



Fundamentos de bases de datos relacionales

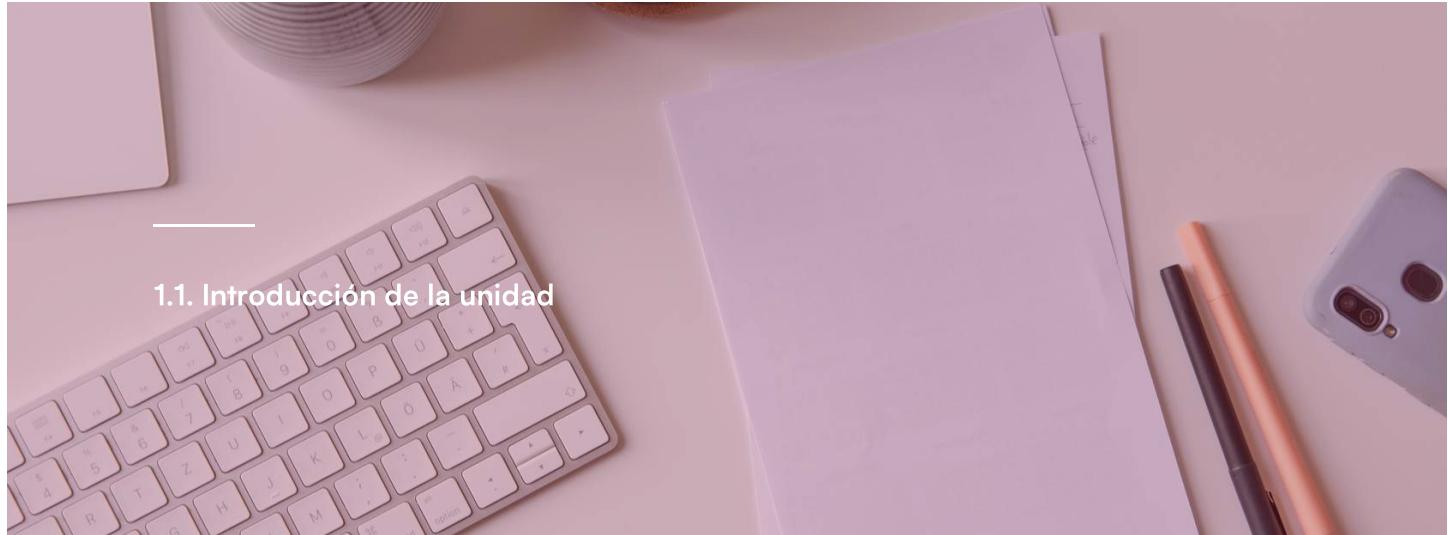


- ≡ I. Introducción
- ≡ II. Objetivos
- ≡ III. El modelo relacional
- ≡ IV. SQLiteStudio
- ≡ V. El lenguaje SQL
- ≡ VI. Resumen
- ≡ VII. Caso práctico con solución
- ≡ VIII. Lecturas recomendadas
- ≡ IX. Enlaces de interés
- ≡ X. Glosario
- ≡ XI. Bibliografía

QUESTION BANKS

I. Introducción

1.1. Introducción de la unidad



En las últimas décadas, el mecanismo más utilizado para **almacenar la información en los sistemas informáticos** han 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.

El modelo relacional, planteado por **Edgar Codd** en el año 1970, **sigue vigente a día de hoy**. Una de las claves de su éxito es la existencia de un **lenguaje de consulta, SQL**, que permite extraer datos de una base de datos de manera sencilla y muy versátil. Pero lo más interesante es que, independientemente del motor de base de datos, **el lenguaje se mantiene prácticamente inalterado**; “prácticamente” porque cada sistema ha incorporado sus funcionalidades propias que han hecho que el lenguaje diverja en algunos aspectos, aunque la esencia sigue siendo la misma y pasar de un sistema a otro no requiere demasiado esfuerzo.

Por otro lado, dentro del fenómeno **big data**, han surgido nuevos **modelos de persistencia de datos** denominados genéricamente **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. Por ello, **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**.

Asimismo, se presentará **SQLiteStudio**, una aplicación que **implementa el lenguaje SQLite** (basado en SQL), sobre la cual se realizarán los ejercicios de la unidad. SQLite **es un sistema de base de datos que no requiere una infraestructura compleja ni una aplicación corriendo de forma continuada en modo servidor**. SQLite es simplemente **una base de datos almacenada en un fichero**, siguiendo un formato que permite su consulta de forma eficiente.

Esto ha convertido a SQLite **en una de las bases de datos preferida para el desarrollo de apps móviles y para dispositivos IoT**, puesto que solo necesita unos pocos bytes de almacenamiento.

Por otro lado, SQLiteStudio **es una aplicación con una interfaz de usuario que permite acceder a una base de datos SQLite y ejecutar consultas de cierta complejidad**.

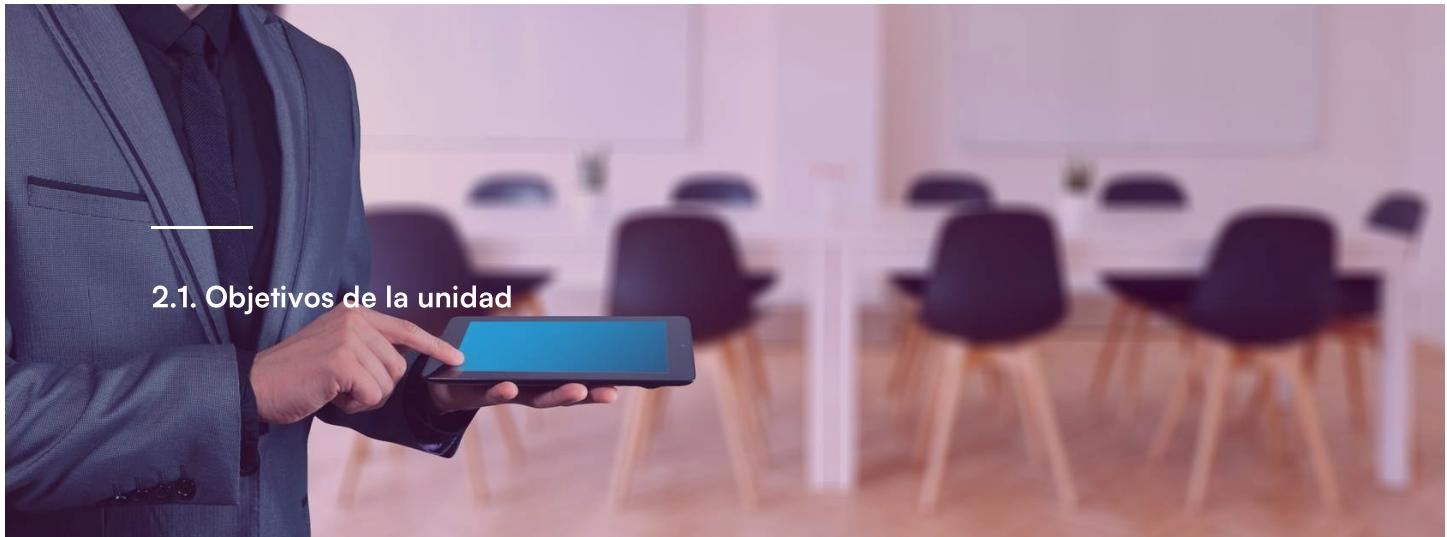
A continuación, se introducirán los **elementos principales del lenguaje SQL** que permiten a un usuario gestionar una base de datos relacional.

Se mostrará **cómo realizar las operaciones más habituales en una base de datos**, que son las **operaciones CRUD (create, read, update y delete)**, utilizando la sintaxis que ofrece el lenguaje SQL. Es decir, no solo se va a utilizar SQL para extraer datos, sino que ofrece la **posibilidad de realizar operaciones de edición, no solo de los datos, sino de toda la estructura de la base de datos subyacente**.

Consejo:

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



2.1. Objetivos de la unidad

Los objetivos de esta unidad son:

- 1 Mostrar los principios fundamentales del modelo relacional en el que se basan las bases de datos relacionales.
- 2 Estudiar cómo diseñar, crear y explotar una base de datos relacional usando SQL.
- 3 Usar un sistema de gestión de bases de datos relacionales concreto (SQLiteStudio).
- 4 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**.

CONTINUAR

3.1. Dominio, atributo e instancia

DOMINIO

ATRIBUTO

INSTANCIA

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...

DOMINIO

ATRIBUTO

INSTANCIA

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.

DOMINIO	ATRIBUTO	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".

