *default* 0.

**Create table sucursal**

```
(nombre_sucursal VARCHAR2(15) CONSTRAINT suc_PK PRIMARY KEY,  
ciudad CHAR(20) NOT NULL CONSTRAINT ci_UK UNIQUE,  
activos NUMBER(12,2) default 0);
```

**Tabla cliente**

- Dni de cliente de hasta 9 caracteres, no nulo, llave primaria.
- Nombre de cliente de hasta 25 caracteres, no nulo.
- Domicilio de hasta 50 caracteres, no nulo.

**Create table cliente**

```
(dni VARCHAR2(9) NOT NULL,  
nombre_cliente CHAR(35) NOT NULL,  
domicilio CHAR(50) NOT NULL,  
CONSTRAINT cl_PK PRIMARY KEY (dni));
```

**Tabla impositor**

- Dni de cliente de hasta 9 caracteres, no nulo, clave externa a Dni de tabla cliente.
- Número de cuenta de hasta 20 caracteres, no nulo.
- Llaves primarias: Dni y número de cuenta.
- Número de cuenta es clave externa a la clave externa de la tabla cuenta.

**Create table impositor**

```
(numero_cuenta CHAR (20) PRIMARY KEY,  
nombre_sucursal char(15) REFERENCES sucursal,  
saldo NUMBER(12,2) default 100,  
CONSTRAINT imp_minimo CHECK(saldo >=100))
```

**Tabla cuenta**

- Número de cuenta de hasta 20 caracteres, llave primaria.
- Nombre de sucursal de hasta 15 caracteres, clave externa a tabla sucursal.
- Saldo, numérico con precisión 12 y escala 2, valor por *default* 100, saldo mínimo 100.

**Create table cuenta**

```
(dni CHAR(9) CONSTRAINT imp_dni_FK REFERENCES cliente,
numero_cuenta CHAR(20) NOT NULL,
CONSTRAINT imp_PK PRIMARY KEY (dni, numero_cuenta),
CONSTRAINT imp_ct_FK FOREIGN KEY (numero_cuenta) REFERENCES cuenta)
```



Create table empleados(

codigo\_empl VARCHAR2(9) NOT NULL,

nombre\_empleado CHAR(35) NOT NULL,

teléfono CHAR(12) NOT NULL,



direccion CHAR(50) NOT NULL,

ciudad CHAR(50) NOT NULL,

sueldo NUMBER(12,2) default 100.0,

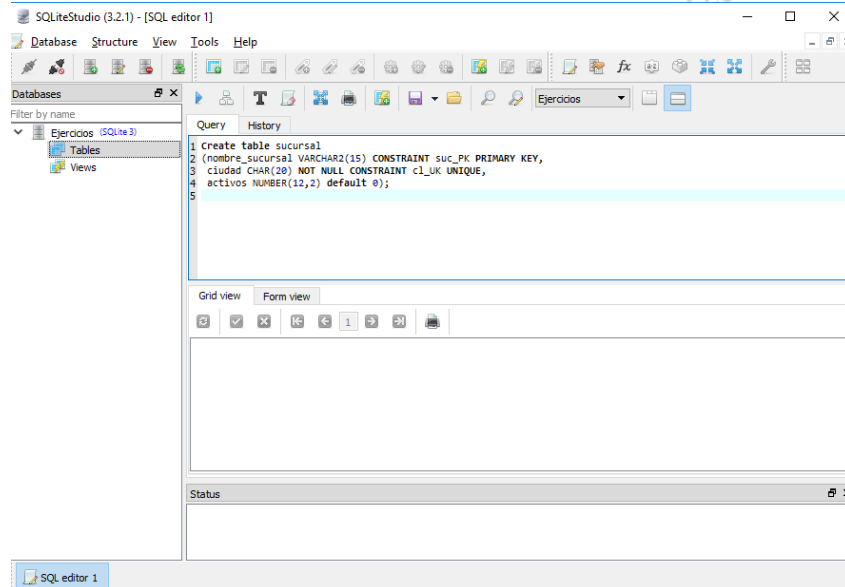
CONSTRAINT cl\_PK PRIMARY KEY (codigo\_empl)

);

Para crear las siguientes tablas, se puede usar un formulario gráfico (icono ) o bien el editor de SQL (icono ) . Para ilustrar el uso de SQL se va a utilizar el editor, en el que habrá que introducir las sentencias en la sintaxis de SQL.


1

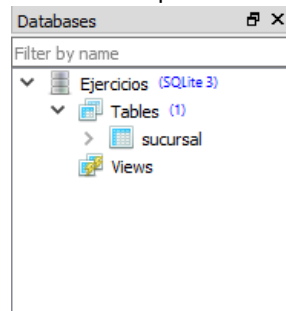
Se crea la tabla sucursal para almacenar la información sobre las mismas. Para ello se introduce la siguiente sentencia en el editor (figura 4.11.):



**Figura 4.11.** Sentencia de SQL para crear tabla.

2

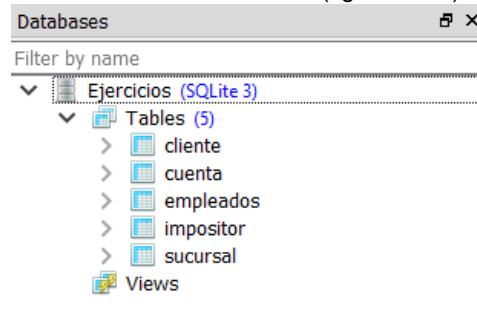
A continuación, se pulsa sobre el icono de ejecutar sentencia SQL (icono ) para crear la tabla. La nueva tabla aparece listada en la base de datos (figura 4.12.).



**Figura 4.12.** Tabla creada.

3

Las tablas cliente, cuenta e impositor se crean con el mismo procedimiento. Las nuevas tablas aparecen listadas en la base de datos (figura 4.13.).



**Figura 4.13.** Tablas creadas.

### 5.1.6. Claves foráneas

Cuando se define una clave foránea, se pueden especificar las políticas de borrado y modificación de filas que tiene una clave primaria referenciada por claves foráneas de la siguiente forma:

FOREIGN KEY clave\_secundaria REFERENCES tabla [(clave\_primaria)]

[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

Donde:

NO ACTION indica no realizar ninguna acción: un intento de borrar o actualizar un valor de clave primaria no será permitido si en la tabla referenciada hay un valor de clave foránea relacionado.

