

USERS



SQL



Conceptos preliminares.
Sentencias y consultas.
Manejo de bases de datos.

SQL



Conceptos preliminares.
Sentencias y consultas.
Manejo de bases de datos.

USERS

Título: SQ: en 48 horas - Conceptos preliminares - Sentencias y consultas - Manejo de bases de datos

Autor: Alejandro Belmar / **Coordinador editorial / Edición:** Claudio Peña

Producción gráfica: Gustavo De Matteo / **Colección:** USERS ebooks - LPCU 345

Copyright © MMXXII. Es una publicación de Six Ediciones. Hecho el depósito que marca la ley 11723. Todos los derechos reservados. Esta publicación no puede ser reproducida ni en todo ni en parte, por ningún medio actual o futuro, sin el permiso previo y por escrito de Six Ediciones. Su infracción está penada por las leyes 11723 y 25446. La editorial no asume responsabilidad alguna por cualquier consecuencia derivada de la fabricación, funcionamiento y/o utilización de los servicios y productos que se describen y/o analizan. Todas las marcas mencionadas en este libro son propiedad exclusiva de sus respectivos dueños. Libro de edición argentina.

Belmar, Alejandro

SQL: en 48 horas: Conceptos preliminares. Sentencias y consultas. Manejo de bases de datos / Alejandro Belmar. - 1a ed. - Ciudad Autónoma de Buenos Aires: Six Ediciones, 2022.

Libro digital, PDF/A

Archivo Digital: online

ISBN 978-987-8414-35-5

1. Computación. 2. Desarrollo de Programas. I. Título.

CDD 005.133

ISBN 978-987-8414-35-5



9 789878 414355

Prólogo

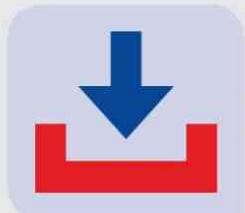
Si piensas en el valor que SQL te ofrece, sin duda lo más relevante es que te permitirá interactuar con una base de datos. El lenguaje de consulta estructurado SQL es el lenguaje de base de datos más implementado y, por lo tanto, más valioso, para cualquier persona que se encuentre involucrada en el desarrollo de aplicaciones o que utilice bases de datos para recopilar y organizar información.

Se trata de un lenguaje que funciona basado en un sistema de consultas estructuradas. Se utiliza para administrar bases de datos relacionales y para realizar operaciones con los datos que se encuentran contenidos en ellas. SQL te servirá para realizar preguntas a las bases de datos y, de esta forma, obtener respuestas que relacionen y presenten la información que incluyen. Es una herramienta que te permitirá llevar a cabo análisis y tomar decisiones basándote en los datos almacenados.

En este sentido, si te dedicas o si quieras dedicarte a la informática y al desarrollo, conocer SQL es un elemento básico que no puedes dejar pasar y, en este e-book, encontrarás lo esencial para que puedas realizar tus primeras consultas SQL en menos de 48 horas.



Este e-book incluye, al final de cada capítulo, una serie de actividades para que puedas autoevaluarte. Asimismo, cada vez que veas la señal de la derecha, significará que encontrarás material adicional disponible en el archivo de contenidos complementarios que puedes descargar **AQUÍ**!



Sobre este e-book

En este e-book encontrarás la información para aprender lo básico de SQL en menos de 48 horas. El contenido de la obra se encuentra dividido en cinco capítulos en los que se ofrece tanto la información teórica necesaria como los procedimientos prácticos que requerirás para realizar tus primeras consultas.

En los dos primeros capítulos, aprenderás qué es SQL y para qué sirve, así como también verás la forma en que deberás implementar tu entorno de trabajo para probar tus consultas SQL.

En el tercer capítulo, aprenderás a crear bases de datos y tablas, y a realizar algunas operaciones básicas sobre ellas. En el cuarto capítulo, revisarás la forma en que puedes efectuar diferentes consultas para agregar, modificar y rescatar la información desde una base de datos.

Finalmente, en el capítulo cinco verás la sintaxis de algunas opciones más complejas mediante las cuales será posible realizar subconsultas y, también, combinar información de diferentes tablas.



Contenido

CAPÍTULO 1 Conceptos preliminares

- ¿Qué es SQL? / 9
- Estructura mínima / 11
- Para qué sirve / 15
- Preparar el entorno / 20
 - Alternativas online / 23
 - SQLite Online
 - SQL Fiddle
 - JDoodle
- Actividades / 27
- Test de autoevaluación / 27



CAPÍTULO 2 Elementos del lenguaje y sentencias

- Campos y datos / 29
- Sentencias SQL / 32
- Sintaxis de una consulta / 33
 - SELECT / 35
 - ALL / DISTINCT / 37
 - Nombres de campos / 39
 - AS / 40
 - FROM / 40
 - WHERE / 40
 - Condiciones / 42
 - ORDER BY / 47
 - ASC / DESC / 48
 - Operadores lógicos / 49
- Actividades / 51
- Test de autoevaluación / 51



CAPÍTULO 3 Bases de datos y tablas

- Bases de datos / 53
- CREATE DATABASE / 53
- Crear una tabla / 54



Estructura de una tabla / 58

CONSTRAINT / 63

Modificar una tabla / 65

Eliminar una tabla / 66

Actividades / 71

Test de autoevaluación / 71



CAPÍTULO 4 Insertar y modificar datos

Insertar datos / 73

Modificar datos / 74

Eliminar datos / 75

Commit y rollback / 76

Visualizar datos / 77

WHERE / 84

COUNT / 86

Sum, Avg, Min, Max / 87

DISTINCT / 87

ORDER BY / 88

UNION / 89

Actividades / 91

Test de autoevaluación / 91



CAPÍTULO 5 Subconsultas y uniones

Subconsultas / 93

Agrupaciones y combinaciones / 99

Autocombinación / 102

Actividades / 104

Test de autoevaluación / 104

Glosario / 105

Capítulo 01



Conceptos preliminares

Para que puedas enfrentarte a la tarea de aprender lo más básico de SQL en tan solo 48 horas, es necesario comenzar entendiendo qué es SQL, para qué te servirá y, también, configurar tu primer entorno de pruebas.

Encontrarás todo esto en el primer capítulo de este e-book.

Al terminar este capítulo podrás realizar las siguientes tareas:

- ✿ Saber qué es y para qué sirve SQL.
- ✿ Indicar en qué escenarios será útil el manejo de sentencias SQL.
- ✿ Preparar el entorno de pruebas para ejecutar tus sentencias SQL.

¿Qué es SQL? / 9

Estructura mínima / 11

Para qué sirve / 15

Preparar el entorno / 20

Alternativas online / 23

SQLite Online

SQL Fiddle

JDoodle

Actividades / 27

Test de autoevaluación / 27

¿Qué es SQL?

La **base de datos** es una colección sistemática de datos. Estas admiten el almacenamiento y la manipulación de datos y facilitan su administración, accesibles, por ejemplo, mediante ODBC, JDBC o ADO.

