

**Fundamentos de bases de datos
relacionales © EDICIONES ROBLE,
S.L.**

Índice

I. Introducción	3
II. Objetivos	4
III. El modelo relacional	5
IV. El lenguaje SQL	11
4.1. Creación de tablas en SQL	11
4.2. Modificación de tablas	16
4.3. Otras operaciones sobre tablas	18
4.4. Índices	19
4.5. Consultas y recuperación de información en SQL	19
4.6. Modificación de tablas en SQL	28
V. SQLitestudio	34
VI. Resumen	43
VII. Caso práctico	44
Recursos	52
Enlaces de Interés	52
Bibliografía	52
Glosario.	52

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. A continuación, se introducirán los elementos principales del lenguaje SQL que permiten a un usuario gestionar una base de datos relacional. También se presentará SQLiteStudio, una aplicación que implementa el lenguaje SQLite —una implementación de SQL ligera muy utilizada en ámbitos como la telefonía móvil— y se mostrará cómo usarla desde Python.

II. Objetivos

Los objetivos que el alumnado alcanzará en esta unidad son:



- 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 que definidas en la relación. En el ejemplo anterior, tanto el grado de la relación como su cardinales es 4 (4 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, Numero}.

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.** 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.** 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:

- Inserciones en una relación que tenga una clave foránea.
- Modificaciones que afecten a atributos que pertenezcan a la clave foránea de una relación.
- Borrados en relaciones referenciadas por otras relaciones.
- 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:

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

- Restricción. Consiste en no aceptar la operación de actualización siempre que afecte a una clave primaria referenciada por una clave foránea.
- Actualización en cascada. Se permite la operación de actualización y se lleva las mismas operaciones sobre las tuplas que las referencian en otras relaciones.
- 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. 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:

- Es un lenguaje declarativo basado en el álgebra relacional.
- Está soportado por la mayoría de los sistemas relacionales comerciales.
- Se puede utilizar de manera interactiva o embebido en un programa.

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

- Relación.
- Atributos.
- Tuplas.

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

- Tablas.
- Columnas.
- Filas.

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

4.1. 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.

a) Tipos de datos

Los principales tipos de datos predefinidos en SQL que pueden asociarse a una columna aparecen en la figura 3.1.

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 3.1. Restricciones por tabla.

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

- YEAR (0001..9999)
- MONTH (01..12)
- DAY (01..31)
- HOUR (00..23)
- MINUT (00..59)
- 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'.

b) 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 (figura 3.2.).

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

Figura 3.2. Funciones para definir valores por defecto.

c) 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 (figura 3.3.):

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

Figura 3.3. Restricciones de columna.

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

[CONSTRAINT nombre_restricción] CHECK (condiciones).

d) 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 (figura 3.4.):

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

Figura 3.4. Restricciones por tabla.

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

[CONSTRAINT nombre_restricción] CHECK (condiciones).

e) Ejemplos de creación de tablas

Create table sucursal

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

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));

Create table cuenta

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

Create table impositor

```
(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)
```

f) 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 (figura 3.5.):

```
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)

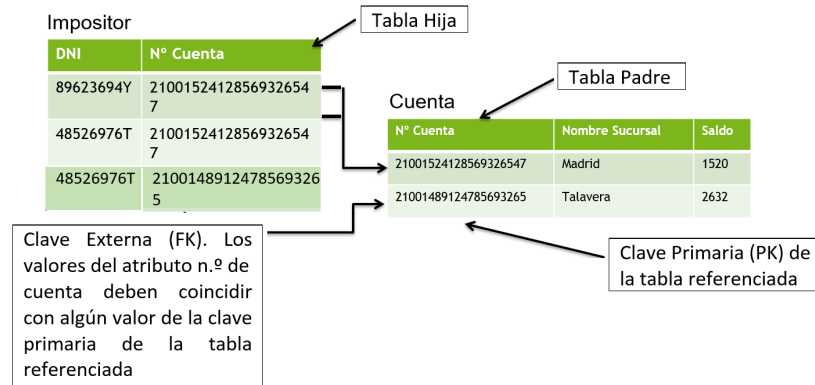


Figura 3.5. Ejemplo de claves foráneas.

4.2. 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



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

ALTER TABLE cuenta ADD fecha_apertura DATE;

ALTER TABLE cuenta DROP COLUMN nombre_sucursal;

ALTER TABLE cuenta MODIFY comision DEFAULT 1.5;

ALTER TABLE cliente MODIFY nombre_cliente NULL;

ALTER TABLE sucursal ADD CONSTRAINT cd_UK UNIQUE(ciudad);

4.3. 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
```

4.4. 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 3.6., se muestra un ejemplo.