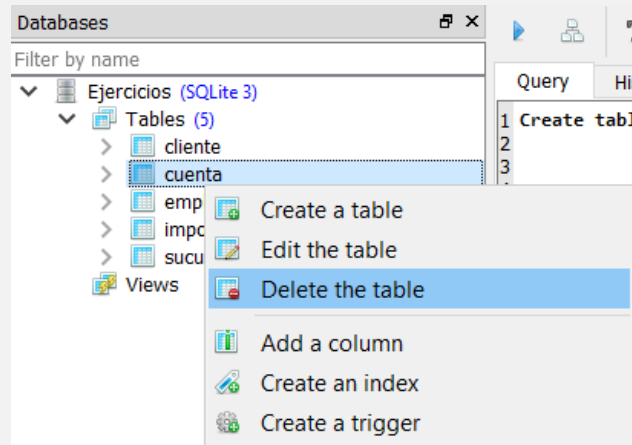
CASCADE representa la actualización en cascada. Borra o actualiza el registro en la tabla referenciada y automáticamente borra o actualiza los registros coincidentes en la tabla actual.

SET NULL borra o actualiza el registro en la tabla referenciada y establece en NULL la/s columna/s de clave foránea en la tabla actual.

SET DEFAULT indica que se ponga el valor especificado por defecto.



Por ejemplo, elimine las tablas cuenta e impositor; para ello, se selecciona cada tabla con el ratón y con botón derecho aparece un desplegable en el que se elige "Delete the table". Se van a crear nuevamente las tablas con claves foráneas (figura 4.14.)



**Figura 4.14.** Eliminar tabla.

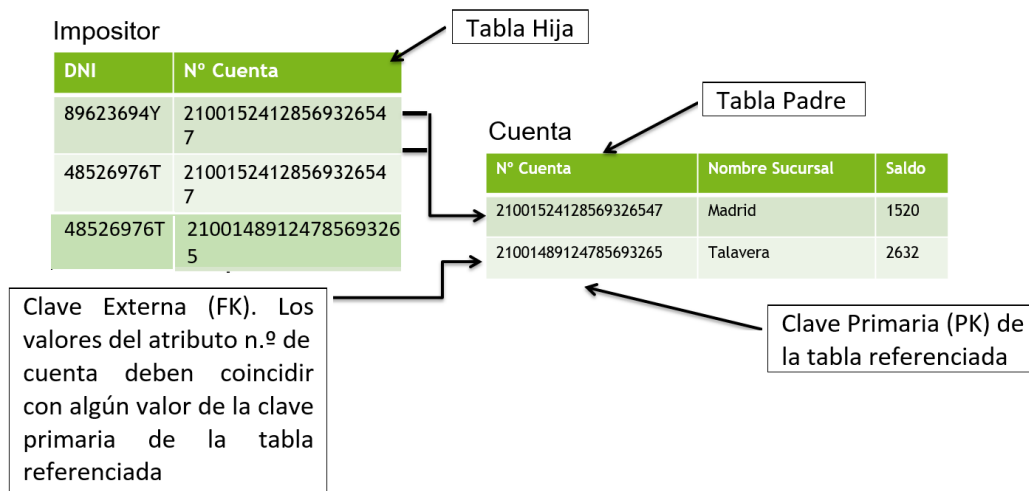
#### Create table cuenta

```
(numero_cuenta CHAR (20) PRIMARY KEY,
nombre_sucursal char(15)
CONSTRAINT ct_FK REFERENCES sucursal on delete set null,
saldo NUMBER(12,2) default 100,
CONSTRAINT imp_minimo CHECK(saldo >=100))
```

#### Create table impositor

```
(dni CHAR(9) CONSTRAINT imp_dni_FK REFERENCES cliente on delete cascade,
numero_cuenta CHAR(20),
CONSTRAINT imp_PK PRIMARY KEY (dni, numero_cuenta),
CONSTRAINT imp_ct_FK FOREIGN KEY (numero_cuenta) REFERENCES cuenta
on delete cascade)
```





**Figura 4.15.** Ejemplo de claves foráneas.

## 5.2. Modificación de tablas

Para modificar una tabla, se utiliza la sentencia **ALTER TABLE**:

`ALTER TABLE nombre_tabla {acción_modificar_columna | acción_modif_restricción_tabla};`

Donde:

**acción\_modificar\_columna** puede ser:

`{ADD [COLUMN] columna def_columna |`

`ALTER [COLUMN] columna {SET def_defecto|DROP DEFAULT}|`

`DROP [COLUMN ] columna {RESTRICT|CASCADE}}`

**acción\_modif\_restricción\_tabla** puede ser:

`{ADD restricción|DROP CONSTRAINT restricción {RESTRICT|CASCADE}}`

Así pues, las acciones de modificación que pueden realizarse sobre una tabla son:

### Añadir atributos

**Añadir** atributos a una tabla.

`alter table R add Atributo Dominio [propiedades]`

**Eliminar atributos**

**Eliminar** atributos de una tabla.

alter table *R* drop COLUMN Atributo

- No se puede eliminar la única columna de una tabla.
- Si la columna interviene en una constraint dará error:

alter table *R* drop Atributo CASCADE CONSTRAINTS

**Modificar atributos**

**Modificar** atributos en una tabla.

alter table *R* modify (Atributo Dominio [propiedades])

**Renombrar atributos**

**Renombrar** atributos de una tabla.

alter table *R* rename column Atributo1 to Atributo2

**Añadir restricciones**

**Añadir** restricciones a una tabla.

alter table *R* add CONSTRAINT nombre Tipo (columnas)

**Elimina restricciones**

**Eliminar** restricciones de una tabla.

alter table *R* drop {PRIMARY KEY|UNIQUE(campos)|

CONSTRAINT nombre [CASCADE]}

La opción CASCADE hace que se eliminen las restricciones de integridad que dependen de la eliminada.