En otras palabras, una base de datos es uno o varios archivos en donde la información está registrada de forma estructurada en **tablas** que contienen **registros**. Estos, a su vez, están compuestos de **campos** bien identificados.

La base de datos más simple es un archivo de texto correspondiente a una tabla, en donde los campos son delimitados por un carácter (como una coma) o por posición (tamaño fijo).

El **sistema de gestión de bases de datos (DBMS)** es una colección de programas que permite a sus usuarios acceder a la base de datos, manipularlos y realizar informes/representación de ellos.

Existen cuatro tipos principales de DBMS.

A continuación, se presentan en detalle.

- **Jerárquico:** este tipo de DBMS emplea la relación “padre-hijo” de almacenamiento de datos, pero rara vez se usa hoy en día. Su estructura es como un árbol con nodos que representan registros y ramas que constituyen campos. El registro de Windows utilizado en Windows XP es un ejemplo de una base de datos jerárquica. Los ajustes de configuración se almacenan como estructuras de árbol con nodos.
- **DBMS de red:** este tipo de DBMS admite muchas relaciones. Por lo general, esto da como resultado estructuras de bases de datos complejas. El servidor **RDM** es un ejemplo de un sistema de gestión de bases de datos que implementa el modelo de red.
- **DBMS relacional:** este tipo de DBMS define las relaciones de la base de datos en forma de tablas, también conocidas como **relaciones**. A diferencia del DBMS de red, el **RDBMS** no admite muchas relaciones. El DBMS relacional generalmente tiene predefinidos los tipos de datos

1. Conceptos preliminares

que puede admitir. Este es el DBMS más popular en el mercado. Algunos ejemplos de sistemas de administración de bases de datos relacionales incluyen las bases de datos **MySQL, Oracle y Microsoft SQL Server**.

- **DBMS de relación orientada a objetos:** este admite el almacenamiento de nuevos tipos de datos, que estarán en forma de **objetos**. Los objetos que se almacenarán en la base de datos tienen **atributos** (es decir, género, edad) y **métodos** que definen qué hacer con los datos. PostgreSQL es un ejemplo de un SGBD relacional orientado a objetos.

El lenguaje de consulta estructurado (**SQL**) es en realidad el lenguaje estándar para tratar con las bases de datos relacionales.

SQL significa **lenguaje estructurado de consulta** (*Structured Query Language*). Se trata de un lenguaje estándar de cuarta generación que se utiliza para definir, gestionar y manipular la información contenida en una **Base de Datos Relacional**.

Es un lenguaje definido por el estándar ISO/ANSI SQL que utilizan los principales fabricantes de sistemas de gestión de bases de datos relacionales.

En los lenguajes procedimentales de tercera generación, se deben especificar todos los pasos que hay que dar para conseguir el resultado. Sin embargo, en SQL tan solo es necesario indicar al SGBD qué es lo que quieres obtener, y el sistema decidirá cómo lograrlo.

Es un lenguaje sencillo y potente que se emplea para la gestión de la base de datos a distintos niveles de uso: usuarios, programadores y administradores de la base de datos.

Es importante que consideres que el lenguaje SQL está compuesto por **comandos, cláusulas, operadores y funciones de agregado**.

Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

SQL proporciona métodos para definir la base datos, para manipular la información y para gestionar los permisos de acceso a dicha información.

Estructura mínima

La estructura mínima de almacenamiento que encontrarás en una base de datos contiene los siguientes elementos:

- **Tabla.** Objeto de almacenamiento perteneciente a una BD. Es una estructura en forma de cuadrante donde se almacenan registros o filas de datos. Cada tabla tiene un nombre único en la BD.
- **Registro.** Cada una de las filas de una tabla; está compuesto por campos o atributos.
- **Campo.** Cada uno de los “cajoncitos” de un registro donde se guardan los datos. Cada campo tiene un nombre único para la tabla de la cual forma parte; además, es de un tipo (naturaleza) determinado, por lo tanto, no se pueden guardar limones en el cajón de las naranjas, en términos informáticos y a modo de ejemplo, no se encontrará un dato alfanumérico (letras y números) en un campo diseñado para guardar datos numéricos.

A continuación, se muestra un ejemplo.

Tabla Jefes de Departamento

ID	FName	LName	Teléfono	ManagerID	DepartmentID	Salario	Fecha de contratación
1	James	Herrero	1234567890	NULO	1	1000	01-01-2002
2	Juan	Johnson	2468101214	1	1	400	23-03-2005
3	Miguel	Williams	1357911131	1	2	600	12-05-2009
4	Johnathon	Herrero	1212121212	2	1	500	24-07-2016

Para llegar a esta tabla de datos, se precisan las siguientes sentencias SQL, aunque por ahora no es necesario que te detengas en ellas, solo basta con

1. Conceptos preliminares

saber que existe una serie de sentencias que te permitirán crear bases de datos, tablas y datos.

```
CREATE TABLE Employees (
    Id INT NOT NULL AUTO_INCREMENT,
    FName VARCHAR(35) NOT NULL,
    LName VARCHAR(35) NOT NULL,
    PhoneNumber VARCHAR(11),
    ManagerId INT,
    DepartmentId INT NOT NULL,
    Salary INT NOT NULL,
    HireDate DATETIME NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (ManagerId) REFERENCES Employees(Id),
    FOREIGN KEY (DepartmentId) REFERENCES Departments(Id)
);
```

```
INSERT INTO Employees
    ([Id], [FName], [LName], [PhoneNumber], [ManagerId],
    [DepartmentId], [Salary], [HireDate])
VALUES
    (1, 'James', 'Smith', 1234567890, NULL, 1, 1000, '01-01-2002'),
    (2, 'John', 'Johnson', 2468101214, '1', 1, 400, '23-03-2005'),
    (3, 'Michael', 'Williams', 1357911131, '1', 2, 600, '12-05-2009'),
    (4, 'Johnathon', 'Smith', 1212121212, '2', 1, 500, '24-07-2016')
;
```

A continuación, puedes ver otros ejemplos comunes, con la tabla final y las sentencias SQL asociadas:

Tabla Automóviles

ID	Identificación del cliente	ID de empleado	Modelo	Estado	Coste total
1	1	2	Ford F-150	LISTO	230
2	1	2	Ford F-150	LISTO	200
3	2	1	Ford Mustang	ESPERANDO	100
4	3	3	Toyota Prius	TRABAJANDO	1254

Sentencias SQL para crear la tabla:

```
CREATE TABLE Cars (
    Id INT NOT NULL AUTO_INCREMENT,
    CustomerId INT NOT NULL,
    EmployeeId INT NOT NULL,
    Model varchar(50) NOT NULL,
    Status varchar(25) NOT NULL,
    TotalCost INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (CustomerId) REFERENCES Customers(Id),
    FOREIGN KEY (EmployeeId) REFERENCES Employees(Id)
);

INSERT INTO Cars
    ([Id], [CustomerId], [EmployeeId], [Model], [Status],
    [TotalCost])
VALUES
    ('1', '1', '2', 'Ford F-150', 'READY', '230'),
```