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

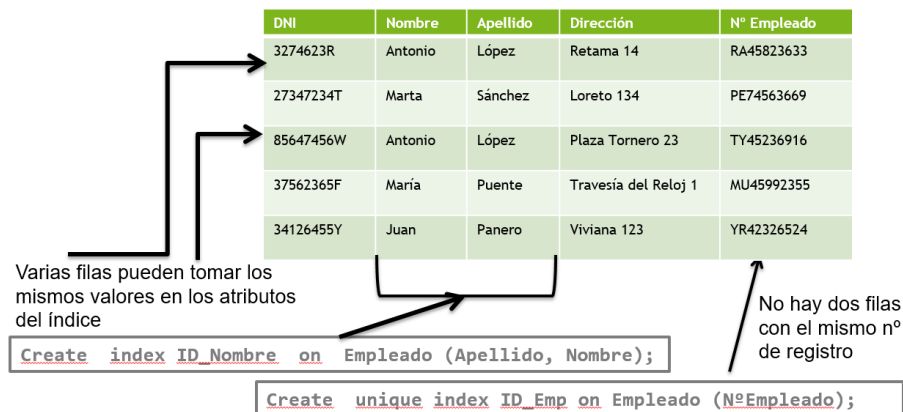


Figura 3.6. Índices.

4.5. Consultas y recuperación de información en SQL

4.5.1. Consultas sobre una sola tabla

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.

Por ejemplo, si se quieren seleccionar las columnas código, nombre, dirección y ciudad de la tabla clientes, se usaría la sentencia:

→ *SELECT codigo_cli, nombre_cli, direccion, ciudad FROM clientes;*

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 clientes;*

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];*

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 sueldos de la tabla empleados:

```
SELECT DISTINCT sueldo FROM empleados;
```

Y si se quieren recuperar los empleados de la tabla empleados, cuyo sueldo es mayor de 1000 euros:

```
SELECT * FROM empleados WHERE sueldo > 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 quiere obtener los proyectos de la tabla proyectos que se corresponden con un cliente que tiene como NIF el número "444555-E":

```
SELECT * FROM proyectos WHERE codigo_cliente = (SELECT código_cli FROM clientes WHERE nif="444555-E")
```

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 clientes que viven en Madrid y Zaragoza:

*SELECT * FROM clientes 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;

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 clientes cuya ciudad de residencia termina por la letra d:

*SELECT * FROM clientes WHERE ciudad LIKE '%d';*

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

*SELECT * FROM clientes 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 clientes que no tienen un número de teléfono:

- *SELECT * FROM clientes 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 empleados que están asignados a algún proyecto:

- *SELECT * FROM empleados WHERE EXISTS (SELECT * FROM proyectos WHERE codigo_proyec = num_proyec);*

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;

Por ejemplo, si se quiere recuperar todos los proyectos en los que los sueldos de todos los empleados asignados son menores que el precio del proyecto:

- *SELECT * FROM proyectos WHERE precio > ALL (SELECT sueldo FROM empleados WHERE codigo_proyec = num_proyec);*

Si la condición es más laxa, y solo se pide que la condición únicamente se dé para algunos empleados, entonces sería:

- *SELECT * FROM proyectos WHERE precio > SOME (SELECT sueldo FROM empleados WHERE codigo_proyec = num_proyec);*

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_empl, sueldo DESC;
```

4.5.2. 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, se quiere obtener el NIF del cliente y el precio de los proyectos desarrollados para el cliente con código 30.

```
SELECT p.precio, c.nif FROM clientes c JOIN proyectos p ON c.codigo_cli = p.codigo_cliente WHERE c.codigo_cli = 30;
```

Alternativamente se podría obtener con la siguiente consulta:

```
→ SELECT p.precio, c.nif FROM clientes c, proyectos p WHERE c.codigo_cli = p.codigo_cliente AND c.codigo_cli = 20;
```