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 dos 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).

CONTINUAR

3.2. Claves: conceptos y tipos

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 de forma única las tuplas de una relación**.

Saber más

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}...

Se denomina **clave candidata** de una relación a **una superclave de la relación que cumple que ningún subconjunto propio sea superclave**. Es decir, **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.

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 las claves candidatas no elegidas se denominan 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.

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}.

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}.

EMPLEADOS(DNI, Nombre, Apellidos, DNIJefe, PlantaDespacho, NumDespacho)

DESPACHOS(Planta, Número, Plazas)

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, qué otro empleado 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.

CONTINUAR

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

CONSULTA

Permite **modificar la información representada en una relación**. Puede 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.

ACTUALIZACIÓN

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 relacionales¹.

¹[CRUD operations explained](#).

CONTINUAR

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 —

En una relación entre dos tablas, donde la clave primaria de una es la clave foránea en la otra, el dato de la clave foránea debe existir en la clave primaria de la otra tabla. Es decir, todos los valores que toma una clave foránea deben ser nulos o valores que existen en la clave primaria que referencia (figura 1).

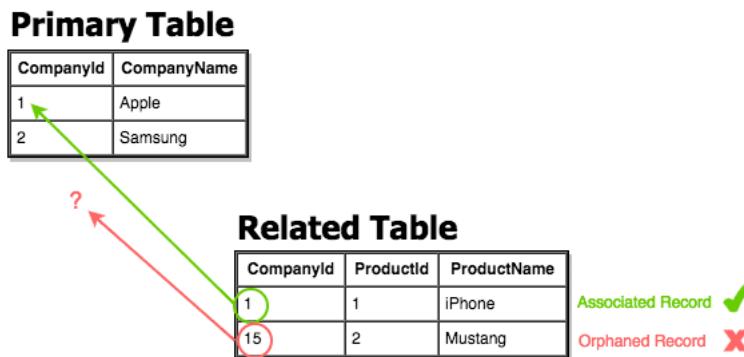


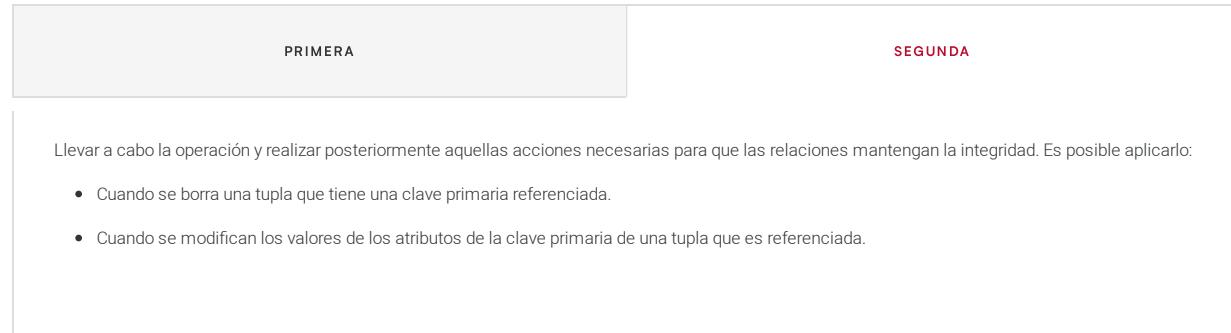
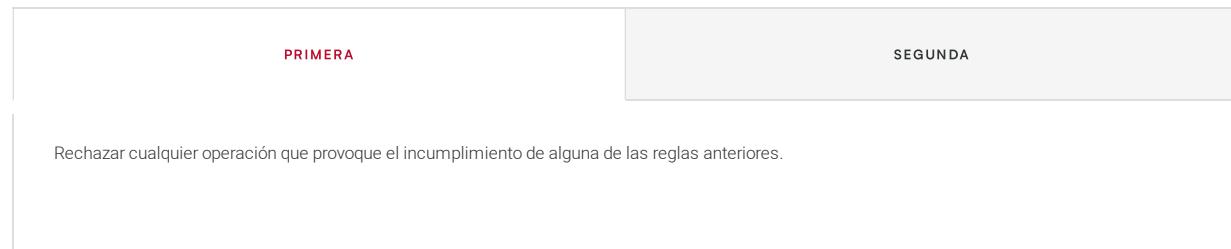
Figura 1. Integridad referencial.

Fuente: <https://database.guide/>

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.

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



Para los casos anteriores existen **tres 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 a 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.

CONTINUAR

3.5. Bases de datos relacionales

Hasta el momento se ha visto el **modelo relacional**, que es un modelo teórico. En la práctica existen multitud de motores de bases de datos que implementan este modelo, algunos de ellos de código libre y otros como servicios de pago.

En el ámbito profesional, algunos de los más utilizados son **PostgreSQL**, **MySQL**, **Microsoft SQL** y **Oracle**².

²[Top SQL engines as of 2021](#).

Para facilitar la comprensión y práctica del lenguaje SQL, en esta unidad se trabaja con **SQLite**, que es un motor de base de datos que se puede usar en un entorno local, sin necesidad de utilizar un proceso servidor que actúe de intermediario entre la aplicación de consulta y los datos.

Saber más

SQLite se utiliza mucho en *smartphones* y dispositivos IoT, dada su ligereza y robustez en entornos locales

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](#) (figura 2).

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.



3.3.3 released!

Rather small, yet important bugfix release. Includes update to the most recent SQLite (3.35.4) and brings back independent SQLite library file to allow user manual updates if necessary.

[Read More →](#)

Posted on 12 April 2021

[More news →](#)

Figura 2. Zona de descarga de SQLiteStudio.

Fuente: elaboración propia.

Saber más

Se puede consultar más información sobre SQLite en la siguiente dirección: <http://www.sqlite.org/lang.html>

Habrá que descargar la **versión portable**, desempaquetar en la ruta que se quiera, pulsar en el archivo "SQLiteStudio.exe" y aparecerá la interfaz principal (figura 3):

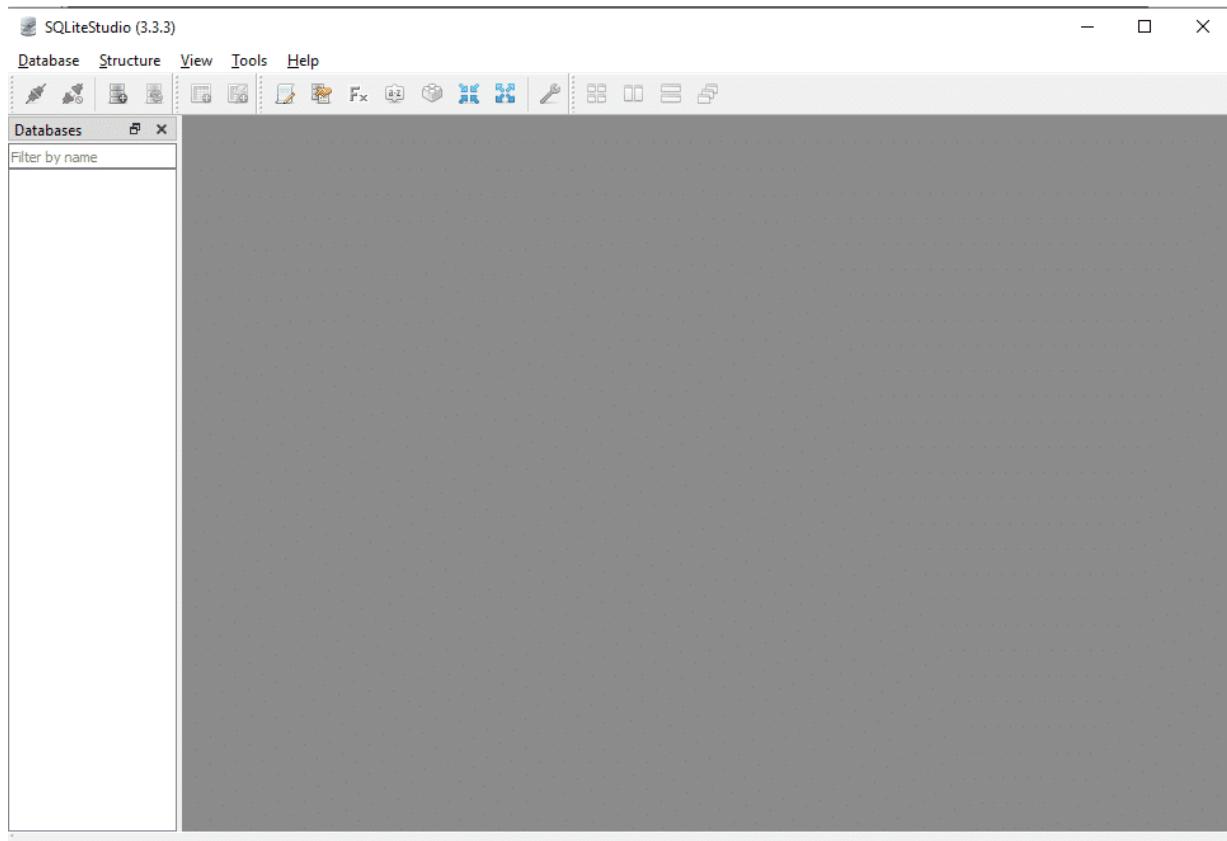


Figura 3. Interfaz principal de SQLiteStudio.
Fuente: elaboración propia

CONTINUAR

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.