1. Conceptos preliminares

```
(‘2’, ‘1’, ‘2’, ‘Ford F-150’, ‘READY’, ‘200’),  
(‘3’, ‘2’, ‘1’, ‘Ford Mustang’, ‘WAITING’, ‘100’),  
(‘4’, ‘3’, ‘3’, ‘Toyota Prius’, ‘WORKING’, ‘1254’)
```

Tabla Socios

ID	FName	LName	E-mail	Número de teléfono	Contacto preferido
1	William	Jones	william.jones@example.com	3347927472	Teléfono
2	David	Molinero	dmiller@example.net	2137921892	Correo electrónico
3	Ricardo	Davis	richard0123@example.com	NULO	Correo electrónico

Sentencias SQL asociadas:

```
CREATE TABLE Customers (  
    Id INT NOT NULL AUTO_INCREMENT,  
    FName VARCHAR(35) NOT NULL,  
    LName VARCHAR(35) NOT NULL,  
    Email varchar(100) NOT NULL,  
    PhoneNumber VARCHAR(11),  
    PreferredContact VARCHAR(5) NOT NULL,  
    PRIMARY KEY(Id)  
);
```

```
INSERT INTO Customers  
    ([Id], [FName], [LName], [Email], [PhoneNumber],  
    [PreferredContact])  
VALUES
```

```
(1, 'William', 'Jones', 'william.jones@example.com',
'3347927472', 'PHONE'),
(2, 'David', 'Miller', 'dmiller@example.net',
'2137921892', 'EMAIL'),
(3, 'Richard', 'Davis', 'richard0123@example.com',
NULL, 'EMAIL')
;
```

Para qué sirve

SQL se convirtió en la norma del **American National Standards Institute (ANSI)** en el año 1986 y, al año siguiente, en 1987, se convirtió en la norma de la **Organización Internacional de Normalización (ISO)**. Hoy vives en el mundo de internet y de la tecnología. Te encuentras rodeado de toneladas de datos. Entonces, para almacenar estos datos de forma segura y administrarlos, necesitas una base de datos adecuada y, para administrar una base de datos, precisas un lenguaje como SQL. Este cuenta con una amplia gama de aplicaciones y es utilizado por profesionales de negocios, desarrolladores, científicos de datos, para mantener y manipular los datos. Según las estadísticas recientes de 2020-21, SQL sigue siendo uno de los lenguajes más utilizados en todo el mundo.

Sus principales usos son los siguientes:

- **Base de datos relacional:** para comprender la base de datos relacional, primero debes comprender el modelo relacional abreviado como **RM**. RM de base de datos es simplemente almacenar los datos y administrarlos de una manera estructurada y particular. En RM, se almacenan datos en forma de filas, conocidas principalmente como **tuplas**, que se agrupan en relaciones. RM ayuda a almacenar datos de manera descriptiva y

1. Conceptos preliminares

concisa, lo que colabora para recuperar y manipular datos con consultas simples. En el año 1970, apareció un software llamado “sistema de administración de bases de datos relacionales (RDBMS)”, que se utiliza para mantener RM usando SQL. SQL ayuda a consultar y mantener datos en el RDBMS. Este es uno de los principales usos del lenguaje SQL, le da la orden a RDBMS para realizar ciertas tareas en forma de consultas.

- **Realización de las operaciones básicas:** SQL ayuda a realizar toneladas de comandos que permiten ejecutar varias operaciones en una base de datos. En términos generales, se pueden clasificar en cuatro categorías:
 - **Lenguaje de definición de datos:** SQL ayuda a crear una base de datos y tablas, y realizar ciertas operaciones, como crear, modificar, renombrar, truncar, etcétera.
 - **Lenguaje de consulta de datos:** se usa ampliamente el comando **SELECT** para recuperar información de la base de datos.
 - **Lenguaje de manipulación de datos:** SQL se utiliza para manipular datos en una base de datos.
 - **Lenguaje de control de datos:** los comandos de control se utilizan para otorgar permiso/acceso al usuario para realizar una operación específica.
- **Control de transacciones:** SQL también se usa para mantener las transacciones que ocurren en las bases de datos. Incluye seguir reglas básicas para mantener la consistencia en dicha base. Estas reglas se denominan **propiedades ACID**. Los comandos de control más utilizados son **COMMIT**, **ROLLBACK**, etcétera.
- **SQL UNION:** en álgebra relacional, se usa el operador de unión para combinar dos conjuntos en un solo conjunto con distintos valores en él. De manera similar, SQL UNION ayuda a proporcionar un solo resultado utilizando dos instrucciones **SELECT** distintas y separadas. La unión de dos tablas da atributos distintos (columnas) presentes en ambas tablas.

- **SQL JOIN:** esta instrucción ayuda a unir dos o más tablas para recuperar información. **JOIN** se usa principalmente porque, en una base de datos, es posible tener dos o más tablas. Por ejemplo, hay dos tablas: **Cliente**, que consta de información del cliente, y **Producto**, que consta de información sobre el producto. Entonces, para saber los detalles sobre los datos del cliente que compró el producto, necesitas la operación **JOIN**.
- **SQL en la Web:** hay una aplicación importante de lenguaje de consulta como SQL en sitios web interactivos que contienen mucha información sobre usuarios, productos en forma de bases de datos. El back-end de cada sitio web está respaldado por una base de datos, y SQL se usa principalmente para recuperar y almacenar estos datos. Cada sitio web tiene su propia base de datos que contiene mucha información sobre los usuarios. Esta base de datos se utiliza tanto para recuperar información como para almacenarla. La mayoría de los sitios de comercio electrónico, **IRCTC**, sitios de reserva de películas, etcétera, utilizan lenguajes de consulta que están integrados en el código para realizar varias operaciones en sus datos según las necesidades del usuario.
- **Compatibilidad y flexibilidad:** SQL es compatible con bases de datos relacionales, como **Microsoft SQL Server**, **MS Access**, **Oracle**, **MySQL**, etcétera. También brinda flexibilidad, y otorga permiso para acceder y manipular qué tabla se usará en la base de datos. Puede administrar grandes registros y transacciones que ocurren en el sitio web con facilidad. Hay varias bibliotecas especiales presentes en SQL, como **SQLite**, que ayudan a conectar la aplicación web del cliente a la base de datos que usan los desarrolladores, lo que colabora para trabajar con los conjuntos de datos de los clientes.
- **Integración con otros lenguajes:** SQL se integra fácilmente con dos famosos lenguajes de programación: **Python** y **R**. Después de la

1. Conceptos preliminares

integración, puedes administrar la base de datos con estos lenguajes de secuencias de comandos. Esto es utilizado en especial por ingenieros de aprendizaje automático, analistas de datos, matemáticos, que manejan grandes cantidades de datos estadísticos a la vez.