Por ejemplo, si se quieren los códigos de los proyectos que son más caros que el proyecto con código 30:

```
SELECT p1.codigo_proyec FROM proyectos p1 JOIN proyectos p2 ON p1.precio > p2.precio WHERE p2.codigo_proyec = 30;
```

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 los empleados cuyo departamento se encuentra situado en Madrid:

```
SELECT * FROM empleados NATURAL JOIN departamentos WHERE ciudad = 'Madrid';
```

De forma equivalente se podría consultar:

```
→ SELECT * FROM empleados JOIN departamentos USING (nombre_dep, ciudad_dep) WHERE ciudad = 'Madrid';
```

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 empleados, proyectos y clientes:

```
SELECT * FROM empleados, proyectos, clientes WHERE num_proyec =
codigo_proyec AND codigo_cliente = codigo_cli;
```

O bien:

```
SELECT * FROM (empleados JOIN proyectos ON num_proyec =
codigo_proyec) JOIN clientes ON codigo_cliente = codigo_cli;
```

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** clientes **UNION** **SELECT** ciudad_dep **FROM** departamentos;

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, si se quiere saber las ciudades de los clientes en las que hay departamentos:

→ Usando la intersección.

SELECT ciudad FROM clientes INTERSECT SELECT ciudad_dep FROM departamentos;

→ Usando IN

SELECT c.ciudad FROM clientes c WHERE c.ciudad IN (SELECT d.ciudad_dep FROM departamentos d);

→ Usando EXISTS

*SELECT c.ciudad FROM clientes c WHERE EXISTS (SELECT * FROM departamentos d WHERE c.ciudad = d.ciudad_dep;*

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 clientes en las que no hay departamentos:

→ Usando la intersección.

```
SELECT ciudad FROM clientes EXCEPT SELECT ciudad_dep FROM departamentos;
```

→ Usando IN

```
SELECT c.ciudad FROM clientes c WHERE c.ciudad NOT IN (SELECT d.ciudad_dep FROM departamentos d);
```

→ Usando EXISTS

```
SELECT c.ciudad FROM clientes c WHERE NOT EXISTS (SELECT * FROM departamentos d WHERE c.ciudad = d.ciudad_dep);
```

4.6. Modificación de tablas en SQL

4.6.1 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>};
```

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 clientes que tiene las columnas :nif, nombre_cli, codigo_cli, telefono, direccion, ciudad, se podría hacer de dos formas:

```
INSERT INTO clientes VALUES (10, 'Mercadona', '122233444-C', 'Gran vida 8', 'Madrid', DEFAULT);
```

O bien:

```
INSERT INTO clientes(nif, nombre_cli, codigo_cli, telefono, direccion, ciudad) VALUES ('122233444-C', 'Mercadona', 10, DEFAULT, 'Gran vida 8', 'Madrid');
```

Insertar un préstamo en la relación Préstamo:

```
INSERT INTO Prestamo VALUES ('Navacerrada', 'Pepe Pérez', 125.000)
```

También es posible obtener los datos mediante una consulta SELECT que actúe como proveedor de datos:

```
INSERT INTO Prestamo SELECT * FROM Nuevos_Prestamos
```

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 proyectos se usaría la sentencia:

```
DELETE FROM proyectos;
```

Sin embargo, si solo se quieren borrar las filas de la tabla en las que el valor de la columna cliente vale 12, entonces se usaría la sentencia:

```
DELETE FROM proyectos WHERE codigo_cliente = 12;
```

Por ejemplo, para borrar todos los clientes que tengan un préstamo no registrado en la relación Préstamo:

```
DELETE FROM Clientes WHERE Clientes.NumPrestamo NOT IN (SELECT NumPrestamo FROM Prestamo)
```

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 = {expr|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 del proyecto 2 en 500 euros:

```
UPDATE empleados SET sueldo = 500 WHERE num_proyec = 2;
```

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

```
UPDATE Prestamo SET importe=200.000 WHERE NumPrestamo='P-170'
```

La cláusula WHERE admite consultas anidadas. Por ejemplo "Modificar todos los prestamos cuya sucursal ha sido cerrada a la sucursal 'Centro'"