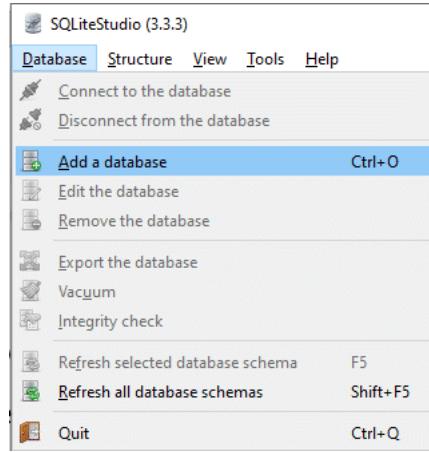


Figura 4. Seleccionar *Add a database*.

Fuente: elaboración propia.

En primer lugar, hay que crear la base de datos:

- Se pulsa sobre *Database*.
- En el desplegable que aparece, se selecciona *Add a database* (figura 4).

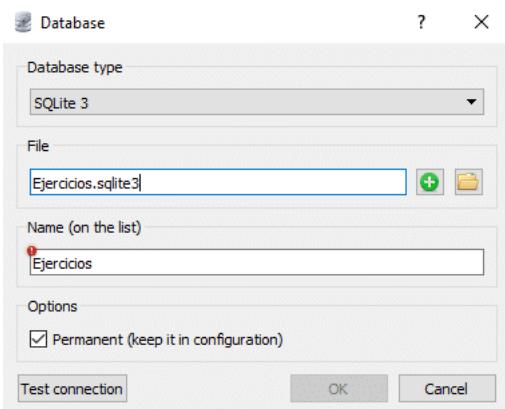


Figura 5. Formulario de configuración.

Fuente: elaboración propia.

Aparece un formulario (figura 5) en el que se debe indicar el tipo de base de datos (sección *Database type*), dónde 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”.

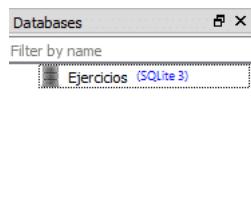


Figura 6. Base de datos creada.

Fuente: elaboración propia.

Cuando se pulsa sobre *OK*, la nueva base de datos aparece en el marco izquierdo de la interfaz principal (figura 6).

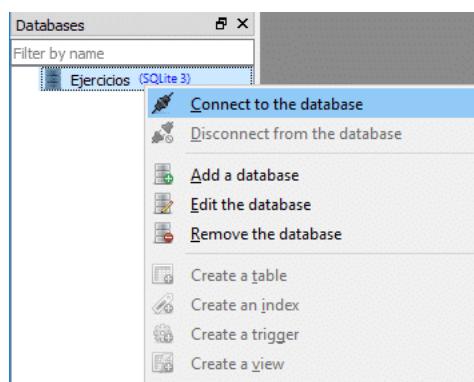


Figura 7. Se pulsa sobre Connect to the database.

Fuente: elaboración propia.

Para poder usar la base de datos, se selecciona con el ratón y, con el botón derecho, aparece un desplegable en el que se elige *Connect to the database* (figura 7).



Figura 8. Recursos asociados con la base de datos.

Fuente: elaboración propia.

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 8).

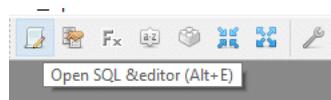


Figura 9. Acceso al editor de SQL.

Fuente: elaboración propia.

Para interactuar con la base de datos, hay que abrir el editor de SQL pulsando sobre el ícono en forma de lápiz (figura 9).

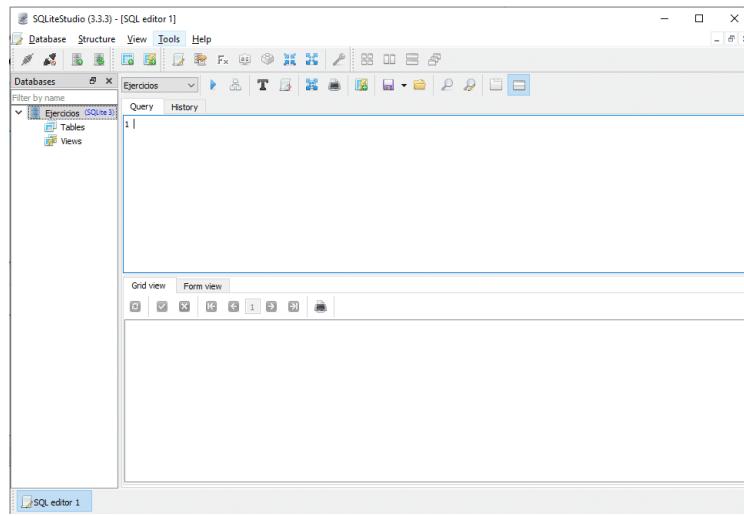


Figura 10. Editor de SQL. Fuente: elaboración propia.

El editor de SQL se muestra en la parte derecha de la interfaz principal (figura 10). Este editor será necesario para los ejercicios del siguiente tema.

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 de consulta 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 **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.

CONCEPTOS

CONCEPTOS SIMILARES

En el modelo relacional se estructura la información en función de los conceptos:

- Relación.
- Atributos.
- Tuplas.

CONCEPTOS

CONCEPTOS SIMILARES

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

- Tablas.
- Columnas.
- Filas.



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

CONTINUAR

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 de la sintaxis del comando *CREATE TABLE*.

- Cada **definición_columna** tiene el siguiente formato:

```
nombre_columna tipo_de_dato restricciones valor_por_defecto
```

CONTINUAR

Tipos de datos

Los principales tipos de datos predefinidos en SQL que pueden asociarse a una columna aparecen en la figura 11.

Estos tipos de datos se agrupan en **clases de almacenamiento**, que son los tipos de datos básicos que se utilizan a la hora de guardar la información en disco. Por ejemplo, los datos de tipo REAL, FLOAT y DOUBLE PRECISION pertenecen a la clase de almacenamiento **REAL** (son todos números reales, de mayor o menor precisión, y por tanto se representan en disco de manera similar).

Tipos de datos	Tipos de datos predefinidos
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 11. Principales tipos en SQL.

Fuente: elaboración propia.

Saber más

Se pueden consultar todas las relaciones entre tipos de datos (**tipos de afinidad**) y las clases de almacenamiento a las que pertenecen en la [documentación oficial de SQLite](#).

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)

Ejemplo

- 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.

CONTINUAR

Definiciones por defecto

La opción **valor_por_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 1).

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

Tabla 1. Funciones para definir valores por defecto.
Fuente: elaboración propia.

CONTINUAR

Restricciones por columna

Cuando se define una columna, además de especificar su nombre y tipo, **se puede establecer un conjunto de restricciones que siempre se tienen que cumplir** (tabla 2):

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

Tabla 2. Restricciones de columna.

Fuente: elaboración propia.

A las restricciones se les puede poner un nombre de la siguiente manera, aunque no es necesario. Por ejemplo:

```
[CONSTRAINT nombre_restricción] CHECK (condiciones)
```

CONTINUAR

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 3):

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 nulos ni repetidos. Es una clave primaria.
PRIMARY KEY	La columna no puede tener valores repetidos ni nulos. Es la clave primaria
FOREIGN KEY (columna [, columna..]) REFERENCES tabla [(columna2 [,columna2...])]	El conjunto de las columnas especificadas es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2. Es una clave alternativa.
CHECK (condiciones)	La tabla debe cumplir las condiciones especificadas

Tabla 3. Restricciones por tabla.
Fuente: elaboración propia.