**Desactiva restricciones**

**Desactivar** restricciones.

alter table *R* disable CONSTRAINT nombre [CASCADE]

**Activar restricciones**

alter table *R* enable CONSTRAINT nombre

**Añade el atributo comisión (numérico con precisión 4 y escala 2) a la tabla cuenta.**

```
ALTER TABLE cuenta ADD comision NUMBER(4,2);
```

**Añade el atributo fecha\_apertura (tipo fecha) a la tabla cuenta.**

```
ALTER TABLE cuenta ADD fecha_apertura DATE;
```

**Elimina el atributo nombre sucursal de la tabla cuenta. Como se puede ver, en SQLite Studio no funciona el comando DROP COLUMN, por lo que es necesario utilizar transacciones:**

```
ALTER TABLE cuenta DROP COLUMN nombre_sucursal;

PRAGMA foreign_keys=off;

BEGIN TRANSACTION;

CREATE TEMPORARY TABLE cuenta_temporal(numero_cuenta, saldo, comision, fecha_apertura);

INSERT INTO cuenta_temporal SELECT numero_cuenta, saldo, comision, fecha_apertura FROM cuenta;

DROP TABLE cuenta on delete cascade;

CREATE TABLE cuenta(numero_cuenta, saldo, comision, fecha_apertura);

INSERT INTO cuenta SELECT numero_cuenta, saldo, comision, fecha_apertura FROM cuenta_temporal;

DROP TABLE cuenta_temporal;

COMMIT;

PRAGMA foreign_keys=on;
```

### 5.3. Otras operaciones sobre tablas

#### Borrar una tabla

Para borrar una tabla, se utiliza la sentencia DROP TABLE:

```
DROP TABLE nombre_tabla{RESTRICT|CASCADE};
```

Donde:

- La opción RESTRICT indica que la tabla no se borrará si está referenciada.
- La opción CASCADE indica que todo lo que referencie a la tabla se borrará con esta.

Descripción de una tabla
describe R
Renombrar una tabla
rename R to S
Borrar contenidos
truncate table R

## 5.4. Índices

Los índices permiten que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY y UNIQUE.

Se pueden crear explícitamente para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente. En la figura 4.16., se muestra un ejemplo.

CREATE [unique] INDEX *NombreIndice* ON *NombreTabla*(*col1*,...,*colk*);

DNI	Nombre	Apellido	Dirección	Nº Empleado
3274623R	Antonio	López	Retama 14	RA45823633
27347234T	Marta	Sánchez	Loreto 134	PE74563669
85647456W	Antonio	López	Plaza Tornado 23	TY45236916
37562365F	María	Puente	Travesía del Reloj 1	MU45992355
34126455Y	Juan	Panero	Viviana 123	YR42326524

Varias filas pueden tomar los mismos valores en los atributos del índice

No hay dos filas con el mismo nº de registro

```
Create index ID_Nombre on Empleado (Apellido, Nombre);
```

```
Create unique index ID_Emp on Empleado (NºEmpleado);
```

Figura 4.16. Índices

## 5.5. Consultas y recuperación de información en SQL

### 5.5.1. Consultas sobre una sola tabla

**SELECT**

Para hacer consultas sobre una tabla, se utiliza la sentencia SELECT:

- *SELECT nombre\_columna\_a\_seleccionar [[AS] col\_renombrada] [,nombre\_columna\_a\_seleccionar [[AS] col\_renombrada]...] FROM tabla\_a\_consultar [[AS] tabla\_renombrada];*

Obsérvese que la palabra clave AS permite renombrar las columnas que se quieren seleccionar o las tablas que se quieren consultar. Esta palabra es opcional y muchas veces se sustituye por un espacio en blanco. . Para realizar los siguientes ejercicios, es necesario ejecutar el archivo *Inserts.sql*/dentro de la base de datos Ejercicios. Se puede descargar el archivo [en este enlace](#).



Por ejemplo, si se quieren seleccionar las columnas dni y nombre\_cliente de la tabla clientes, se usaría la sentencia:

- *SELECT dni, nombre\_cliente FROM cliente;*

\*

Sin embargo, si se quieren recuperar todas las columnas de la tabla se usa el símbolo \*, en vez de listar todas las columnas:

- *SELECT \* FROM cliente;*

Si se quieren seleccionar qué filas han de ser recuperadas, hay que utilizar en la consulta SELECT la palabra reservada WHERE:

- *SELECT [DISTINCT|ALL] nombre\_columnas\_a\_seleccionar FROM tabla\_a\_consultar [WHERE condiciones];*

**Tener en cuenta**

Téngase en cuenta que:

- La cláusula **WHERE** permite recuperar solo aquellas filas que cumplen la condición especificada.
- La cláusula **DISTINCT** permite ordenar que nos muestre las filas resultantes sin repeticiones. La opción por defecto es ALL, que indica que muestre todas las filas.
- Para construir las condiciones de la cláusula **WHERE**, es necesario usar operadores de comparación o lógicos: < (menor), > (mayor), = (igual), <= (menor o igual), >= (mayor o igual), <> (distinto), AND (conjunción de condiciones), OR (disyunción de condiciones), NOT (negación).



Por ejemplo, si se quieren recuperar los diferentes saldos de la tabla cuenta:

```
SELECT DISTINCT saldo FROM cuenta;
```

Y si se quieren recuperar las cuentas cuyo saldo sea mayor que 1000:

```
SELECT * FROM cuenta WHERE saldo > 1000;
```

Una subconsulta es una consulta incluida dentro de otra consulta, que aparecerá como parte de una cláusula WHERE o HAVING —se verá más adelante—.



Por ejemplo, se necesita obtener los números de cuenta del cliente llamado 'Luis Díaz':

```
Select numero_cuenta from impositor where dni = (select dni from cliente where nombre_cliente = 'Luis Díaz')
```

## WHERE

En la condición que aparece en la cláusula WHERE, se puede utilizar un conjunto de predicados predefinidos para construir las condiciones:

### BETWEEN

Expresa que se quiere encontrar un valor entre unos límites concretos:

- **SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE** columna **BETWEEN** límite1 **AND** límite2;



Por ejemplo, se quieren recuperar todos los empleados cuyos sueldos están entre 1000 y 2000 euros:

```
SELECT codigo_empl FROM empleados WHERE sueldo BETWEEN 1000 and 2000;
```

**IN**

Comprueba si un valor coincide con los elementos de una lista (**IN**) o no coincide (**NOT IN**):

- **SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE** columna [**NOT**] **IN** (valor1, ..., valorN);



Por ejemplo, se quieren recuperar todos los empleados que viven en Madrid y Zaragoza:

```
SELECT * FROM empleados WHERE ciudad IN ('Madrid', 'Zaragoza');
```

**LIKE**

Comprueba si una columna de tipo carácter cumple una condición determinada.

- **SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE** columna **LIKE** condición;

**Caracteres comodín**

Existen un conjunto de caracteres que actúan como comodines:

- El carácter **\_** para representar un carácter individual.
- El carácter **%** para expresar una secuencia de caracteres incluyendo la secuencia vacía.