```
UPDATE Prestamo SET sucursal= 'Centro' WHERE sucursal IN (SELECT sucursal FROM Sucursales_Cerradas)
```

4.6.2 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 3.7.

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 3.7. 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_clie FROM clientes 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.

Por ejemplo, si se quiere conocer el importe total de los proyectos agrupados por clientes:

```
SELECT código_cliente, SUM(precio) AS importe FROM clientes GROUP BY código_cliente;
```

Y si solo queremos aquellos clientes con un importe facturado mayor de 10000 euros:

```
SELECT código_cliente FROM clientes GROUP BY código_cliente HAVING SUM(precio)>10000
```

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:

clientes					
codigo_cli	nombre_cli	nif	dirección	ciudad	teléfono
10	Carrefour	38.567.893-C	Gran vía 11	Madrid	NULL
20	El Corte Inglés	38.123.898-E	Plaza de España 22	Zaragoza	976 45 56 78
30	Mercadona	36.432.127-A	Begoña, 33	Bilbao	940 34 56 90

Figura 3.8. Tabla clientes.

pedidos				
código_pedido	precio	fecha_pedido	fecha_entrega	codigo_cliente
1	1,0E+6	1-1-98	1-1-99	10
2	2,0E+6	1-10-96	31-3-98	10
3	1,0E+6	10-2-98	1-2-99	20

Figura 3.9. Tabla pedidos.

Si se quiere crear una vista que indique para cada cliente el número de pedidos que tiene encargado, se definiría la vista:

```
CREATE VIEW pedidos_por_cliente (codigo_cli, num_pedidos) AS (SELECT c.codigo_cli, COUNT(*)
FROM pedidos p, clientes c WHERE p.codigo_cliente = c.codigo_cli GROUP BY c.codigo_cli);
```

Y se obtendría la vista (figura 3.10.):

pedidos_por_clientes	
codigo_cli	num pedidos
10	2
20	1
30	1

Figura 3.10. Vista resultante.

V. SQLitestudio

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



<http://sqlitestudio.pl/?act=download> (figura 3.11.).



Figura 3.11. Zona de descarga de SQLiteStudio.

SQLiteStudio es un sistema de gestión de bases de datos relacionales basado en SQLite que implementa un subconjunto del estándar SQL. Se puede consultar más información sobre SQLite en la siguiente dirección:



<http://www.sqlite.org/lang.html>

La descarga del enlace anterior es un fichero ejecutable. Si se pulsa sobre el ejecutable, aparecerá la interface principal (figura 3.12.):

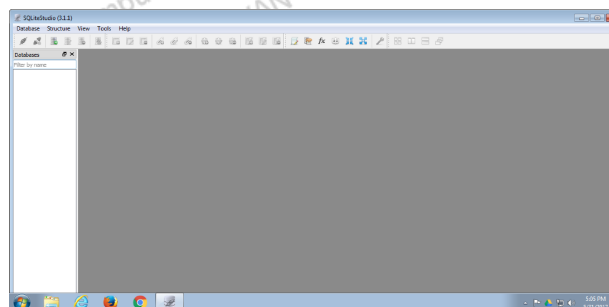


Figura 3.12. Interface principal de SQLiteStudio.

Para ilustrar algunas de las funcionalidades de SQLiteStudio, se va a considerar una base de datos para gestionar una empresa de arquitectura que tiene sedes en Madrid, Zaragoza y Granada. La información que se desea almacenar es:

Información a mostrar

Sobre los empleados: código de empleado, nombre y apellido, sueldo, nombre de la ciudad y de su departamento y el número de proyecto al que están asignados.

Sobre cada departamento: nombre, ciudad donde se encuentran y teléfono. Un departamento con el mismo nombre puede estar en ciudades diferentes y, en una misma ciudad, puede haber departamentos con nombres diferentes.

Sobre los proyectos: código, nombre, precio, fecha de inicio, fecha prevista de finalización, fecha real de finalización y el código de cliente.

Sobre los clientes: código de cliente, el nombre, el NIF, la dirección, la ciudad y el teléfono.

Creación de la base de datos

En primer lugar, hay que crear la base de datos que contendrá las tablas que almacenarán la información:

Se pulsa sobre “Database”.

En el desplegable que aparece se selecciona “Add database” (figura 3.13.).

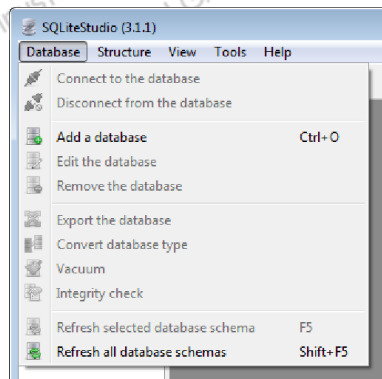


Figura 3.13. Figura Add database.

Tipo de base de datos

Aparece un formulario (figura 3.14.) 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 “Empresa.sqlite3”.

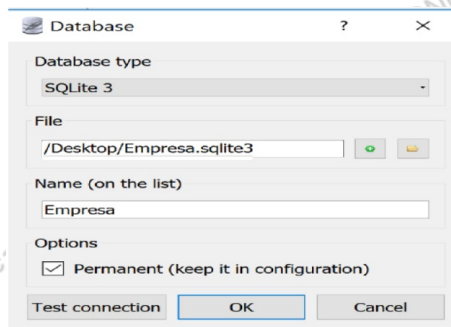


Figura 3.14. Formulario de configuración.

Base de datos en la interface principal

Cuando se pulsa sobre “OK”, la nueva base de datos aparece en el marco izquierdo de la interface principal (figura 3.15.).

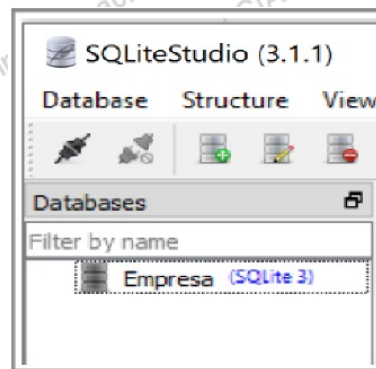


Figura 3.15. Base de datos creada.

Conectar a la base de datos

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 3.16.).

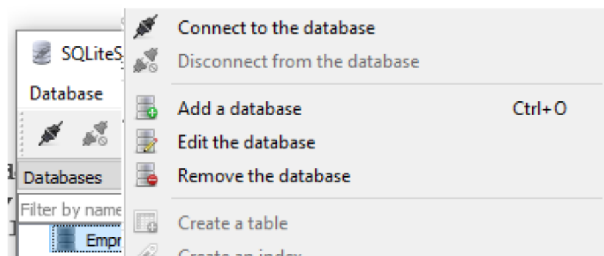


Figura 3.16. Se pulsa sobre "Connect to the database".

Ver las tablas asociadas

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 3.17.).

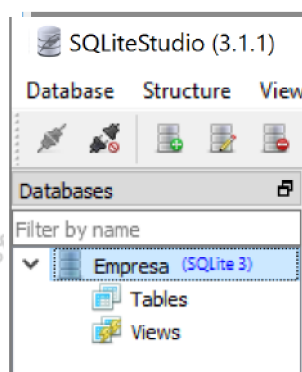


Figura 3.17. Recursos asociados con la base de datos.

Abrir el editor SQL

Para interactuar con la base de datos, se debe abrir el editor de SQL, pulsando sobre el icono en forma de lápiz (figura 3.18.).



Figura 3.18. Acceso al editor de SQL.

Posición del editor

El editor de SQL se muestra en la parte derecha de la interface principal (figura 3.19.).

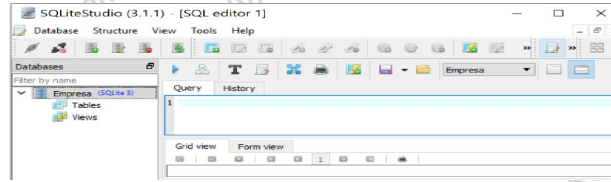



Figura 3.19. Editor de SQL.

Creación de tablas

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

Se crea una tabla clientes para almacenar la información sobre los mismos. Para ello se introduce la siguiente sentencia en el editor (figura 3.20.):

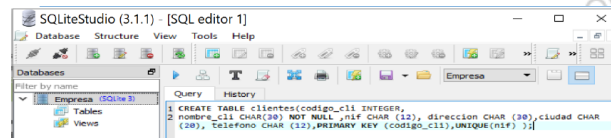


Figura 3.20. Sentencia de SQL para crear tablas.