A las restricciones se les puede poner un nombre de la siguiente manera, aunque no es necesario. Por ejemplo:

```
[CONSTRAINT nombre_restricción] CHECK (condiciones)
```

CONTINUAR

Los siguientes ejemplos de creación de tablas se han elaborado con 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 2, valor por defecto 0.

```
CREATE TABLE sucursal (
    nombre_sucursal VARCHAR(15) PRIMARY KEY,
    ciudad CHAR(20) NOT NULL,
    activos NUMBER(12,2) DEFAULT 0
);
```

Tabla cliente:

- DNI de cliente de hasta 9 caracteres, no nulo, llave primaria (esta vez definida la llave primaria como restricción de tabla).
- Nombre de cliente de hasta 25 caracteres, no nulo.
- Domicilio de hasta 50 caracteres, no nulo.

```
CREATE TABLE cliente (
    dni VARCHAR(9) NOT NULL,
    nombre_cliente CHAR(35) NOT NULL,
    domicilio CHAR(50) NOT NULL,
    PRIMARY KEY (dni)
);
```

Tabla cuenta:

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

```
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)
);
```

Tabla impositor:

- DNI de cliente de hasta 9 caracteres, no nulo, clave foránea a DNI de tabla cliente.
- Número de cuenta de hasta 20 caracteres, no nulo.
- Llaves primarias: DNI y número de cuenta.
- El número de cuenta es clave foránea a la clave primaria de la tabla cuenta.

```
CREATE TABLE impositor (
    dni VARCHAR(9) NOT NULL,
    numero_cuenta CHAR(20) NOT NULL,
    PRIMARY KEY (dni, numero_cuenta),
    FOREIGN KEY (dni) REFERENCES cliente (dni),
    FOREIGN KEY (numero_cuenta) REFERENCES cuenta (numero_cuenta)
);
```

Tabla empleados:

- Código empleado de hasta 9 caracteres, restricción no-nulo.
- Nombre del empleado de hasta 35 caracteres, restricción no-nulo.
- Teléfono de hasta 12 caracteres, restricción no-nulo.
- Dirección de hasta 50 caracteres, restricción no-nulo.
- Ciudad de hasta 50 caracteres, restricción no-nulo.
- Sueldo tipo double de hasta 12 dígitos y 2 decimales, por defecto 100.0
- Clave primaria codigo_empl.

```
CREATE TABLE empleados (
    codigo_empl VARCHAR2(9) PRIMARY KEY,
    nombre_empleado CHAR(35) NOT NULL,
    telefono CHAR(12) NOT NULL,
    direccion CHAR(50) NOT NULL,
    ciudad CHAR(50) NOT NULL,
    sueldo NUMBER(12,2) DEFAULT 100.0,
    nombre_sucursal VARCHAR(15),
    FOREIGN KEY (nombre_sucursal) REFERENCES sucursal (nombre_sucursal)
);
```

CONTINUAR

```

1 CREATE TABLE tarjeta (
2     numero_tarjeta VARCHAR(15) PRIMARY KEY,
3     nombre_tarjeta VARCHAR(20),
4     activos NUMERIC(12,2) DEFAULT 0
5 );
6
7 CREATE TABLE cliente (
8     dni INT PRIMARY KEY,
9     nombre_cliente VARCHAR(20) NOT NULL,
10    numero_cliente CHAR(2) NOT NULL,
11    telefono_cliente CHAR(9) NOT NULL,
12    ciudad_cliente CHAR(20) NOT NULL,
13    PRIMARY KEY (dni)
14 );
15
16 CREATE TABLE cuenta (
17     dni INT PRIMARY KEY,
18     dni_cliente CHAR(2) NOT NULL,
19     numero_cuenta CHAR(20) NOT NULL,
20     saldo NUMBER(12,2) NOT NULL,
21     FOREIGN KEY (dni) REFERENCES cliente (dni),
22     FOREIGN KEY (dni_cliente) REFERENCES cliente (numero_cliente)
23 );
24
25 CREATE TABLE empleado (
26     codig_empl VARCHAR(12) PRIMARY KEY,
27     nombre_empl VARCHAR(20) NOT NULL,
28     apellido_empl VARCHAR(20) NOT NULL,
29     telefono_empl CHAR(9) NOT NULL,
30     dni_empl CHAR(9) NOT NULL,
31     ciudad_empl CHAR(20) NOT NULL,
32     sueldo NUMBER(12,2) NOT NULL,
33     FOREIGN KEY (codig_empl) REFERENCES empleado (codig_empl),
34     FOREIGN KEY (dni_empl) REFERENCES empleado (dni),
35     FOREIGN KEY (ciudad_empl) REFERENCES empleado (ciudad_empl)
36 );
37
38 CREATE TABLE Sucursal (
39     nombre_sucursal VARCHAR(20) PRIMARY KEY,
40     telefono_sucursal CHAR(9) NOT NULL,
41     ciudad_sucursal CHAR(20) NOT NULL,
42     FOREIGN KEY (nombre_sucursal) REFERENCES sucursal (nombre_sucursal)
43 );

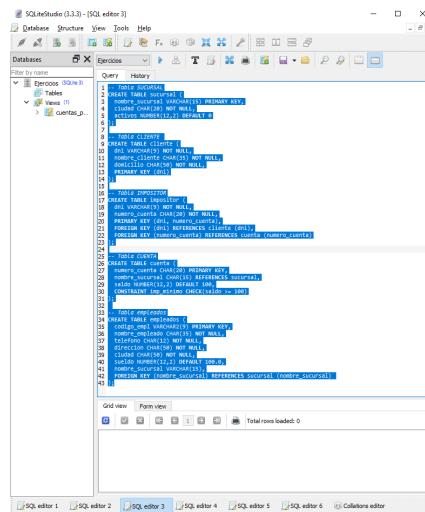
```

Figura 12. Sentencia de SQL para crear tabla.

Fuente: elaboración propia.

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

Se pueden crear las cinco tablas en una sola ejecución si se agrupan todas las sentencias *CREATE TABLE* en el editor (figura 12).



The screenshot shows the SQLiteStudio interface with several SQL statements listed in the query editor. The statements are:

```

1  --CREATE TABLE
2  CREATE TABLE clientes (
3      cliente_id INTEGER PRIMARY KEY,
4      nombre_cliente VARCHAR(50) NOT NULL,
5      telefono_cliente CHAR(10) NOT NULL,
6      email_cliente CHAR(50) NOT NULL,
7      PRIMARY KEY (cliente_id)
8
9  --CREATE TABLE
10 CREATE TABLE cuentas (
11     cuenta_id INTEGER PRIMARY KEY,
12     numero_cuenta VARCHAR(15) NOT NULL,
13     cliente_cliente_id INTEGER NOT NULL,
14     FOREIGN KEY (cliente_cliente_id) REFERENCES clientes (cliente_id),
15     FOREIGN KEY (numero_cuenta) REFERENCES clientes (telefono_cliente)
16
17  --CREATE TABLE
18 CREATE TABLE socios (
19     socio_id INTEGER PRIMARY KEY,
20     nombre_socio VARCHAR(50) NOT NULL,
21     telefono_socio CHAR(10) NOT NULL,
22     email_socio CHAR(50) NOT NULL,
23     cliente_cliente_id INTEGER NOT NULL,
24     FOREIGN KEY (cliente_cliente_id) REFERENCES clientes (telefono_cliente),
25     FOREIGN KEY (numero_cuenta) REFERENCES cuentas (numero_cuenta)
26
27  --CREATE TABLE
28 CREATE TABLE empleados (
29     empleado_id INTEGER PRIMARY KEY,
30     nombre_empleado VARCHAR(50) NOT NULL,
31     telefono_empleado CHAR(10) NOT NULL,
32     email_empleado CHAR(50) NOT NULL,
33     cliente_cliente_id INTEGER NOT NULL,
34     FOREIGN KEY (cliente_cliente_id) REFERENCES clientes (telefono_cliente)
35
36  --HELP
37  --HELP INDEXES
38  --HELP INDEX
39  --HELP INDEXES
40  --HELP INDEX
41  --HELP INDEXES
42  --HELP INDEX
43
44  --HELP INDEXES
45  --HELP INDEX
46  --HELP INDEXES
47  --HELP INDEX
48  --HELP INDEX
49  --HELP INDEXES
50  --HELP INDEX
51
52  --HELP INDEXES
53  --HELP INDEX
54  --HELP INDEXES
55  --HELP INDEX
56  --HELP INDEX
57  --HELP INDEXES
58  --HELP INDEX
59
60  --HELP INDEXES
61  --HELP INDEX
62  --HELP INDEXES
63  --HELP INDEX
64  --HELP INDEX
65  --HELP INDEXES
66
67  --HELP INDEXES
68  --HELP INDEX
69  --HELP INDEXES
70  --HELP INDEX
71  --HELP INDEX
72  --HELP INDEXES
73
74  --HELP INDEXES
75  --HELP INDEX
76  --HELP INDEXES
77  --HELP INDEX
78  --HELP INDEX
79  --HELP INDEXES
80
81  --HELP INDEXES
82  --HELP INDEX
83  --HELP INDEXES
84  --HELP INDEX
85  --HELP INDEX
86  --HELP INDEXES
87
88  --HELP INDEXES
89  --HELP INDEX
90  --HELP INDEXES
91  --HELP INDEX
92  --HELP INDEX
93  --HELP INDEXES
94
95  --HELP INDEXES
96  --HELP INDEX
97  --HELP INDEXES
98  --HELP INDEX
99  --HELP INDEX
100 --HELP INDEXES

```

Figura 13. Selección de sentencias que ejecutar.