- **SQL para ciencia y análisis de datos:** los lenguajes de secuencias de comandos se pueden integrar fácilmente con SQL. Los analistas trabajan con enormes conjuntos de datos en bases de datos relacionales para las que SQL es muy útil, ya que puede administrar conjuntos de datos más grandes con consultas simples. Un ejemplo básico donde se usa SQL es el filtrado de datos, en el que utilizas la cláusula **DONDE** con algunos operadores lógicos y condiciones. También, puedes realizar otras operaciones, como corte, indexación, agregaciones, etcétera, con los conjuntos de datos.
- **SQL para aprendizaje automático:** los ingenieros de aprendizaje automático trabajan con grandes conjuntos de datos para crear un modelo preciso, ya que “sin datos no hay *Machine Learning*”. Un gran ejemplo es **BigQuery ML**, una plataforma en la nube de Google. Ayuda en la creación y ejecución de varios modelos de Machine Learning utilizando varias consultas y herramientas en SQL. El aprendizaje automático con SQL y la integración con lenguajes de secuencias de comandos modernos con **Tensor Flow** son los próximos grandes avances para manejar datos descomunales.

Luego de ver en detalle en qué ocasiones podrás utilizar SQL, es hora de que te quedes con lo más importante. En definitiva podrás utilizar SQL para:

- Crear una nueva base de datos con SQL e insertar diversos datos en dicha base.
- Modificar o actualizar datos anteriores y recuperar datos de la base de datos.

- Eliminar datos y crear una nueva tabla en una base de datos o, incluso, eliminar la tabla.
- Establecer permisos para tablas, **procedimientos** y **vistas**, y crear funciones, vistas y procedimientos de almacenamiento.

Por lo tanto, considera que, en un sistema de gestión de bases de datos, todos los trabajos se realizan con la ayuda de SQL.

Las aplicaciones específicas de SQL se pueden resumir en el siguiente listado:

- SQL se utiliza como un lenguaje de definición de datos (**DDL**), lo que significa que puede crear una base de datos de forma autónoma, caracterizar su estructura, usarla y luego desecharla cuando haya terminado con ella. También se transmite como un lenguaje de control de datos (**DCL**) que determina cómo puede proteger tu base de datos contra la degradación y el uso indebido.
- SQL se usa como un lenguaje de manipulación de datos (**DML**), lo que implica que puedes usarlo para mantener una base de datos previamente existente. Por lo tanto, es un lenguaje increíble para ingresar, cambiar y separar información con respecto a una base de datos.
- Se usa ampliamente como un lenguaje de Cliente o Servidor para interconectar el front-end con el back-end y, en consecuencia, respaldar la arquitectura del cliente o del trabajador. Asimismo, se puede utilizar en el diseño de tres niveles de un cliente, un trabajador de la aplicación y una base de datos que caracteriza la arquitectura de internet.

Estos últimos párrafos resumen todo lo que necesitas saber antes de continuar.

Preparar el entorno

Para comenzar a trabajar con SQL necesitas un entorno con las herramientas adecuadas. Por su simplicidad lo harás con **SQLite Studio**. SQLite es una biblioteca que implementa un motor de base de datos SQL transaccional independiente, sin servidor y sin necesidad de configuración. El código para SQLite es de dominio público y, por lo tanto, es gratuito para cualquier uso, ya sea comercial o privado.

SQLite es una de las bases de datos más implementadas en el mundo; cuenta con una gran cantidad de aplicaciones, incluidos varios proyectos de alto perfil.

Puedes obtener más información en su sitio web:

<https://www.sqlite.org/about.html>.

El **GUI** o interfaz gráfica de usuario de SQLite se llama SQLite Studio y permite administrar bases de datos de forma gráfica y fácil.

Esta aplicación ofrece las siguientes características:

- Portátil: no es necesario instalarlo o desinstalarlo. Solo descarga, descomprime y ejecuta.
- Interfaz intuitiva.
- Potente pero ligero y rápido.
- Todas las características **SQLite3** y **SQLite2** incluidas en una GUI simple.
- Multiplataforma: se ejecuta en **Windows 9x / 2k / XP / 2003 / Vista / 7, Linux, macOS X**, y debería funcionar en otros sistemas **Unix** (aún no probados).
- Exportación a varios formatos (sentencias **SQL, CSV, HTML, XML, PDF, JSON**).
- Importación de datos de varios formatos (**CSV**, archivos de texto personalizados).
- Numerosas pequeñas adiciones, como el código de formato, el historial de consultas ejecutadas en las ventanas del editor, la verificación de la sintaxis sobre la marcha, y más.

- Soporte Unicode.
- Skinnable (la interfaz puede parecer nativa para Windows 9x / XP, KDE, GTK, macOS X o dibujar widgets para adaptarse a otros entornos, Windows Maker, etcétera).
- Colores configurables, fuentes y accesos directos.
- Código abierto y gratuito: lanzado bajo la licencia GPLv3.

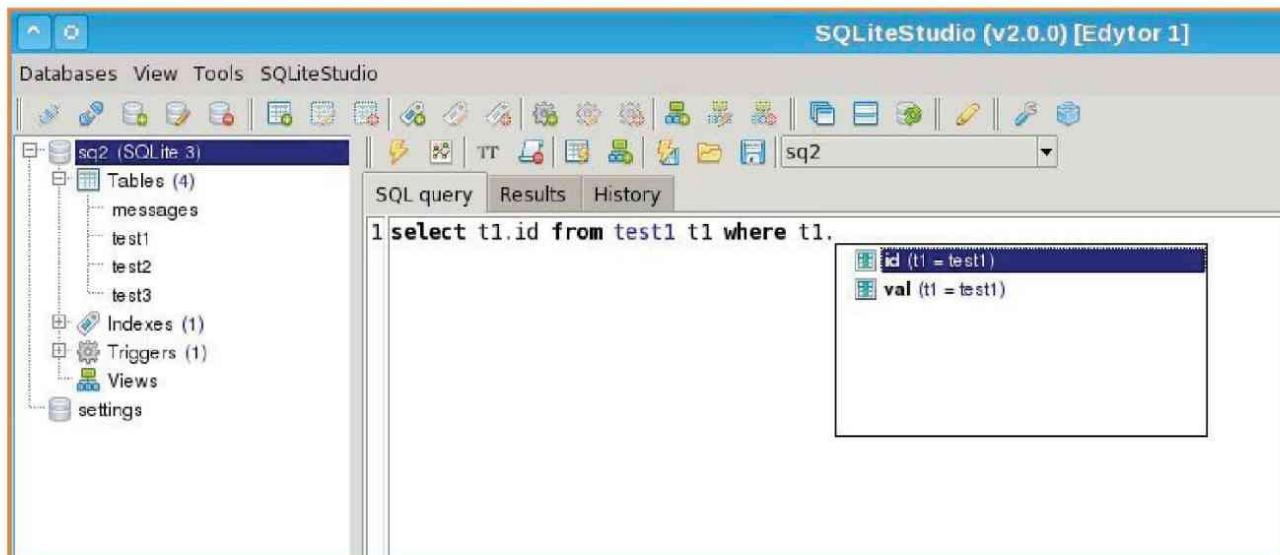


Figura 1. Una vez instalado, podrás acceder a la interfaz gráfica de SQLite Studio.