```
CREATE TABLE clientes (codigo_cli INTEGER,
nombre_cli CHAR (30) NOT NULL ,nif CHAR (12), direccion CHAR (30),ciudad CHAR (20), telefono CHAR
(12),PRIMARY KEY (codigo_cli),UNIQUE(nif) );
```

Ejecutar sentencias

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 3.21.).

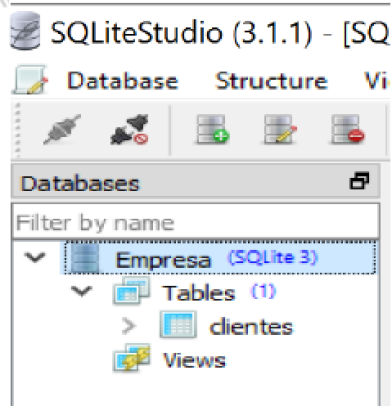


Figura 3.21. Tabla creada.

Navegación

Si se pulsa sobre la tabla, aparece en la parte derecha de la interface la definición de la tabla creada (figura 3.22.).

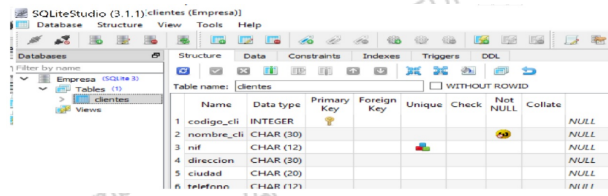


Figura 3.22. Definición de la tabla creada.

Podéis navegar por las diferentes pestañas ("Structure", "Data"...) y sobre los iconos inferiores

() para investigar su utilidad.

Creación del resto de tablas

A continuación, se pueden crear el resto de tablas de la base de datos de ejemplo de la misma forma:

Tabla departamentos

```
CREATE TABLE departamentos
(nombre_dep CHAR(20),
ciudad_dep CHAR(20),
telefono INTEGER DEFAULT NULL,
PRIMARY KEY (nombre_dep, ciudad_dep));
```

Tabla proyectos

```
CREATE TABLE proyectos
(codigo_proyec INTEGER,nombre_proyec CHAR(20),
precio REAL, fecha_inicio DATE, fecha_prev_fin DATE,
fecha_fin DATE DEFAULT NULL,codigo_cliente INTEGER,
PRIMARY KEY (codigo_proyec),
CHECK (fecha_inicio < fecha_prev_fin),
CHECK (fecha_inicio < fecha_fin),
FOREIGN KEY ( codigo_cliente ) REFERENCES clientes ( codigo_cli));
```

Tabla empleados

```
CREATE TABLE empleados
(codigo_empl INTEGER,nombre_empl CHAR (20),
apellido_empl CHAR(20),sueldo REAL CHECK (sueldo > 7000),
nombre_dep CHAR(20),ciudad_dep CHAR(20),
num_proyec INTEGER,PRIMARY KEY (codigo_empl),
FOREIGN KEY (nombre_dep, ciudad_dep) REFERENCES
departamentos (nombre_dep, ciudad_dep),
FOREIGN KEY (num_proyec) REFERENCES proyectos (codigo_proyec) );
```


Resultado

El resultado final debería ser similar al siguiente (figura 3.23.):

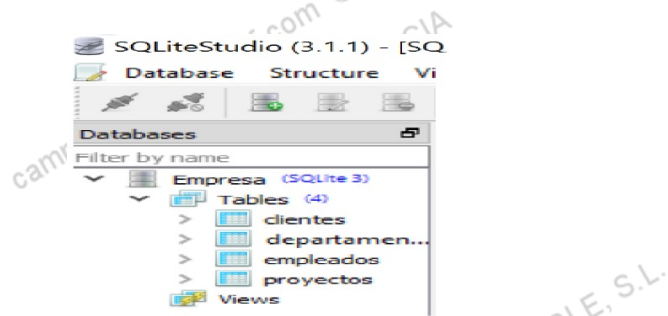


Figura 3.23. Tablas creadas en la base de datos.

Hay que tener en cuenta que, al crear una tabla, las restricciones se pueden definir a nivel de columna o bien a nivel de tabla. Así, en el caso de que la restricción haga referencia a una sola columna, se puede optar por una opción u otra. Sin embargo, si hace referencia a más de una columna, es necesario definir la restricción a nivel de tabla.

Creación de una vista

A continuación, se va a crear una vista utilizando las tablas proyectos y clientes (figura 3.24.). Para ello, se usa el editor de SQL de la misma forma que en el caso de la creación de tablas.