Por ejemplo, si se quieren recuperar los empleados cuya ciudad de residencia termina por la letra d:

```
SELECT * FROM empleados WHERE ciudad LIKE '%d'
```

**'\_'**

Y si se quiere refinar la consulta anterior y recuperar los empleados cuya ciudad de residencia termina por la letra d y el nombre de la ciudad tiene 6 letras:

```
SELECT * FROM empleados WHERE ciudad LIKE '_____d'
```

**IS NULL**

Comprueba si un valor nulo (IS NULL) o no lo es (IS NOT NULL):

**SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE** columna **IS [NOT] NULL**;



Por ejemplo, se quieren recuperar todos los empleados que no tienen un número de teléfono:

**SELECT \* FROM** empleados **WHERE** teléfono **IS NULL**;

**EXISTS**

Comprueba si una consulta produce algún resultado (EXISTS) o no (NOT EXISTS):

**SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE [NOT] EXISTS** subconsulta;



Por ejemplo, se quieren recuperar todos los clientes que tienen alguna cuenta:

**select \* from** cliente **where exists** ( **select \* from** impositor );

**ANY/SOME/ALL**

Comprueba si todas (ALL) o algunas (SOME/ANY) de las filas de una columna cumplen las condiciones especificadas:

**SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar **WHERE** columna operador\_comparación {**ALL|ANY|SOME**} subconsulta;



**ORDER BY**

Para ordenar los resultados de una consulta, se utiliza la cláusula ORDER BY:

**SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla\_a\_consultar [**WHERE** condiciones] **ORDER BY** columna\_según\_la\_cual\_se\_quiere\_ordenar [DESC] [, col\_ordenación [DESC]]...;

Por defecto, los resultados se ordenan de manera ascendente. Así, si se realiza una ordenación descendente, se debe indicar usando la cláusula DESC.



Por ejemplo, si se quieren ordenar los empleados por orden alfabético ascendente de acuerdo a su nombre y descendente de acuerdo a su sueldo:

```
SELECT * FROM empleados ORDER BY nombre_empleado, sueldo DESC
```

**5.5.2. Consultas sobre más de una tabla**

En la cláusula FROM, es posible especificar más de una tabla cuando se quieren consultar columnas de tablas diferentes. Existen varios casos:

**Combinación**

Se crea una sola tabla a partir de las tablas especificadas, haciendo coincidir los valores de las columnas relacionadas de las tablas.

- **SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla1 **JOIN** tabla2 {**ON** condiciones|**USING** (columna [, columna...])} [**WHERE** condiciones];



Obsérvese:

- La opción ON permite expresar condiciones con cualquiera de los operadores de comparación sobre las columnas especificadas.
- Es posible utilizar una misma tabla dos veces usando alias diferentes para diferenciarlas.
- Puede ocurrir que las tablas consideradas tengan columnas con los mismos nombres. En este caso, es obligatorio diferenciarlas especificando en cada columna a qué tabla pertenecen.



- Por ejemplo, si se quiere obtener el dni de los clientes y sus saldos.

```
SELECT c.dni, c. nombre_cliente, cu.saldo from cliente c JOIN impositor i ON c.dni = i.dni JOIN cuenta cu ON i. numero_cuenta = cu. numero_cuenta
```

- Por ejemplo, si se quiere obtener el dni y nombre de los clientes que tengan saldo mayor que 100:

```
SELECT c.dni, c. nombre_cliente, cu.saldo from cliente c JOIN impositor i ON c.dni = i.dni JOIN cuenta cu ON i. numero_cuenta = cu. numero_cuenta where cu.saldo > 100;
```

### Combinación natural

Consiste en una combinación en la que se eliminan las columnas repetidas.

- **SELECT** nombre\_columnas\_a\_seleccionar **FROM** tabla1 **NATURAL JOIN** tabla2 [**WHERE** condiciones];



Por ejemplo, si se quiere obtener las cuentas por sucursal:

```
select * from sucursal NATURAL JOIN cuenta
```

De forma equivalente se podría consultar:

- `select * from sucursal JOIN cuenta using ( nombre_sucursal )`

### Combinación interna vs. externa

- **Interna(inner join)**. Solo se consideran las filas que tienen valores idénticos en las columnas de las tablas que compara.

```
SELECT nombre_columnas_a_seleccionar FROM t1 [NATURAL] [INNER] JOIN t2 {ON condiciones|USING (columna [,columna...])} [WHERE condiciones];
```

- **Externa(outer join)**. Se consideran los valores de la tabla derecha, de la izquierda o de ambas tablas.

```
SELECT nombre_columnas_a_seleccionar FROM t1 [NATURAL] [LEFT|RIGHT|FULL] [OUTER] JOIN t2 {ON condiciones|USING (columna [,columna...])} [WHERE condiciones];
```

### Combinación de más de 2 tablas

Para combinar más de 2 tablas, basta con añadirlas en el FROM de la consulta y establecer las relaciones necesarias en el WHERE, o bien combinar las tablas por pares de manera que la resultante será el primer componente del siguiente par.



Por ejemplo, si se quieren combinar las tablas cliente, cuenta e impositor:

```
SELECT * FROM cliente, cuenta , impositor
```

O bien:

```
SELECT * FROM (cliente c JOIN impositor i ON c.dni = i.dni) JOIN cuenta cu ON  
i.numero_cuenta = cu.numero_cuenta;
```

Entre 2 tablas se pueden definir las siguientes operaciones:

### Unión

Permite unir los resultados de 2 o más consultas.

- `SELECT columnas FROM tabla [WHERE condiciones] UNION [ALL] SELECT columnas FROM tabla [WHERE condiciones];`

Obsérvese que la cláusula ALL permite indicar si se quieren obtener todas las filas de la unión (incluidas las repetidas).

Por ejemplo, si se quiere obtener todas las ciudades que aparecen en las tablas de la base de datos:

- `SELECT ciudad FROM empleados UNION SELECT ciudad FROM sucursal;`

### Intersección

Permite hacer la intersección entre los resultados de 2 o más consultas.

**SELECT** columnas **FROM** tabla **[WHERE condiciones] INTERSECT [ALL] SELECT** columnas **FROM** tabla **[WHERE condiciones];**

Téngase en cuenta que la cláusula ALL permite indicar si se quieren obtener todas las filas de la intersección (incluidas las repetidas).

Se puede simular usando:

- **IN**

**SELECT** columnas **FROM** tabla **WHERE** columna **IN (SELECT** columna **FROM** tabla **[WHERE condiciones]);**

- **EXISTS**

**SELECT** columnas **FROM** tabla **WHERE EXISTS** (**SELECT** \* **FROM** tabla **WHERE** condiciones);



Por ejemplo, usando la **intersección**.