SQLite lee y escribe directamente, en archivos de disco ordinarios, el contenido de una base de datos y lo almacena en un solo archivo de disco multiplataforma, lo que le proporciona una gran versatilidad. En su funcionamiento, hay un equilibrio entre el uso de memoria y velocidad –a más memoria más rapidez de ejecución–, y tiene un rendimiento óptimo, incluso, en entornos con poca memoria.

Las funcionalidades y capacidades de SQLite se actualizan en forma continua para mejorar su fiabilidad y rendimiento al tiempo que mantiene la compatibilidad con la especificación publicada, la interfaz, la sintaxis SQL y el formato de archivo de base de datos.

SQLite es una excelente opción para comenzar pues considera los siguientes puntos importantes:

1. Conceptos preliminares

- Es muy útil para aplicaciones de escritorio y aplicaciones móviles. Por ejemplo, no sería adecuado instalar MySQL en una aplicación de escritorio o en una para dispositivos móviles, donde lo único que se quiere almacenar es una agenda de contactos.
- Es ideal para sistemas embebidos y aplicaciones relacionadas con *Internet of Things (IoT)*. Es una buena opción para usar en aplicaciones móviles, televisiones, consolas, cámaras, relojes, drones, etcétera.
- También es posible utilizar SQLite para sitios web que no tengan un tráfico elevado. La web oficial del proyecto (<https://www.sqlite.org>) utiliza una base de datos en SQLite.
- Te permite crear bases de datos que son almacenadas exclusivamente en memoria. Esta característica es útil para crear bases temporales con poca información, que no requieren un almacenamiento persistente.
- También son usadas con fines educativos debido a que no es necesario realizar ninguna configuración especial para trabajar con SQLite, y el proceso de instalación es muy sencillo.

Si te encuentras en un entorno GNU/Linux, para instalar SQLite debes hacer lo siguiente:

```
sudo apt-get update  
sudo apt-get install sqlite3
```

Una vez instalada la utilidad, solo debes escribir **sqlite3** en un terminal para iniciarla. Deberías obtener una salida similar a la que se muestra a continuación.

```
$ sqlite3  
SQLite version 3.22.0 2018-01-22 18:45:57
```

```
Enter ".help" for usage hints.  
Connected to a transient in-memory database.  
Use ".open FILENAME" to reopen on a persistent database.  
sqlite>
```

Por otra parte, si te encuentras en Windows o macOS, para instalar SQLite solo tendrás que descargar el archivo binario de la web oficial <https://www.sqlite.org/download.html>.

Alternativas online

Si recién estás comenzando con SQL o no te atreves a ejecutar un entorno como SQLite, es una buena idea practicar con algunas herramientas online. A continuación se presentan varias opciones.

SQLite Online

SQLite Online es un servicio gratis que se encuentra en internet y que te permite ejecutar de forma online los diferentes tipos de sentencias SQL. Además, puedes elegir entre los motores de bases de datos más usados en la actualidad: **MS SQL, PostgreSQL y MariaDB**.

Entre las características más importantes que ofrece esta opción, se encuentran las siguientes:

- Posibilidad de seleccionar entre los tres diferentes motores de bases de datos que se ubican en el lado izquierdo del sitio.
- Exportar el resultado en **XML, CSV** y **JSON** de las sentencias que ejecutes. Además, es posible exportar en archivos **.SQL** las sentencias que escribas.
- Abrir varias pestañas en el editor de texto.
- Registrarte como usuario y así poder guardar tus sesiones de trabajo.

1. Conceptos preliminares

SQL Fiddle

SQL Fiddle es otro sitio web gratis que te permite escribir y ejecutar tu código SQL y que, además, cuenta con varias características que lo hacen un buen recurso online, como por ejemplo la opción para elegir entre varios motores de bases de datos o plantillas para crear y consultar tablas.

Entre sus características se encuentran las siguientes:

- Posibilidad de elegir entre cinco motores de base de datos para comenzar a trabajar.
- Elección de la forma de visualización de la consulta entre columnas o texto plano.
- Opción de crear tablas a través de archivos CSV.

JDoodle

JDoodle es otro recurso online gratuito que, si bien te permite ejecutar código SQL online, también admite ejecutar código en diferentes lenguajes de programación, porque no está enfocado solo en lenguaje SQL y bases de datos. Sin embargo, no deja de ser una buena herramienta online gratuita. Sus características son las siguientes:

- Puedes elegir entre los motores de bases de datos MySQL y MongoDB.
- Si deseas guardar tus sesiones de trabajo, puedes hacerlo a través de la creación de un usuario en el sitio.
- Permite guardar y abrir archivos SQL.
- Te da la posibilidad de crear un link con el código que hayas escrito para insertarlo en tu blog o sitio web.

Volviendo a los ejemplos listados antes, podrás cargar uno de ellos en la interfaz web de JDoodle:

```
CREATE TABLE Cars (
    Id INT NOT NULL,
    CustomerId INT NOT NULL,
    EmployeeId INT NOT NULL,
    Model varchar(50) NOT NULL,
    Status varchar(25) NOT NULL,
    TotalCost INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (CustomerId) REFERENCES Customers(Id),
    FOREIGN KEY (EmployeeId) REFERENCES Employees(Id)
);
```

```
INSERT INTO Cars
    ([Id], [CustomerId], [EmployeeId], [Model], [Status],
    [TotalCost])
VALUES
    ('1', '1', '2', 'Ford F-150', 'READY', '230'),
    ('2', '1', '2', 'Ford F-150', 'READY', '200'),
    ('3', '2', '1', 'Ford Mustang', 'WAITING', '100'),
    ('4', '3', '3', 'Toyota Prius', 'WORKING', '1254');
```

Ahora agrega la siguiente línea al final:

```
SELECT * FROM Cars;
```

y ejecuta presionando el botón **Execute**. Verás lo siguiente:

1		1		2		Ford F-150		READY		230
---	--	---	--	---	--	------------	--	-------	--	-----

1. Conceptos preliminares

```
2|1|2|Ford F-150|READY|200  
3|2|1|Ford Mustang|WAITING|100  
4|3|3|Toyota Prius|WORKING|1254
```

Esto quiere decir que, gracias a las sentencias SQL anteriores, se ha creado la tabla, se agregaron datos y después se rescataron para ser mostrados en pantalla. Por ahora no te preocupes si no entiendes el código, solo es necesario que comprendas la lógica tras la ejecución de las sentencias SQL.