```
CREATE VIEW proyectos_por_cliente AS
```

```
SELECT c.codigo_cli, COUNT(*)
```

```
FROM proyectos p, clientes c
```

```
WHERE p.codigo_cliente = c.codigo_cli
```

```
GROUP BY c.codigo_cli;
```

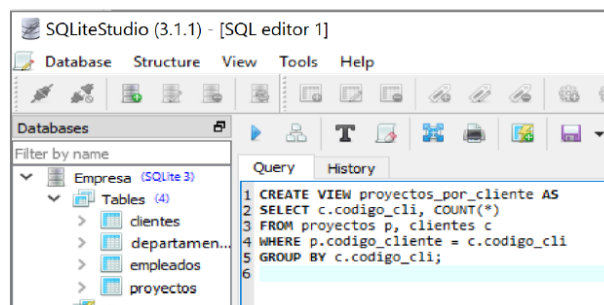


Figura 3.24. Creación de una vista.

Resultado de la ejecución

El resultado de la ejecución debería ser similar a la siguiente (figura 3.25.):

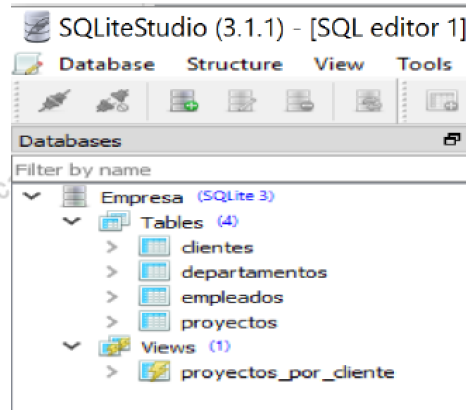


Figura 3.25. Resultado de la ejecución de la vista.

Otras pruebas

Por último, se puede probar lo siguiente:

- Eliminar la vista proyectos_por_cliente usando la sentencia: DROP VIEW proyectos_por_cliente.
- Eliminar la tabla clientes usando la sentencia: DROP TABLE clientes.
- Añadir una nueva columna denominada "país" de tipo CHAR(20) a la tabla clientes: ALTER TABLE clientes ADD COLUMN pais CHAR(20)-

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:

- Sobre los institutos: nombre, área de la ciudad donde se encuentra y número máximo de plazas.
- Sobre los estudiantes: identificador de estudiante, nombre, puntos que tiene para acceder y un valor de corrección.
- Sobre las solicitudes: identificador de estudiante, nombre del instituto, vía solicitada y decisión sobre la solicitud.

Tablas

Se van a crear 3 tablas:

Institutos

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

Estudiantes

```
CREATE TABLE Estudiantes(ID INTEGER, Nombre_Est CHAR(35), Puntos REAL, Valor INTEGER,
PRIMARY KEY ( ID ),
UNIQUE ( ID ) );
```

Solicitudes

```
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 ) );
```

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', 'Tecnologia', '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:

- ➔ `SELECT Nombre_Est, Puntos, Decision FROM Estudiantes, Solicitudes WHERE Estudiantes.ID = Solicitudes.ID AND Valor < 1000 AND Via = 'Tecnologia' AND Nombre_Inst = 'Instituto Ramiro de Maeztu';`
- ➔ `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;`
- ➔ `SELECT * FROM Solicitudes WHERE Via like '%Ciencias%';`
- ➔ `SELECT ID, Nombre_Est, Puntos, Puntos*Valor/1000.0 as Ponderada FROM Estudiantes WHERE ABS(Puntos*(Valor/1000.0) - Puntos) > 1.0;`
- ➔ `DELETE FROM Solicitudes WHERE ID IN (SELECT ID FROM Solicitudes GROUP BY ID HAVING COUNT (DISTINCT Via) >2);`
- ➔ `SELECT Via, Puntos FROM Estudiantes, Solicitudes WHERE Estudiantes.ID= Solicitudes.ID GROUP BY Via HAVING MAX(Puntos) < (Select AVG(Puntos) FROM Estudiantes);`
- ➔ `SELECT DISTINCT Nombre_Est, Via FROM Estudiantes JOIN Solicitudes ON Estudiantes.ID = Solicitudes.ID;`
- ➔ `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';`

x

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 clientes UNION SELECT ciudad_dep FROM departamentos;`

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, si se quiere saber las ciudades de los clientes en las que hay departamentos:

→ Usando la intersección.