**SELECT** ciudad **FROM** empleados **INTERSECT** **SELECT** ciudad **FROM** sucursal;

#### Usando **IN**

**SELECT** c.ciudad **FROM** empleados c **WHERE** c.ciudad **IN** (**SELECT** d.ciudad **FROM** sucursal d);

#### Usando **EXISTS**

**SELECT** c.ciudad **FROM** empleados c **WHERE EXISTS** (**SELECT** \* **FROM** sucursal d **WHERE** c.ciudad = d.ciudad);

### Diferencia

Permite diferenciar los resultados de 2 o más consultas.

- **SELECT** columnas **FROM** tabla [**WHERE** condiciones] **EXCEPT** [**ALL**] **SELECT** columnas **FROM** tabla [**WHERE** condiciones];

Obsérvese que la cláusula **ALL** permite indicar si se quieren obtener todas las filas de la intersección (incluidas las repetidas).

Se puede simular usando:

#### - **NOT IN**

**SELECT** columnas **FROM** tabla **WHERE** columna **NOT IN** (**SELECT** columna **FROM** tabla [**WHERE** condiciones]);

#### - **NOT EXISTS**

**SELECT** columnas **FROM** tabla **WHERE NOT EXISTS** (**SELECT** \* **FROM** tabla **WHERE** condiciones);



Por ejemplo, si se quiere saber las ciudades de los empleados en los que no hay sucursales:

Usando la intersección.

```
SELECT ciudad FROM empleados EXCEPT SELECT ciudad FROM sucursal;
```

Usando IN

```
SELECT c.ciudad FROM empleados c WHERE c.ciudad NOT IN (SELECT d.ciudad FROM sucursal d);
```

Usando EXISTS

```
SELECT c.ciudad FROM empleados c WHERE NOT EXISTS (SELECT * FROM sucursal d WHERE c.ciudad = d.ciudad);
```

## 5.6. Modificación de tablas en SQL

### 5.6.1. Operaciones de actualización

#### Inserción

Para poder consultar los datos de una base de datos, hay que introducirlos con la sentencia INSERT INTO VALUES:

```
INSERT INTO nombre_tabla [(columnas)] {VALUES ({v1|DEFAULT|NULL}, ..., {vn|DEFAULT|NULL})|<consulta>};
```

**NOTA:** se recomienda que, una vez que el alumno haya introducido datos en la base de datos de ejemplo 'Ejercicios', repase las sentencias SELECT del apartado 5.5.

Téngase en cuenta que:

- Los valores v1, v2... vn se deben corresponder con las columnas de la tabla especificada y deben estar en el mismo orden, a menos que las volvamos a colocar a continuación del nombre de la tabla. En este último caso, los valores se deben disponer de forma coherente con el nuevo orden.
- Si se quiere especificar un valor por omisión se usa la palabra reservada DEFAULT y si se trata de un valor nulo se usa la palabra reservada NULL.
- Obsérvese que, para insertar más de una fila con una sola sentencia, se deben obtener los datos mediante una consulta a otras tablas.



Por ejemplo, si se quiere insertar en una tabla cliente que tiene las columnas: dni, nombre\_cliente, domicilio, se podría hacer de dos formas, se podría hacer de dos formas:

```
INSERT INTO cliente (dni,nombre_cliente,domicilio) VALUES ( '12345', 'Luis Díaz', 'Santa Rita No. 18' );
```

```
INSERT INTO cliente (dni,nombre_cliente,domicilio) VALUES ( '12344', 'Javier Lozano', '11 Sur No.35' );
```

```
INSERT INTO cliente (dni,nombre_cliente,domicilio) VALUES ( '12343', 'Andres López', 'San Agustin No.81' );
```

```
INSERT INTO cliente (dni,nombre_cliente,domicilio) VALUES ( '12346', 'Alondra López', 'Santa Rita No.1501' );
```

O bien:

```
INSERT INTO cliente VALUES ( '12345', 'Luis Díaz', 'Santa Rita No. 18' );
```

```
INSERT INTO cliente VALUES ( '12344', 'Javier Lozano', '11 Sur No.35' );
```

```
INSERT INTO cliente VALUES ( '12343', 'Andres López', 'San Agustin No.81' );
```

```
INSERT INTO cliente VALUES ( '12346', 'Alondra López', 'Santa Rita No.1501' );
```

Insertar cuentas con su relación impositor:

```
INSERT INTO sucursal VALUES('Dorada', 'Madrid', 1500);
```

```
INSERT INTO sucursal VALUES('Angelopolis', 'Sevilla', 1150);
```

```
INSERT INTO cuenta VALUES ( '6789', 'Dorada', 120.01, 3.1, '2011-03-12' );
```

```
INSERT INTO cuenta VALUES ( '6788', 'Dorada', 23445.01, 3.1, '2011-03-12' );
```

```
INSERT INTO cuenta VALUES ( '6787', 'Dorada', 4566.78, 3.1, '2011-03-12' );
```

```
INSERT INTO cuenta VALUES ( '6786', 'Angelopolis', 1222.3, 3.1, '2011-03-12' );
```

```
INSERT INTO impositor VALUES ( '12346', '6789' );
```

```
INSERT INTO impositor VALUES ( '12345', '6786' );
```

```
INSERT INTO impositor VALUES ( '12344', '6788' );
```

```
INSERT INTO impositor VALUES ( '12343', '6787' );
```

También es posible obtener los datos mediante una consulta SELECT que actúe como proveedor de datos, supongamos que existe la tabla nuevas\_cuentas:

```
INSERT INTO cuenta SELECT * FROM nuevas_cuentas
```

## Borrado

Para borrar valores de algunas filas de una tabla, se usa la sentencia DELETE:

**DELETE FROM** nombre\_tabla [**WHERE** condiciones];

Hay que tener en cuenta que si no se utiliza la cláusula WHERE se borrarán todas las filas de la tabla, en cambio, si se utiliza WHERE, solo se borrarán aquellas filas que cumplan las condiciones especificadas.



Por ejemplo, si se quieren borrar todas las filas de la tabla empleados se usaría la sentencia:

```
DELETE FROM empleados;
```

Sin embargo, si solo se quieren borrar las filas de la tabla en las que el valor de la columna codigo\_empl sea '34567', entonces se usaría la sentencia:

```
delete from empleados where codigo_empl = '34567'
```

## Modificación

Para modificar los valores de algunas filas de una tabla se usa la sentencia UPDATE:

**UPDATE** nombre\_tabla **SET** columna = {expresión|DEFAULT|NULL} [, columna = {expresión|DEFAULT|NULL} ...] **WHERE** condiciones;

La cláusula SET indica qué columna modificar y los valores que puede recibir, y la cláusula WHERE especifica qué filas deben actualizarse.