Online SQL IDE

```
1 CREATE TABLE Cars (  
2     Id INT NOT NULL,  
3     CustomerId INT NOT NULL,  
4     EmployeeId INT NOT NULL,  
5     Model varchar(50) NOT NULL,  
6     Status varchar(25) NOT NULL,  
7     TotalCost INT NOT NULL,  
8     PRIMARY KEY(Id),  
9     FOREIGN KEY (CustomerId) REFERENCES Customers(Id),  
10    FOREIGN KEY (EmployeeId) REFERENCES Employees(Id)  
11 );  
12  
13 INSERT INTO Cars  
14     ([Id], [CustomerId], [EmployeeId], [Model], [Status], [TotalCost])  
15 VALUES  
16     ('1', '1', '2', 'Ford F-150', 'READY', '200'),  
17     ('2', '1', '2', 'Ford F-150', 'READY', '200'),  
18     ('3', '2', '1', 'Ford Mustang', 'WAITING', '100'),  
19     ('4', '3', '3', 'Toyota Prius', 'WORKING', '1254');  
20  
21 SELECT * FROM Cars;
```

Execute Mode, Version, Inputs & Arguments

SQlite 3.37.0 Interactive

Result
executed in 0.005 sec(s)

```
1|1|2|Ford F-150|READY|200  
2|1|2|Ford F-150|READY|200  
3|2|1|Ford Mustang|WAITING|100  
4|3|3|Toyota Prius|WORKING|1254
```

Figura 2. En esta imagen puedes ver el resultado de las sentencias SQL ejecutadas en un entorno web.

Actividades

A continuación verás las preguntas que deberías saber responder para considerar aprendido el capítulo.

Test de autoevaluación

1. *¿Qué es SQL?*
 2. *¿Para qué sirve SQL?*
 3. *¿Identifica algunos ejemplos de sentencias SQL?*
 4. *¿Qué opciones en línea existen para que ejecutes código SQL?*
 5. *¿Qué es SQLite?*
 6. *¿Cómo funciona JDoodle?*
-

Capítulo 02



Elementos del lenguaje y sentencias

En este capítulo, conocerás los elementos de SQL que utilizarás más adelante para generar tus propias sentencias y, también, verás en detalle las partes que componen una sentencia SQL.

Al terminar este capítulo podrás realizar las siguientes tareas:

- ✿ Conocer los elementos básicos de SQL y los tipos de datos que puedes utilizar.
- ✿ Conocer la estructura de una sentencia SQL y la sintaxis básica de una consulta.

Campos y datos / 29

Sentencias SQL / 32

Sintaxis de una consulta / 33

 SELECT / 35

 ALL / DISTINCT / 37

 Nombres de campos / 39

 AS / 40

 FROM / 40

 WHERE / 40

 Condiciones / 42

 ORDER BY / 47

 ASC / DESC / 48

 Operadores lógicos / 49

Actividades / 51

Test de autoevaluación / 51

Campos y datos

Una base de datos está compuesta de tablas donde se almacenan registros catalogados en función de distintos campos.

Cada base de datos utiliza tipos de valores de campo que no necesariamente encontrarás en otras.

Sin embargo, existe un conjunto de tipos que están representados en la totalidad de estas bases:

Datos	Valores
Alfanuméricos	Contienen cifras y letras. Presentan una longitud limitada (255 caracteres).
Numéricos	Existen de varios tipos, principalmente, enteros (sin decimales) y reales (con decimales).
Booleanos	Poseen dos formas: Verdadero y falso (Sí o No).
Fechas	Almacenan fechas facilitando su posterior explotación. Este tipo de almacenamiento posibilita ordenar los registros por fechas o calcular los días entre una fecha y otra.
Memos	Son campos alfanuméricos de longitud ilimitada. Presentan el inconveniente de no poder ser indexados (verás más adelante qué quiere decir esto).
Autoincrementables	Son campos numéricos enteros que incrementan en una unidad su valor para cada registro incorporado. Su utilidad resulta más que evidente: servir de identificador ya que son exclusivos de un registro.

Por otra parte, los datos SQL se clasifican en trece tipos de datos primarios y de varios **sinónimos** válidos reconocidos por dichos tipos de datos.

2. Elementos del lenguaje y sentencias

Tipo de datos	Longitud	Descripción
BINARY	1 byte	Para consultas sobre tabla adjunta de productos de bases de datos que definen un tipo de datos binario.
BIT	1 byte	Valores Sí/No o True/False
BYTE	1 byte	Un valor entero entre 0 y 255.
COUNTER	4 bytes	Un número incrementado automáticamente (de tipo Long).
CURRENCY	8 bytes	Un entero escalable entre 922.337.203.685.477,5808 y 922.337.203.685.477,5807.
DATETIME	8 bytes	Un valor de fecha u hora entre los años 100 y 9999.
SINGLE	4 bytes	Un valor en punto flotante de precisión simple con un rango de -3.402823*1038 a -1.401298*10-45 para valores negativos, de 1.401298*10- 45 a 3.402823*1038 para valores positivos, y 0.
DOUBLE	8 bytes	Un valor en punto flotante de doble precisión con un rango de -1.79769313486232*10308 a -4.94065645841247*10-324 para valores negativos, de 4.94065645841247*10-324 a 1.79769313486232*10308 para valores positivos, y 0.
SHORT	2 bytes	Un entero corto entre -32,768 y 32,767.
LONG	4 bytes	Un entero largo entre -2,147,483,648 y 2,147,483,647.
LONGTEXT	1 byte por carácter	De 0 a un máximo de 1.2 gigabytes.
LONGBINARY	Según se necesite	De 0 a 1 gigabyte. Utilizado para objetos OLE.
TEXT	1 byte por carácter	De 0 a 255 caracteres.

A continuación puedes ver los sinónimos para los tipos de datos definidos:

Tipo de dato	Sinónimos
BINARY	VARBINARY
BIT	BOOLEAN LOGICAL LOGICAL1 YESNO
BYTE	INTEGER1
COUNTER	AUTOINCREMENT
CURRENCY	MONEY
DATETIME	DATE TIME TIMESTAMP
SINGLE	FLOAT4 IEEE SINGLE REAL
DOUBLE	FLOAT FLOAT8 IEEE DOUBLE NUMBER NUMERIC
SHORT	INTEGER2 SMALLINT
LONG	INT INTEGER INTEGER4
LONGBINARY	GENERAL OLEOBJECT
LONGTEXT	LONGCHAR MEMO NOTE

2. Elementos del lenguaje y sentencias

TEXT	ALPHANUMERIC CHAR - CHARACTER STRING - VARCHAR
VARIANT (No admitido).	VALUE

Sentencias SQL

En SQL hay bastantes sentencias que se utilizan para realizar diversas tareas. Dependiendo de las tareas, estas sentencias se pueden clasificar en tres grupos principales: **DML, DDL, DCL**.