Fuente: elaboración propia.

Para ejecutar las cinco sentencias *CREATE TABLE* a la vez, hay que seleccionarlas todas. De lo contrario, el intérprete de SQLiteStudio solo ejecutará la última sentencia definida (figura 13).

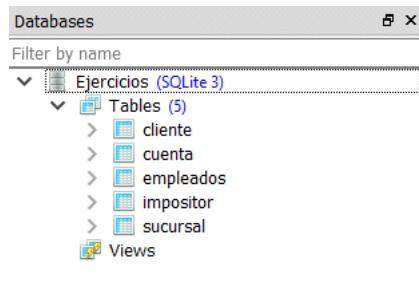


Figura 14. Tablas creadas.

Fuente: elaboración propia.

Para ejecutar las cinco sentencias *CREATE TABLE* seleccionadas, se debe pulsar sobre el icono de ejecutar sentencia SQL. Una vez ejecutadas las sentencias, las cinco nuevas tablas aparecerán creadas en la base de datos (figura 14).

A continuación se muestra el **diagrama entidad-relación**, utilizado para construir el modelo de datos (figura 15).

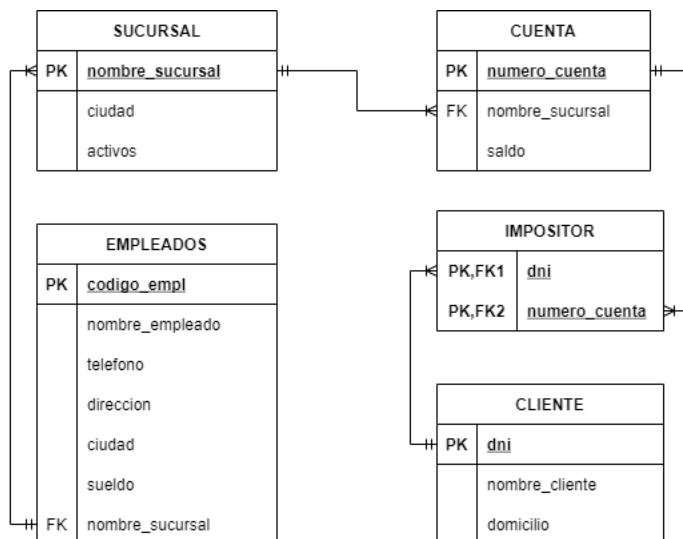


Figura 15. Diagrama entidad-relación.

Fuente: elaboración propia.

CONTINUAR

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. Mediante estas políticas, se definen acciones que se ejecutarán de manera automática en las tablas foráneas cuando ocurrán cambios en la tabla principal. Esto es especialmente importante a la hora de asegurar la **integridad referencial** de los datos, mencionada anteriormente.

```
FOREIGN KEY clave_secundaria REFERENCES tabla [(clave_primaria)]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL | RESTRICT}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL | RESTRICT}]
```

Donde, en caso de borrado (DELETE) o actualización (UPDATE) de la clave padre (clave primaria de la tabla referenciada por la clave foránea que se está definiendo) en alguna de las tuplas de la tabla padre, en la tabla hija:

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 —

Propaga los cambios realizados en la tabla padre hacia la tabla hija. Por ejemplo, si se borra una tupla en la tabla padre, esta política borrará automáticamente aquellas tuplas de la tabla hija que referencien a esa tupla de la tabla padre.

SET NULL —

Las tuplas con la misma clave foránea, que referencian a esa clave primaria, fijarán esa clave foránea a valor NULL.

SET DEFAULT —

Las tuplas cuya clave foránea haga referencia a esa clave primaria, cambiarán el valor de su clave foránea, fijándolo al valor definido por defecto en el momento de la creación de la tabla.

RESTRICT —

Bloquea cualquier operación de actualización o borrado en la tabla padre (concretamente en la clave primaria), siempre y cuando haya alguna tupla en la tabla hija que la refiere.

Ejemplo

Para eliminar las tablas cuenta e impositor, se selecciona cada tabla con el ratón y al hacer clic con el botón derecho aparece un desplegable en el que se elige *Delete the table* (figura 16). Se van a crear nuevamente las tablas con claves foráneas (figura 17).

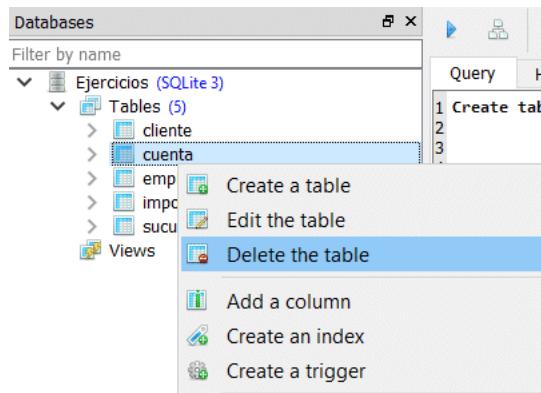


Figura 16. Eliminar tabla.
Fuente: elaboración propia.

```
CREATE TABLE cuenta (
    numero_cuenta CHAR(20) PRIMARY KEY,
    nombre_sucursal CHAR(15) REFERENCES sucursal ON DELETE SET NULL,
    saldo NUMBER(12,2) DEFAULT 100,
    CONSTRAINT imp_minimo CHECK(saldo >= 100)
);
```

```
CREATE TABLE impositor (
    dni VARCHAR(9) NOT NULL,
    numero_cuenta CHAR(20) NOT NULL,
    PRIMARY KEY (dni, numero_cuenta),
    FOREIGN KEY (dni) REFERENCES cliente (dni) ON DELETE CASCADE,
    FOREIGN KEY (numero_cuenta) REFERENCES cuenta (numero_cuenta) ON DELETE CASCADE
);
```

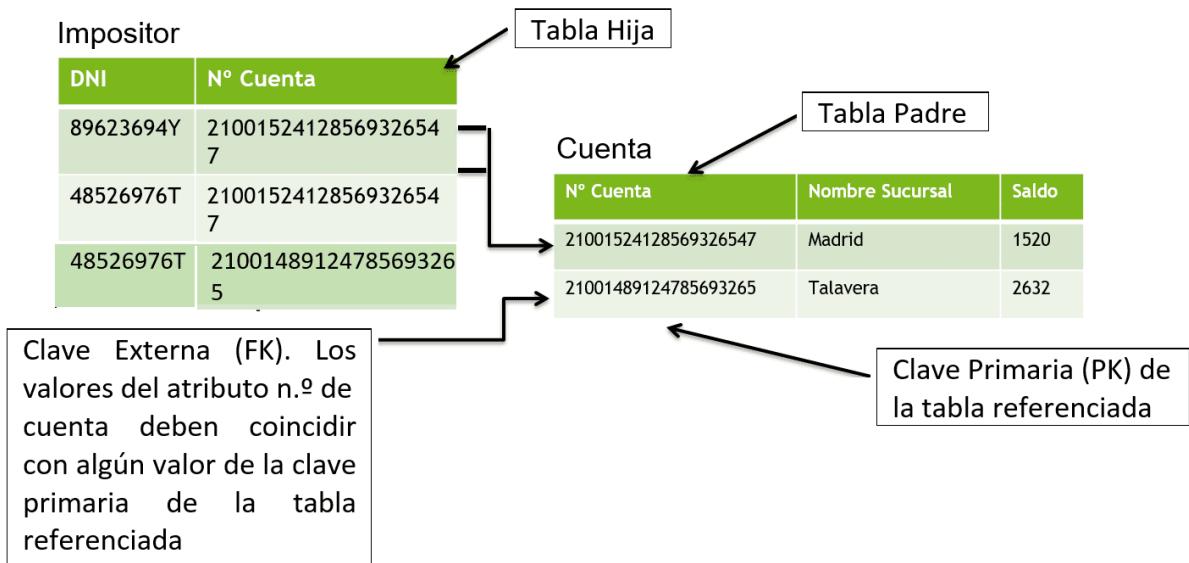


Figura 17. Ejemplo de claves foráneas.