```
SELECT ciudad FROM clientes INTERSECT SELECT ciudad_dep FROM departamentos;
```

→ Usando IN

```
SELECT c.ciudad FROM clientes c WHERE c.ciudad IN (SELECT d.ciudad_dep FROM departamentos d);
```

→ Usando EXISTS

```
SELECT c.ciudad FROM clientes c WHERE EXISTS (SELECT * FROM departamentos d WHERE c.ciudad = d.ciudad_dep);
```

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 clientes en las que no hay departamentos:

→ Usando la intersección.

```
SELECT ciudad FROM clientes EXCEPT SELECT ciudad_dep FROM departamentos;
```

→ Usando IN

```
SELECT c.ciudad FROM clientes c WHERE c.ciudad NOT IN (SELECT d.ciudad_dep FROM departamentos d);
```

→ Usando EXISTS

```
SELECT c.ciudad FROM clientes c WHERE NOT EXISTS (SELECT * FROM departamentos d WHERE c.ciudad = d.ciudad_dep;
```

x

Las funciones de agregación son funciones que permiten realizar operaciones sobre los datos de una columna. Algunas funciones se muestran en la figura 3.7.

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 3.7. 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_clie FROM clientes 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.

Por ejemplo, si se quiere conocer el importe total de los proyectos agrupados por clientes:

```
SELECT código_cliente, SUM(precio) AS importe FROM clientes GROUP BY código_cliente;
```

Y si solo queremos aquellos clientes con un importe facturado mayor de 10000 euros:

```
SELECT código_cliente FROM clientes GROUP BY código_cliente HAVING SUM(precio)>10000
```

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:

clientes					
codigo_cli	nombre_cli	nif	dirección	ciudad	teléfono
10	Carrefour	38.567.893-C	Gran vía 11	Madrid	NULL
20	El Corte Inglés	38.123.898-E	Plaza de España 22	Zaragoza	976 45 56 78
30	Mercadona	36.432.127-A	Begoña, 33	Bilbao	940 34 56 90

Figura 3.8. Tabla clientes.

pedidos				
código_pedido	precio	fecha_pedido	fecha_entrega	codigo_cliente
1	1,0E+6	1-1-98	1-1-99	10
2	2,0E+6	1-10-96	31-3-98	10
3	1,0E+6	10-2-98	1-2-99	20

Figura 3.9. Tabla pedidos.

Si se quiere crear una vista que indique para cada cliente el número de pedidos que tiene encargado, se definiría la vista:

```
CREATE VIEW pedidos_por_cliente (codigo_cli, num_pedidos) AS (SELECT c.codigo_cli, COUNT(*)
FROM pedidos p, clientes c WHERE p.codigo_cliente = c.codigo_cli GROUP BY c.codigo_cli);
```

Y se obtendría la vista (figura 3.10.):

pedidos_por_clientes	
codigo_cli	num pedidos
10	2
20	1
30	1

Figura 3.10. Vista resultante.

Recursos

Enlaces de Interés



<http://sqlitestudio.pl/?act=download>

<http://sqlitestudio.pl/?act=download>

Descarga SQLiteStudio



<http://www.sqlite.org/lang.html>

<http://www.sqlite.org/lang.html>

Información de SQLiteStudio

Bibliografía

- Elmasri, R., Navathe, S. B. *Fundamentals of Database Systems*. Addison Wesley; 2010, 6.ª ed.:
- García Molina, H., Ulman, J. D., Widom, J. *Database Systems: The Complete Book* Prentice Hall; 2009, 2ª ed. :
- Silberschatz, Abraham. *Fundamentos de bases de datos*. McGraw Hill/Interamericana de España; 2006, 5ª ed. :
- SQLite Language. [En línea] URL disponible en <https://sqlite.org/lang.html> :

Glosario.

- **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 o atributo**: 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.
- **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 o fila:** 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.