Sentencia	Descripción
DML Manipulación de datos	
	SELECT Recupera datos de la base de datos.
	INSERT Añade nuevas filas de datos a la base de datos.
	DELETE Suprime filas de datos de la base de datos.
	UPDATE Modifica datos existentes en la base de datos.
DDL Definición de datos	
	CREATE TABLE Añade una nueva tabla a la base de datos.
	DROP TABLE Suprime una tabla de la base de datos.
	ALTER TABLE Modifica la estructura de una tabla existente.
	CREATE VIEW Añade una nueva vista a la base de datos.
	DROP VIEW Suprime una vista de la base de datos.
	CREATE INDEX Construye un índice para una columna.
	DROP INDEX Suprime el índice para una columna.
	CREATE SYNONYM Define un alias para un nombre de tabla.
	DROP SYNONYM Suprime un alias para un nombre de tabla.

DCL	Control de acceso	
	GRANT	Concede privilegios de acceso a usuarios.
	REVOKE	Suprime privilegios de acceso a usuarios
	Control de transacciones	
	COMMIT	Finaliza la transacción actual.
	ROLLBACK	Aborta la transacción actual.
PLSQL	SQL Programático	
	DECLARE	Define un cursor para una consulta.
	OPEN	Abre un cursor para recuperar resultados de consulta.
	FETCH	Recupera una fila de resultados de consulta.
	CLOSE	Cierra un cursor.

Sintaxis de una consulta

Es hora de que conozcas en detalle un **disparador** o consulta SQL, no te preocunes por entender cómo generarla pues en este punto solo es importante que reconozcas las partes que la conforman, y que puedas diferenciarlas de códigos generados con otros lenguajes.

Las instrucciones **DML** (*Data Manipulation Language* – Lenguaje de Manipulación de Datos) trabajan sobre los datos almacenados en un SGBD (sistema gestor de base de datos) y te permitirán consultarlos o modificarlos.

En general, las operaciones básicas de manipulación de datos que puedes realizar con SQL se denominan **operaciones CRUD** (*Create, Read, Update and Delete*, o sea, crear, leer, actualizar y borrar).

Existen cuatro instrucciones básicas para realizar estas tareas:

- **INSERT**: inserta filas en una tabla. Se corresponde con la **C** de CRUD.
- **SELECT**: muestra información sobre los datos almacenados en la base de datos. Dicha información puede pertenecer a una o varias tablas. Es la **R**.

2. Elementos del lenguaje y sentencias

- **UPDATE:** actualiza información de una tabla. Es, obviamente, la **U**.
- **DELETE:** borra filas de una tabla. Se corresponde con la **D**.

Ahora comenzarás a experimentar la forma en que puedes recuperar la información que te interesa desde el interior de una base de datos, usando para ello el lenguaje de consulta SQL.

Para realizar consultas sobre las tablas de las bases de datos, dispones de la instrucción **SELECT**. Con ella puedes consultar una o varias tablas. Se trata del comando más versátil del lenguaje SQL por lo que su uso es imprescindible.

Existen muchas cláusulas asociadas a la sentencia **SELECT, GROUP BY, ORDER, HAVING, UNION**. También es una de las instrucciones en la que, con más frecuencia, los motores de bases de datos incorporan cláusulas adicionales al estándar.

A continuación, conocerás las consultas simples, basadas en una sola tabla. Verás cómo obtener filas y columnas de una tabla en el orden en que haga falta.

El resultado de una consulta **SELECT** devuelve una tabla lógica. Es decir, los resultados son una relación de datos, que tiene filas/registros con una serie de campos/columnas, igual que cualquier tabla de la base de datos. Sin embargo, esta tabla permanece en la memoria mientras la utilices y, luego, se descarta. Cada vez que ejecutas la consulta se vuelve a calcular el resultado.

La sintaxis básica de una consulta **SELECT** es la siguiente (los valores opcionales van entre corchetes):

```
SELECT [ ALL / DISTINCT ] [ * ] / [ListaColumnas_
Expresiones] AS
[Expresion]
FROM Nombre_Tabla_Vista
```

```
WHERE Condiciones  
ORDER BY ListaColumnas [ ASC / DESC ]
```

Las partes de la consulta anterior se explican a continuación.

SELECT

Permite seleccionar las columnas que se van a mostrar y en el orden en que lo harán. Simplemente es la instrucción que la base de datos interpreta como que se va a solicitar información.

Por ejemplo, si deseas devolver todos los campos de una tabla, debes utilizar **SELECT ***:

```
SELECT *  
FROM CLIENTES
```

Con el ***** indicas que quieres devolver todos los campos. Si **CLIENTES** dispone de los campos **idCliente**, **nombre** y **descripcion**, lo anterior sería equivalente a:

```
SELECT idCliente, nombre, descripcion  
FROM CLIENTES
```

Si quieres devolver todos los campos, lo anterior es innecesario y, por lo tanto, es más conveniente emplear el asterisco (*****). También sería equivalente usar la notación completa:

2. Elementos del lenguaje y sentencias

```
SELECT CLIENTES.idCliente, CLIENTES.nombre, CLIENTES.descripcion  
FROM CLIENTES
```

Al tener únicamente una tabla involucrada, puedes referirte a los campos sin calificar, dado que no hay duda de a qué tabla se refiere. Cuando veas consultas sobre varias tablas, comprenderás la importancia de incluir la notación completa: **TABLA.campo**.

Otra opción sería devolver un subconjunto de los campos de una tabla:

SELECT DISTINCT:

```
SELECT cp, ciudad  
FROM DIRECCION
```

Esta consulta devolverá únicamente los campos **cp** (código postal) y **ciudad** de la tabla **DIRECCION**. Al existir un subconjunto de los campos, estos no tienen por qué incluir la clave de la tabla, por lo que no tienen por qué ser únicos. Así, si posees muchos registros referidos a distintas calles y números de ese mismo código postal y ciudad, te encontrarás numerosos registros repetidos. Esto puede evitarse con lo siguiente:

```
SELECT DISTINCT cp, ciudad  
FROM CLIENTES
```

Así, se eliminan los registros repetidos, devolviendo únicamente una vez cada par **cp, ciudad**. Esta selección corresponde a un subconjunto de los datos de la tabla que excluye los repetidos.

ALL / DISTINCT

ALL es el valor predeterminado, especifica que el conjunto de resultados puede incluir filas duplicadas. Por regla general, nunca se utiliza.

DISTINCT especifica que el conjunto de resultados solo puede incluir filas únicas. Es decir que, si al realizar una consulta hay registros exactamente iguales que aparecen más de una vez, estos se eliminan.

Los operadores de comparación que presentan una subconsulta se pueden modificar mediante las palabras clave **ALL**, **ANY** o **SOME**.

Se utiliza este tipo de comparación cuando se quiere comparar el resultado de la expresión con una lista de valores y actuar en función del modificador empleado.