Por ejemplo, si se quiere inicializar el sueldo de todos los empleados en 500 euros:

```
UPDATE empleados SET sueldo = 500;
```

La parte WHERE es opcional y, si no se especifica, se actualizarán todas las tuplas de la tabla.

```
UPDATE empleados SET sueldo = 500 where codigo_empl='34566';
```

La cláusula WHERE admite consultas anidadas. Por ejemplo "Modificar las cuentas del cliente con dni = 12345"

```
update cuenta set saldo = 10000 where numero_cuenta in ( SELECT numero_cuenta
FROM impositor where dni ='12345' )
```

## 5.6.2 Operaciones sobre tablas

Las funciones de agregación son funciones que permiten realizar operaciones sobre los datos de una columna. Algunas funciones se muestran en la figura 4.16.

Funciones de agregación	
Función	Descripción
COUNT	Nos da el número total de filas seleccionadas
SUM	Suma los valores de una columna
MIN	Nos da el valor mínimo de una columna
MAX	Nos da el valor máximo de una columna
AVG	Calcula el valor medio de una columna

**Figura 4.16.** Índices

En general, las funciones de agregación se aplican a una columna, excepto COUNT, que se aplica a todas las columnas de las tablas seleccionadas. Se indica como COUNT (\*).

Sin embargo, si se especifica COUNT (distinct columna), solo contará los valores no nulos ni repetidos, y si se especifica COUNT (columna), solo contará los valores no nulos.





Por ejemplo, si se quiere contar el número de clientes de la tabla clientes cuya ciudad es Madrid:

```
SELECT COUNT(*) AS numero_ciudad FROM sucursal WHERE ciudad = 'Madrid';
```

Al realizar una consulta, las filas se pueden agrupar de la siguiente manera:

```
SELECT nombre_columnas_a seleccionar FROM tabla_a_consultar [WHERE condiciones] GROUP BY columnas_según_las_cuales_se_quiere_agrupar
```

```
[HAVING condiciones_por_grupos] [ORDER BY columna_ordenación [DESC] [, columna [DESC]...]];
```

Donde:

- La cláusula GROUP BY permite agrupar las filas según las columnas indicadas, excepto aquellas afectadas por funciones de agregación.
- La cláusula HAVING especifica las condiciones para recuperar grupos de filas.

Una vista es una tabla ficticia —no existen como un conjunto de valores almacenados en la base de datos— que se construye a partir de una consulta a una tabla real. Para definir una vista se usa la siguiente sintaxis:

#### CREATE VIEW

```
CREATE VIEW nombre_vista [(lista_columnas)] AS (consulta) [WITH CHECK OPTION];
```

A continuación de donde se indica el nombre de la vista, se pueden especificar los nombres de las columnas de la vista, se define la consulta que construirá la vista, tras lo que se puede añadir la cláusula “with check option” para evitar inserciones o actualizaciones excepto en los registros en que la cláusula WHERE de la consulta se evalúe como true.

#### DROP VIEW

Para borrar una vista se utiliza la sentencia DROP VIEW:

```
DROP VIEW nombre_vista (RESTRICT|CASCADE);
```

Donde:

- La opción **RESTRICT** indica que la vista no se borrará si está referenciada.
- La opción **CASCADE** indica que todo lo que referencie a la vista se borrará con esta.

Para ilustrar las vistas, se van a considerar las siguientes tablas:

Grid view		Form view	
		1	
		Total rows loaded: 4	
dni	nombre_cliente	domicilio	
1 12345	Luis Díaz	Santa Rita No. 18	
2 12344	Javier Lozano	11 Sur No.35	
3 12343	Andres López	San Agustin No.81	
4 12346	Alondra López	Santa Rita No.1501	

**Figura 4.17.** Tabla de clientes.

Query

Data

Triggers

DDL

Grid view

Form view

1

Total rows loaded: 3

	codigo_er	nombre_empleado	teléfono	direccion	ciudad	sueldo
1	34565	Orlando García	121234563	Calle Tlaxcala No.89	Guadalajara	500
2	34567	Santiago Obrador	123456789	11 Sur No 201	Madrid	500
3	34566	Beatriz Perez	987654321	18 Poniente No. 1302	Puebla	500

**Figura 4.18.** Tabla de empleados.

Si se quiere crear una vista que indique para cada cliente sus cuentas y saldos, se definiría la vista:

```
CREATE VIEW cuentas_por_cliente AS SELECT c.dni, c.nombre_cliente,cu.numero_cuenta,cu.saldo from
cliente c join impositor i on c.dni = i.dni join cuenta cu on i.numero_cuenta=cu.numero_cuenta;
```

Y se obtendría la vista (figura 4.19.):

Query		Data		Triggers		DDL		
Grid view				Form view				
					1			Total rows
	dni	nombre_cliente	numero_cuenta	saldo				
1	12345	Luis Díaz	6789	10000				
2	12345	Luis Díaz	6786	10000				
3	12344	Javier Lozano	6788	23445.01				
4	12343	Andres López	6787	4566.78				

**Figura 4.19.** Vista resultante.

## VI. Resumen



En esta unidad, se han abordado algunos de los conceptos más importantes del modelo relacional. El modelo relacional se basa en el concepto de relación como una abstracción en la cual se quiere almacenar información. Este modelo conceptual es el fundamento de las bases de datos relacionales, que constituyen el mecanismo más extendido actualmente de persistencia de datos.

Las bases de datos relacionales almacenan la información mediante tablas formadas por columnas y filas. Una tabla representa la información de un tipo de entidades o elementos de una misma clase de los cuales se quiere almacenar información. Cada fila representa una instancia de una entidad y las columnas son los atributos de las entidades de los que se guarda información.

También, en esta unidad, se ha introducido un lenguaje para gestionar las bases de datos relacionales. Para poder manipular la información de una tabla existe un lenguaje de consultas estándar denominado SQL que permite recuperar la información de una tabla, modificar los datos o introducir nuevos.