Fuente: elaboración propia.

Si se quieren ver todas las consultas en SQL, se puede descargar el archivo "creacion_tablas.sql" a través del siguiente enlace.



[creacion_tablas.sql.zip](#)
973 B



[CONTINUAR](#)

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}|
```

```
RENAME [COLUMN] columna TO nueva_columna |
```

```
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 a una tabla.

```
alter table R add Atributo Dominio [propiedades]
```



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 de una tabla.

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



Renombrar atributos de una tabla.

```
alter table R rename column Atributo1 to Atributo2
```



Añadir restricciones a una tabla.

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



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.



Desactivar restricciones.

```
alter table R disable CONSTRAINT nombre [CASCADE]
```



Activar restricciones.

```
alter table R enable CONSTRAINT nombre
```

CONTINUAR

Por ejemplo:

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

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

Añadir el atributo *fecha_apertura* (tipo fecha) a la tabla cuenta:

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

Eliminar el atributo *nombre_sucursal* de la tabla cuenta:

```
ALTER TABLE cuenta DROP COLUMN nombre_sucursal;
```

Se añaden de nuevo los atributos originales de la tabla cuenta sin el nombre de sucursal:

```
DROP TABLE cuenta;
```

Modificación tabla CUENTA

```
CREATE TABLE cuenta (
    numero_cuenta CHAR(20) PRIMARY KEY,
    nombre_sucursal CHAR(15) REFERENCES sucursal ON DELETE SET NULL,
    saldo NUMBER(12,2) DEFAULT 100,
    CONSTRAINT imp_minimo CHECK(saldo >= 100)
);
```

Nuevos atributos en cuenta

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

CONTINUAR

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 referencia a la tabla se borrará con esta.

CONTINUAR

5.4. Índices

Los índices permiten que las bases de datos **ejecuten de forma más rápida** 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. No obstante, **se pueden crear índices sobre campos concretos o sobre un conjunto de campos**:

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

En la figura 18 se muestra un ejemplo.

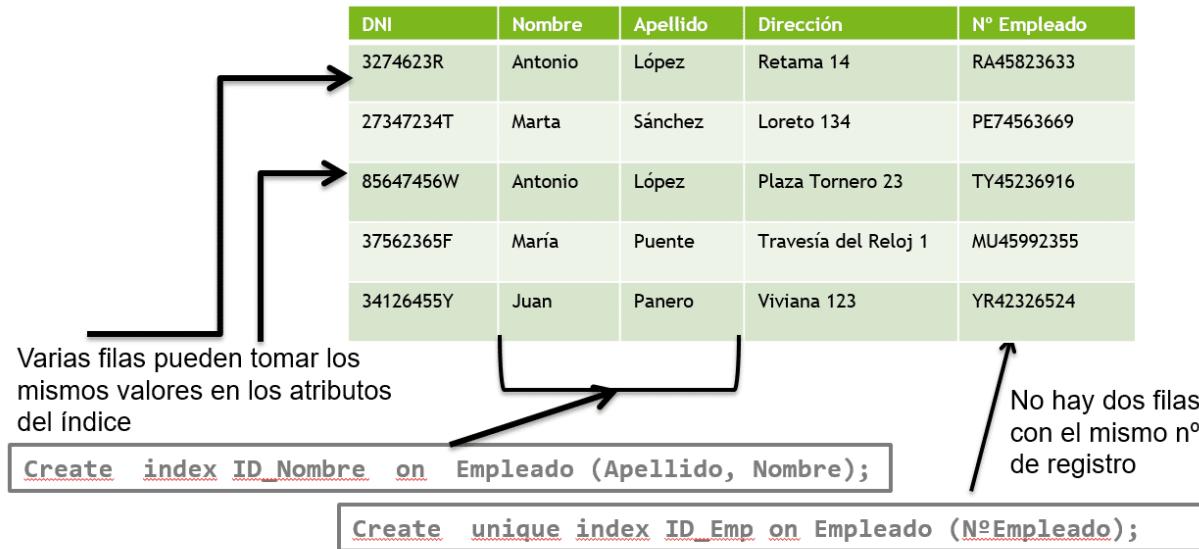


Figura 18. Índices.
Fuente: elaboración propia.

CONTINUAR

5.5. Consultas y recuperación de información en SQL

5.5.1. 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];
```

Atención:

Obsérvese que la palabra clave AS permite renombrar las columnas que se quiere seleccionar o las tablas que se quiere 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 el siguiente enlace.



inserts.sql.zip

1.2 KB



Por ejemplo, **si se quiere seleccionar las columnas "dni" y "nombre_cliente"** de la tabla "cliente", se usaría la sentencia:

```
SELECT dni, nombre_cliente FROM cliente;
```

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

```
SELECT * FROM cliente;
```

Si se quiere **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];
```

CONTINUAR

Téngase en cuenta que:

- La cláusula **WHERE** permite recuperar solo aquellas filas que cumplen la condición especificada.
- La cláusula **DISTINCT** permite ordenar que se muestren 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).



En la definición de estas condiciones se pueden utilizar como parámetros los nombres de las columnas de la tabla, literales, así como cualquiera de las funciones disponibles para la cláusula WHERE.

Por ejemplo, si se quiere recuperar de la tabla únicamente **aquellos saldos que sean diferentes entre sí**:

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

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

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

CONTINUAR

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'
);
```

CONTINUAR

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 limite1 AND limite2;
```

Es una alternativa que permite construir de forma más expresiva y corta una consulta que de otra manera debería componerse así:

```
SELECT nombre_columnas_a_seleccionar FROM tabla_a_consultar WHERE columna >= limite1 AND columna <= limite2;
```

Por ejemplo, se quiere recuperar todos los empleados cuyos sueldos están entre 1000 y 20 000 euros:

```
SELECT codigo_empl FROM empleados WHERE sueldo BETWEEN 1000 and 20000;
```

CONTINUAR

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 quiere recuperar todos los empleados que viven en Madrid y Zaragoza:

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

CONTINUAR

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;
```

Hay 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 quiere 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'
```

CONTINUAR

IS NULL

Comprueba si un valor es 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 quiere recuperar todos los empleados que no tienen un número de teléfono:

```
SELECT * FROM empleados WHERE telefono IS NULL;
```

CONTINUAR

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 quiere recuperar todos los clientes que tienen alguna cuenta:

```
SELECT * FROM cliente WHERE EXISTS (
    SELECT * FROM impositor WHERE impositor.dni == cliente.dni
);
```

CONTINUAR

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_comparacion {ALL|ANY|SOME} subconsulta;
```

CONTINUAR

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 quiere ordenar los empleados por orden alfabético ascendente de acuerdo con su nombre:

```
SELECT * FROM empleados ORDER BY nombre_empleado ASC;
```

Se pueden establecer límites estáticos a la cantidad de filas retornadas por una consulta SELECT. Para ello, se añade al final de la sentencia la cláusula **LIMIT <numero_filas>**.

Por ejemplo, para retornar las cinco primeras filas de la tabla **cliente**:

```
SELECT * FROM cliente LIMIT 5;
```

CONTINUAR

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**. Para ello se introduce la cláusula **JOIN** (con sus variantes), que permite definir una tabla de unión, y las condiciones mediante las cuales se unirán las tuplas de ambas tablas. Existen varios casos:

a) 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**, se pueden combinar las tablas **cliente** e **impositor** de la siguiente manera:

```
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;
-- unimos las tuplas de cliente e impositor cuyos dni sean iguales
-- asimismo, unimos las tuplas de impositor y cuenta cuyo número de cuenta sea el mismo
-- el resultado es que, para cada cliente, tenemos una tupla con todos sus datos de cliente, cuenta e impositor, s
```

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;
```