## VII. Caso práctico

Considérese una base de datos para gestionar las solicitudes de acceso de estudiantes a los institutos. La información que se desea almacenar es:

### Tablas

Se van a crear 3 tablas: **institutos**, **estudiantes** y **solicitudes**.

```
CREATE TABLE Institutos(Nombre_Inst CHAR(35), Area CHAR(35), Plazas INTEGER,
```

```
PRIMARY KEY ( Nombre_Inst ),
```

```
UNIQUE ( Nombre_Inst, Area ));
```

```
CREATE TABLE Estudiantes(ID INTEGER, Nombre_Est CHAR(35), Puntos REAL, Valor INTEGER,
```

```
PRIMARY KEY ( ID ),
```

```
UNIQUE ( ID ));
```

```
CREATE TABLE Solicitudes(ID INTEGER, Nombre_Inst CHAR(35), Via CHAR(35), Decision CHAR(35),
```

```
PRIMARY KEY ( ID, Nombre_Inst, Via ),
```

```
FOREIGN KEY ( ID ) REFERENCES Estudiantes ( ID ),
```

```
FOREIGN KEY ( Nombre_Inst ) REFERENCES
```

```
Institutos ( Nombre_Inst ),
```

UNIQUE ( ID, Nombre\_Inst, Via ) );

### Datos en tablas

A continuación, se rellenan las tablas con los siguientes datos:

**Tabla Institutos**

#	Nombre_Inst	Area	Plazas
1	Instituto San Isidro	Centro	150
2	Instituto Ramiro de Maeztu	Salamanca	360
3	Instituto Arturo Soria	Hortaleza	100
4	Instituto Torres Quevedo	Moncloa	210

**Tabla Estudiantes**

#	ID	Nombre_Est	Puntos	Valor
1	123	Antonio	8.9	1000
2	234	Juan	8.6	1500
3	345	Isabel	8.5	500
4	456	Doris	7.9	1000
5	543	Pedro	5.4	2000
6	567	Eduardo	6.9	2000
7	654	Alfonso	7.9	1000
8	678	Carmen	5.8	200
9	765	Javier	7.9	1500
10	789	Isidro	8.4	800
11	876	Irene	6.9	400
12	987	Elena	6.7	800

Tabla Solicitudes

#	ID	Nombre_Inst	Via	Decision
1	123	Instituto Ramiro de Maeztu	Tecnologia	Si
2	123	Instituto Ramiro de Maeztu	Ciencias Sociales	No
3	123	Instituto San Isidro	Tecnologia	Si
4	123	Instituto Torres Quevedo	Ciencias Sociales	Si
5	234	Instituto San Isidro	Ciencias	No
6	345	Instituto Arturo Soria	Tecnologia	Si
7	345	Instituto Torres Quevedo	Tecnologia	No
8	345	Instituto Torres Quevedo	Ciencias	Si
9	345	Instituto Torres Quevedo	Ciencias Sociales	No
10	678	Instituto Ramiro de Maeztu	Ciencias Sociales	Si
11	987	Instituto Ramiro de Maeztu	Tecnologia	Si
12	987	Instituto San Isidro	Tecnologia	Si
13	876	Instituto Ramiro de Maeztu	Tecnologia	No
14	876	Instituto Arturo Soria	Ciencias	Si
15	876	Instituto Arturo Soria	Ciencias Sociales	No
16	765	Instituto Ramiro de Maeztu	Ciencias Sociales	Si
17	765	Instituto Torres Quevedo	Ciencias Sociales	No
18	765	Instituto Torres Quevedo	Ciencias	Si
19	543	Instituto Arturo Soria	Tecnologia	No

## Consultas

Deben hacerse las siguientes consultas:

1. Obtener los nombres y notas de los estudiantes, así como el resultado de su solicitud, de manera que tengan un valor de corrección menor que 1000 y hayan solicitado la vía de "Tecnología" en el "Instituto Ramiro de Maeztu".
2. Obtener la información sobre todas las solicitudes: ID y nombre del estudiante, nombre del instituto, puntos y plazas, ordenadas de forma decreciente por los puntos y en orden creciente de plazas.
3. Obtener todas las solicitudes a vías denominadas como "Ciencias" o "Ciencias Sociales".
4. Obtener los estudiantes cuya puntuación ponderada cambia en más de un punto respecto a la puntuación original.
5. Borrar a todos los estudiantes que solicitaron más de 2 vías diferentes.
6. Obtener las vías en las que la puntuación máxima de las solicitudes está por debajo de la media.
7. Obtener los nombres de los estudiantes y las vías que han solicitado.
8. Obtener el nombre de los estudiantes y la puntuación con valor de ponderación menor de 1000 que hayan solicitado la vía de "Tecnología" en el "Instituto San Isidro".

## Solución

**Inserción**

La inserción de datos se haría de la siguiente manera:

**Tabla Institutos**

```
insert into Institutos values ('Instituto San Isidro', 'Centro', 150);
insert into Institutos values ('Instituto Ramiro de Maeztu', 'Salamanca', 360);
insert into Institutos values ('Instituto Arturo Soria', 'Hortaleza', 100);
insert into Institutos values ('Instituto Torres Quevedo', 'Moncloa', 210);
```

**Tabla Estudiantes**

```
insert into Estudiantes values (123, 'Antonio', 8.9, 1000);
insert into Estudiantes values (234, 'Juan', 8.6, 1500);
insert into Estudiantes values (345, 'Isabel', 8.5, 500);
insert into Estudiantes values (456, 'Doris', 7.9, 1000);
insert into Estudiantes values (567, 'Eduardo', 6.9, 2000);
insert into Estudiantes values (678, 'Carmen', 5.8, 200);
insert into Estudiantes values (789, 'Isidro', 8.4, 800);
insert into Estudiantes values (987, 'Elena', 6.7, 800);
insert into Estudiantes values (876, 'Irene', 6.9, 400);
insert into Estudiantes values (765, 'Javier', 7.9, 1500);
insert into Estudiantes values (654, 'Alfonso', 7.9, 1000);
insert into Estudiantes values (543, 'Pedro', 5.4, 2000);
```

**Tabla Solicitudes**

```
insert into Solicitudes values (123, 'Instituto Ramiro de Maeztu', 'Tecnología', 'Si');
insert into Solicitudes values (123, 'Instituto Ramiro de Maeztu', 'Ciencias Sociales', 'No');
```

```

insert into Solicitudes values (123, 'Instituto San Isidro', 'Tecnologia', 'Si');

insert into Solicitudes values (123, 'Instituto Torres Quevedo', 'Ciencias Sociales', 'Si');

insert into Solicitudes values (234, 'Instituto San Isidro', 'Ciencias', 'No');

insert into Solicitudes values (345, 'Instituto Arturo Soria', 'Tecnologia', 'Si');

insert into Solicitudes values (345, 'Instituto Torres Quevedo', 'Tecnologia', 'No');

insert into Solicitudes values (345, 'Instituto Torres Quevedo', 'Ciencias', 'Si');

insert into Solicitudes values (345, 'Instituto Torres Quevedo', 'Ciencias Sociales', 'No');

insert into Solicitudes values (678, 'Instituto Ramiro de Maeztu', 'Ciencias Sociales', 'Si');

insert into Solicitudes values (987, 'Instituto Ramiro de Maeztu', 'Tecnologia', 'Si');

insert into Solicitudes values (987, 'Instituto San Isidro', 'Tecnologia', 'Si');

insert into Solicitudes values (876, 'Instituto Ramiro de Maeztu', 'Tecnologia', 'No');

insert into Solicitudes values (876, 'Instituto Arturo Soria', 'Ciencias', 'Si');

insert into Solicitudes values (876, 'Instituto Arturo Soria', 'Ciencias Sociales', 'No');

insert into Solicitudes values (765, 'Instituto Ramiro de Maeztu', 'Ciencias Sociales', 'Si');

insert into Solicitudes values (765, 'Instituto Torres Quevedo', 'Ciencias Sociales', 'No');

insert into Solicitudes values (765, 'Instituto Torres Quevedo', 'Ciencias', 'Si');

insert into Solicitudes values (543, 'Instituto Arturo Soria', 'Tecnologia', 'No');

```

### Consulta

La resolución de las consultas se haría de la siguiente forma:

1. SELECT Nombre\_Est, Decision FROM Estudiantes, Solicitudes  
  
WHERE Estudiantes.ID = Solicitudes.ID  
  
AND Valor < 1000 AND Via = 'Tecnologia' AND Nombre\_Inst = 'Instituto Ramiro de Maeztu';
2. SELECT Estudiantes.ID, Nombre\_Est, Nota, Solicitudes.Nombre\_Inst, Plazas FROM Estudiantes, Institutos, Solicitudes WHERE Solicitudes.ID = Estudiantes.ID AND Solicitudes.Nombre\_Inst = Institutos.Nombre\_Inst ORDER BY Puntos DESC, Plazas;
3. SELECT \* FROM Solicitudes WHERE Via like '%Ciencias%';
4. SELECT ID, Nombre\_Est, Puntos, Puntos\*Valor/1000.0 as Ponderada FROM Estudiantes WHERE ABS(Puntos\*(Valor/1000.0) - Puntos) > 1.0;

5. DELETE FROM Solicitudes WHERE ID IN (SELECT ID FROM Solicitudes GROUP BY ID HAVING COUNT (DISTINCT Via) >2);
6. SELECT Via, Puntos FROM Estudiantes, Solicitudes WHERE Estudiantes.ID= Solicitudes.ID GROUP BY Via HAVING MAX(Puntos) < (Select AVG(Puntos) FROM Estudiantes);
7. SELECT DISTINCT Nombre\_Est, Via FROM Estudiantes JOIN Solicitudes ON Estudiantes.ID = Solicitudes.ID;
8. SELECT Nombre\_Est, Puntos FROM Estudiantes JOIN Solicitudes ON Estudiantes.ID = Solicitudes.ID AND Valor < 1000 AND Via='Tecnologia' AND Nombre\_Inst= 'Instituto San Isidro';



## Recursos

### Enlaces de Interés



#### **SQLite Studio**

<https://sqlitestudio.pl/index.rvt?act=download>

Descarga

### Bibliografía

- **Database Systems: The Complete Book** García Molina, H., Ulman, J. D., Widom, J. Database Systems: The Complete Book. Prentice Hall
- **Fundamentals of Database Systems**: Elmasri, R., Navathe, S. B. Fundamentals of Database Systems. Addison Wesley
- **Fundamentos de bases de datos**: Silberschatz, Abraham. Fundamentos de bases de datos. McGraw Hill/Interamericana de España
- **SQLite Language** :

SQLite Language. [En línea] URL disponible en <https://sqlite.org/lang.html>

### Glosario.

- **Atributo**: son aquellos elementos de información de una relación de los cuales se quiere almacenar sus valores.
- **Base de datos relacional**: es un tipo de base de datos que sigue el modelo relacional.
- **Clave alternativa**: son el resto de claves candidatas no elegidas como claves primarias.
- **Clave candidata**: se denomina clave candidata de una relación a una superclave de la relación que hace cumplir que ningún subconjunto propio sea superclave.
- **Clave foránea**: permite establecer conexiones entre las tuplas de varias relaciones.
- **Clave primaria**: entre todas las claves candidatas de una relación, se elige una como la clave cuyos valores se utilizarán para identificar las tuplas de una relación. Esta clave recibe el nombre de clave primaria. El resto de claves candidatas no elegidas se les denomina claves alternativas.
- **Columna**: son aquellos elementos de información de una relación de los cuales se quiere almacenar sus valores.
- **Consulta**: es una sentencia en SQL que permite recuperar o modificar una tabla de una base de datos relacional.
- **Dominio**: es un conjunto de valores de un tipo de datos que son atómicos.

- **Entidad de la clave primaria:** establece que los atributos de la clave primaria de una relación no puedan tener valores nulos. Se debe garantizar el cumplimiento de esta regla en todas las inserciones y modificaciones que afecten a atributos que pertenezcan a la clave primaria de la relación.
- **Esquema:** representa la estructura de la información, indicando qué tipo de información se va a gestionar. Un esquema, a su vez, se compone de un nombre de la relación y de un conjunto de atributos de la relación.
- **Fila:** son los valores concretos que toma una instancia de una relación en los atributos que se han elegido para representar la relación.
- **Instancia:** representa una materialización o particularización de una relación dada.
- **Integridad del dominio:** establece que todos los valores no nulos para un determinado atributo deban ser del dominio declarado para dicho atributo, y que los operadores que se puedan aplicar sobre los valores dependan del dominio de estos valores.
- **Integridad referencial:** establece que todos los valores que tome una clave foránea deban ser valores nulos o valores que existen en la clave primaria que referencia.
- **Modelo relacional:** es un mecanismo de representación de la información que se basa en el concepto de relación.
- **Relación:** representa una entidad abstracta de la cual se quiere almacenar información de manera persistente.
- **SQL:** es el lenguaje que permite manipular una base de datos relacional.
- **SQLiteStudio:** es un sistema de gestión de bases de datos relacionales basado en el lenguaje SQLite. Este lenguaje está constituido por un subconjunto del lenguaje SQL.
- **Superclave:** es un subconjunto de los atributos del esquema tal que no puede haber dos tuplas de la relación que tengan la misma combinación de valores para los atributos del subconjunto.
- **Tabla:** es la representación conceptual en la que se almacenan los datos.
- **Tupla:** son los valores concretos que toma una instancia de una relación en los atributos que se han elegido para representar la relación.
- **Unicidad de la clave primaria:** establece que toda clave primaria de una relación no deba tener valores repetidos. Se debe garantizar el cumplimiento de esta regla en todas las inserciones y modificaciones que afecten a atributos que pertenezcan a la clave primaria de la relación.