CONTINUAR

b) 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;
```

CONTINUAR

c) Combinación interna vs. externa.

INTERNA(INNER JOIN)

EXTERNA(OUTER JOIN)

Solo se consideran **las filas que tienen valores idénticos en las columnas de las tablas que compara**. Si alguna de las tuplas en ambas tablas no cumple la condición para unirse a una tupla de la otra tabla, no se incluye en el resultado.

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

INTERNA(INNER JOIN)

EXTERNA(OUTER JOIN)

Se consideran los valores de la tabla derecha, de la izquierda o de ambas tablas. Si alguna de las tuplas en ambas tablas no cumple las condiciones para unirse a una tupla de la otra tabla, se incluye en el resultado. Aquellos valores de la tabla para la que no hay referencia se incorporan con valor **NULL**.

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

Se puede consultar la guía detallada de tipos de JOIN disponibles en SQLite en la siguiente dirección web: <https://www.sqlitetutorial.net/sqlite-join/>

CONTINUAR

d) Combinación de tablas.

Es posible combinar las tuplas de varias tablas, de manera que para cada tupla de la primera tabla se tendrán todas las tuplas de la segunda. Para cada tupla de la segunda se tendrán todas las de la tercera y así sucesivamente.

Si por ejemplo se combinan todas las tuplas de las tablas cliente, cuenta e impositor, y cada tabla tiene un número total de tuplas **X**, **Y** y **Z** respectivamente, el número total de tuplas generadas en la consulta será de **X * Y * Z**:

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

CONTINUAR

No obstante, existen mecanismos para combinar los datos de varias consultas:

1



Unión. Permite unir los resultados de dos 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 quiere 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;
```

2



Intersección. Permite hacer la intersección entre los resultados de dos 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 quiere 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

USANDO IN

USANDO EXISTS

```
SELECT ciudad FROM empleados INTERSECT SELECT ciudad FROM sucursal;
```

USANDO LA INTERSECCIÓN

USANDO IN

USANDO EXISTS

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

USANDO LA INTERSECCIÓN

USANDO IN

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 dos 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 quiere 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 cuáles son las ciudades de los empleados en las que no hay sucursales:

USANDO LA INTERSECCIÓN

USANDO IN

USANDO EXISTS

```
SELECT ciudad FROM empleados INTERSECT SELECT ciudad FROM sucursal;
```

USANDO LA INTERSECCIÓN

USANDO IN

USANDO EXISTS

```
SELECT c.ciudad FROM empleados c WHERE c.ciudad IN ( SELECT d.ciudad FROM sucursal d
```

```
 );
```

USANDO LA INTERSECCIÓN

USANDO IN

USANDO EXISTS

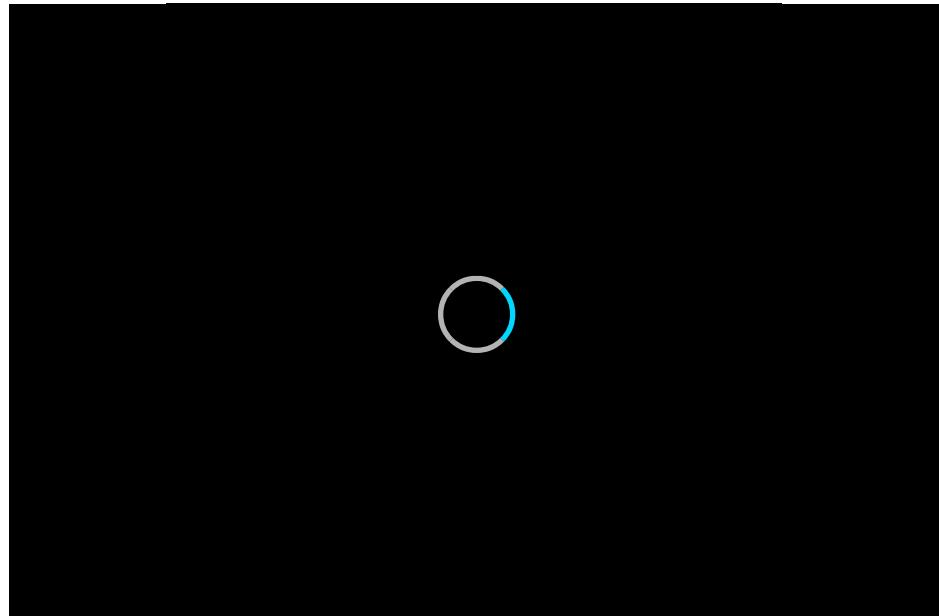
```
SELECT c.ciudad FROM empleados c WHERE EXISTS (SELECT * FROM sucursal d WHERE c.ciudad = d.ciudad  
);
```

Si se quieren ver todas las consultas en SQL, se puede descargar el archivo "consultas_ejemplo.sql" a través del siguiente enlace.



[consultas_ejemplo.sql.zip](#)

1.2 KB



5.6. Modificación de tablas en SQL

5.6.1. Operaciones de actualización

Inserción

Para poder consultar los datos de una base de datos, estos deben existir previamente. Para insertar datos en una tabla se puede utilizar la sentencia **INSERT**:

```
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:

- Con la instrucción **VALUES** se especifican los valores literales que introducir en la tabla. Los valores v₁, v₂... v_n se deben corresponder con las columnas de la tabla especificada y deben estar en el mismo orden, a menos que se vuelvan 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.
- Como alternativa a **VALUES**, se puede definir una lista de tuplas de entrada a partir de los resultados de una consulta a otra u 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:

```
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 Agustín No.81');  
INSERT INTO cliente (dni,nombre_cliente,domicilio) VALUES( '12346', 'Alondra López', 'Santa Rita No.1501');
```

De la misma manera, si se conoce el orden de las columnas:

```
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 Agustín 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. Suponiendo que existe la tabla nuevas_cuentas:

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

CONTINUAR

Borrado

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

```
DELETE FROM nombre_tabla [WHERE condiciones];
```

Atención:

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 quiere borrar todas las filas de la tabla empleados se usaría la sentencia:

```
DELETE FROM empleados;
```

Sin embargo, si solo se quiere 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';
```

CONTINUAR

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;
```

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

La cláusula SET indica qué columna modificar y los valores que puede recibir y la cláusula WHERE especifica qué filas deben actualizarse. Al igual que en el caso de la operación **DELETE**, si no se especifica una cláusula **WHERE**, todas las tuplas de la tabla serán modificadas.

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

Actualización del saldo del empleado cuyo código es '34566':

```
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'
);
```

CONTINUAR

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 19.

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 contaría los valores no nulos. Por ejemplo, si se quiere contar el número de clientes de la tabla clientes cuya ciudad es Madrid:

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 19. Índices.

Fuente: elaboración propia.

```
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.

CONTINUAR

5.7. Vistas

Una vista es el resultado de la ejecución de una consulta al que se da un nombre, de manera que puede utilizarse en otras consultas como si se tratase de una tabla real.

Las vistas pueden ser útiles para abstraer consultas de cierta complejidad, o simplemente para reutilizar consultas muy utilizadas y que, de otra manera, habría que construir en cada uso.

Para construir una vista se puede utilizar la siguiente sintaxis:

```
CREATE [TEMP] VIEW [IF NOT EXISTS] nombre_vista [(lista_columnas)] AS (consulta)
```

TEMP

IF NOT EXISTS

Especifica que **la vista solo existirá mientras exista la actual conexión a base de datos**. Una vez termine la sesión, la vista se borrará automáticamente.

TEMP

IF NOT EXISTS

Solo crea la vista si no existe ya. Si no se indica esta cláusula, una segunda ejecución de la sentencia de creación de la vista fallaría.



Una vez creada, se puede utilizar nombre_vista en cualquier otra consulta como si de una tabla se tratase.

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 refiere a la vista se borrará con esta.

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

	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 Agustín No.81
4	12346	Alondra López	Santa Rita No.1501

Tabla 4. Tabla de clientes.
Fuente: elaboración propia.

	codigo_empl	nombre_empleado	teléfono	direccion	ciudad	sueldo
1	e12345	Pepito Díaz	1222299988	Calle 13 Norte No. 5	Madrid	12343.9
2	e12344	Juan Méndez	1342794996	Calle 41 Norte No. 612	Madrid	12343.9
3	e12343	Maria Ortega	1322693985	Calle 67 Poniente No. 91	Guadalajara	12343.9
4	e12342	Jesús Robles	1422592976	Calle 90 Oriente No. 543	Guadalajara	12343.9
5	e12341	Ricardo Vargas	1522491967	Calle 11 Sur No. 6780	Madrid	12343.9

Tabla 5. Tabla de empleados.
Fuente: elaboración propia.

CONTINUAR

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;
```

Si se ejecuta la siguiente sentencia, se obtendrían los valores de la figura 3.24:

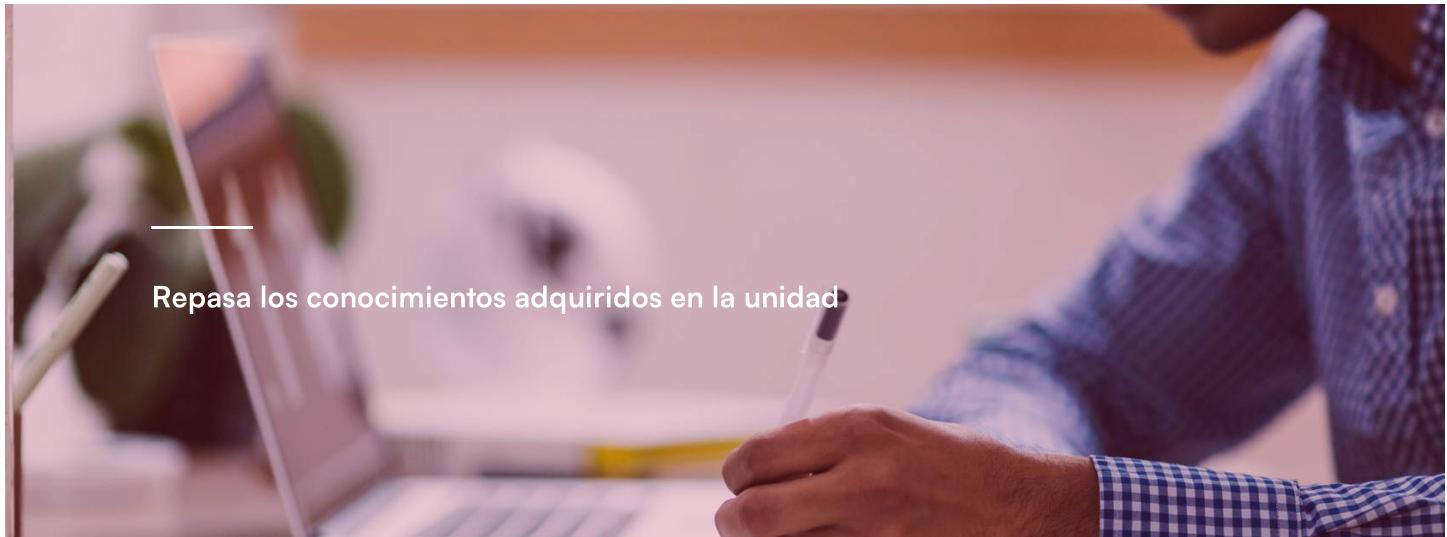
```
SELECT * FROM cuentas_por_cliente;
```

	dni	nombre_cliente	numero_cuenta	saldo
1	12345	Luis Díaz	6789	120.01
2	12345	Luis Díaz	6786	1222.3
3	12344	Javier Lozano	6788	23445.01
4	12343	Andres López	6787	4566.78

Tabla 6. Vista resultante.

Fuente: elaboración propia.

VI. Resumen



Repasa los conocimientos adquiridos en la unidad

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.

Dentro de este modelo, se ha hecho especial hincapié en el concepto de **integridad** de una relación. Para mantener la consistencia de los datos en la base de datos, hay que asegurarse de que se cumplen una serie de reglas. Para ello se ha introducido el concepto de claves, que permiten identificar de forma única entidades concretas.

Estas claves permiten establecer relaciones entre entidades y gracias a la unicidad de las claves se puede asegurar la unicidad también de las relaciones.

Dentro de las formas de integridad, se han definido las dos más importantes:

- Unicidad y entidad de la clave primaria. Es decir, una clave primaria siempre debe tener un valor y ser único.
- Integridad referencial, por la cual se puede garantizar que, en una relación entre dos entidades, ambas existen y son únicas en su entorno, constituyendo pues una relación única.

Se ha visto también que es posible activar mecanismos automáticos para garantizar esta integridad referencial, como los "constraints". Por ejemplo, para asegurar que una entidad se borra cuando, al borrar su entidad relacionada, esta deja de tener sentido (por ejemplo, si se borra un usuario de un banco, sus cuentas deberían ser borradas automáticamente).

Se ha mostrado cómo utilizar el lenguaje SQL para ejecutar cualquier operación CRUD en la base de datos:

- Create: con la sentencia *INSERT*.
- Read: con la sentencia *SELECT*.
- Update: con la sentencia *UPDATE*.
- Delete: con la sentencia *DELETE*.

En el ámbito de la lectura de datos, se ha visto cómo hacer lecturas combinando datos de tablas distintas, utilizando las relaciones entre entidades como los nexos de unión, en sentencias *SELECT* acompañadas de *JOIN*.

Y finalmente se han abstraído consultas de cierta complejidad en vistas, que pueden consultarse posteriormente como si se tratase de tablas.

Con lo aprendido en este módulo, se puede afrontar con confianza el acceso a cualquier sistema de consultas basado en SQL, independientemente del motor de base de datos utilizado. Esto se aplica a los *data warehouses* modernos, que, si bien no siguen un modelo relacional en su implementación, exponen una interfaz donde ejecutar consultas SQL para extraer datos, de forma similar a como se haría en una base de datos relacional.

VII. Caso práctico con solución



Aplica los conocimientos adquiridos en esta unidad

ENUNCIADO

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.

Se van a crear tres 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

- Rellenar 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



Hacer 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 dos 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".

[VER SOLUCIÓN](#)

SOLUCIÓN

- 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');
```

- 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, Puntos, Solicitudes.Nombre_Inst, Plazas FROM Estudiantes, Institutos, Solicitud
```

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) -
```

5.

```
SELECT * FROM Solicitudes WHERE ID IN (SELECT ID FROM Solicitudes GROUP BY ID HAVING COUNT (Via) >2);
```

6.

```
SELECT Via FROM Estudiantes, Solicitudes WHERE Estudiantes.ID= Solicitudes.ID GROUP BY Via HAVING MAX(Puntos) < (S
```

7.

```
SELECT DISTINCT Nombre_Est, Via FROM Estudiantes JOIN Solicitudes ON Estudiantes.ID = Solicitudes.ID;
```

8.

```
SELECT Nombre_Est FROM Estudiantes JOIN Solicitudes ON Estudiantes.ID = Solicitudes.ID AND Valor < 1000 AND Via='T'
```

VIII. Lecturas recomendadas

- Prescott, P. *SQL Para Principiantes*. Ed. Babelcube; 2016. Disponible en la biblioteca virtual.

IX. Enlaces de interés

SQLite Tutorial.

ABRIR ENLACE

X. Glosario



El glosario contiene términos destacados para la comprensión de la unidad

Modelo relacional —

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.

Instancia —

Representa una materialización o particularización de una relación dada.

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.

Base de datos relacional —

Tipo de base de datos que sigue el modelo relacional.

Tabla —

Representación conceptual en la que se almacenan los datos.

Dominio —

Conjunto de valores de un tipo de datos que son atómicos.

Columna o atributo —

Aquellos elementos de información de una relación cuyos valores se quiere almacenar.

Tupla o fila —

Valores concretos que toma una instancia de una relación en los atributos que se han elegido para representar la relación.

Superclave —

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.

Clave candidata —

Superclave de la relación que hace cumplir que ningún subconjunto propio sea superclave.

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 las claves candidatas no elegidas se denominan claves alternativas.

Clave alternativa —

El resto de las claves candidatas no elegidas como claves primarias.

Clave foránea —

Permite establecer conexiones entre las tuplas de varias relaciones.

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.

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.

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.

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.

SQL —

Lenguaje que permite manipular una base de datos relacional.

Consulta —

Sentencia en SQL que permite recuperar o modificar una tabla de una base de datos relacional.

SQLiteStudio



Sistema de gestión de bases de datos relacionales basado en el lenguaje SQLite. Este lenguaje está constituido por un subconjunto del lenguaje SQL.

Diagrama entidad-relación



Esquema que permite representar las relaciones y las restricciones que aplican en sus interacciones.

XI. Bibliografía

- Elmasri, R. y Navathe, S. B. *Fundamentals of Database Systems*. Pearson; 2017, 7.^a ed.
- García Molina, H., Ulman, J. D. y Widom, J. *Database Systems: The Complete Book*. Prentice Hall; 2009, 2.^a ed.
- Silberschatz, A. *Fundamentos de bases de datos*. McGraw Hill; 2014, 6.^a ed.
- [SQLite Language](#).
- [SQLite Tutorial](#).
- Walter Shields. *SQL QuickStart Guide: The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data With SQL*. ClydeBank Media, 2019.
- Anthony Molinaro. *SQL Cookbook*. O'Reilly, 2005.