ANY significa que, para que una fila de la consulta externa satisfaga la condición especificada, la comparación se debe cumplir para al menos un valor de los devueltos por la subconsulta.

Por cada fila de la consulta externa, se evalúa la comparación con cada uno de los valores devueltos por la subconsulta y, si la comparación es

True para alguno de los valores **ANY**, es verdadero. Si la comparación no se cumple con ninguno de los valores de la consulta, **ANY** da **False** a no ser que todos los valores devueltos por la subconsulta sean nulos, en tal caso **ANY** dará **NULL**.

Si la subconsulta no devuelve filas **ANY**, da **False** incluso si la expresión es nula.

```
SELECT *
FROM empleados
WHERE cuota > ANY (SELECT cuota
                     FROM empleados empleados2)
```

2. Elementos del lenguaje y sentencias

```
WHERE empleados.oficina = empleados2.oficina);
```

Obtendrás los empleados que tienen una cuota superior a la cuota de alguno de sus compañeros de oficina, es decir, aquellos que no tengan la menor cuota de su oficina.

En este caso, se usa un alias de tabla en la subconsulta **empleados2** para poder utilizar una referencia externa.

Por otra parte, para que se cumpla la condición con el modificador **ALL**, la comparación se debe cumplir con cada uno de los valores devueltos por la subconsulta. Si la subconsulta no devuelve ninguna fila **ALL**, da **True**.

```
SELECT *
```

```
FROM empleados
```

```
WHERE cuota > ALL (SELECT cuota
```

```
FROM empleados empleados2
```

```
WHERE empleados.oficina = empleados2.oficina);
```

Aquí obtendrás los empleados que tengan una cuota superior a todas las cuotas de la oficina. Se podría pensar que se obtiene el empleado de mayor cuota de su oficina, pero no es así, aquí se presenta un problema: la cuota del empleado aparece en el resultado de subconsulta, por lo tanto, **>** no se cumplirá para todos los valores y solo saldrán los empleados que no tengan oficina (para los que la subconsulta no devuelve filas).

Para salvar el problema, tendrías que quitar del resultado de la subconsulta la cuota del empleado modificando el **WHERE**:

```
WHERE empleados.oficina = empleados2.oficina  
      AND empleados.numemp <> empleados2.numemp);
```

De esta forma saldrían los empleados que tienen una cuota mayor que cualquier otro empleado de su misma oficina.

O bien

```
WHERE empleados.oficina = empleados2.oficina  
      AND empleados.cuota <> empleados2.cuota);
```

para no considerar los empleados que tengan la misma cuota que el empleado que te interesa.

Nombres de campos

Se debe especificar una lista de nombres de campos de la tabla que te interesan y que, por tanto, quieres devolver. Normalmente habrá más de uno, en cuyo caso separa cada nombre de los demás mediante comas.

Se puede anteponer el nombre de la tabla al nombre de las columnas, utilizando el formato **Tabla.Columna**. Además de nombres de columnas, en esta lista se pueden agregar constantes, expresiones aritméticas y funciones, para obtener campos calculados de manera dinámica.

Si deseas que te devuelva todos los campos de la tabla, utiliza el comodín * (asterisco).

Los nombres indicados deben coincidir exactamente con los nombres de los campos de la tabla.

2. Elementos del lenguaje y sentencias

AS

Permite renombrar columnas si lo utilizas en la cláusula **SELECT**, o renombrar tablas si lo utilizas en la cláusula **FROM**. Es opcional. Con ello podrás crear diversos alias de columnas y tablas.

En definitiva, permite definir un alias o nombre alternativo a los nombres originales de las columnas que vaya a devolver un Query. Por ejemplo, en el siguiente código:

```
SELECT Columna1, Columna2 as OtroNombre, Columna3+Columna4  
as ColumnaCalculada  
FROM TablaX
```

la columna **Columna2** será devuelta por el Query como **OtroNombre**, y el resultado de sumar **Columna3** y **Columna4** será devuelto en una columna con el nombre **ColumnaCalculada**.

FROM

Esta cláusula permite indicar las tablas o vistas de las cuales vas a obtener la información. De momento, verás ejemplos para obtener información de una sola tabla.

Como se ha indicado antes, también se pueden renombrar las tablas usando la instrucción **AS**.

WHERE

Especifica la condición de filtro de las filas devueltas. Se utiliza cuando no se desea que se devuelvan todas las filas de una tabla, sino solo las que cumplen ciertas condiciones. Lo habitual es utilizar esta cláusula en la mayoría de las consultas.

La cláusula **WHERE** puede usarse para determinar qué registros de las tablas enumeradas en la cláusula **FROM** aparecerán en los resultados de la instrucción **SELECT**. Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla. **WHERE** es opcional, pero cuando aparece debe ir a continuación de **FROM**. Ahora verás algunos ejemplos:

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario  
> 21000;
```

```
SELECT Id_Producto, Existencias FROM Productos  
WHERE Existencias <= Nuevo_Pedido;
```

```
SELECT * FROM Pedidos WHERE Fecha_Envio = #5/10/94#;
```

```
SELECT Apellidos, Nombre FROM Empleados WHERE  
Apellidos = 'King';
```

```
SELECT Apellidos, Nombre FROM Empleados WHERE  
Apellidos Like 'S*';
```

```
SELECT Apellidos, Salario FROM Empleados WHERE Salario  
Between 200 And 300;
```

```
SELECT Apellidos, Salario FROM Empl WHERE Apellidos  
Between 'Lon' And 'Tol';
```

```
SELECT Id_Pedido, Fecha_Pedido FROM Pedidos WHERE  
Fecha_Pedido  
Between #1-1-94# And #30-6-94#;
```

2. Elementos del lenguaje y sentencias

```
SELECT Apellidos, Nombre, Ciudad FROM Empleados WHERE
Ciudad
In ('Sevilla', 'Los Angeles', 'Barcelona');
```

Condiciones

Son expresiones lógicas por comprobar para la condición de filtro, que tras su resolución devuelven para cada fila **TRUE** o **FALSE**, en función de que se cumplan o no. Se puede utilizar cualquier expresión lógica y en ella incluir diversos operadores como:

- > :mayor
- >=: mayor o igual
- <: menor
- <=: menor o igual
- =: igual
- <> o !=: distinto
- **IS [NOT] NULL**: para comprobar si el valor de una columna es o no es nula, es decir, si contiene o no contiene algún valor.

Una columna de una fila es **NULL** si está completamente vacía. Hay que tener en cuenta que, si se ha introducido cualquier dato, incluso en un campo alfanumérico o si se introduce una cadena en blanco o un cero en un campo numérico, deja de ser **NULL**.

- **LIKE**: para la comparación de un modelo. Para ello utiliza los caracteres comodín especiales: **%** y **_**. Con el primero, se indica que en su lugar puede ir cualquier cadena de caracteres, y con el segundo, que puede ir cualquier carácter individual (un solo carácter). Con la combinación de estos caracteres, es posible obtener múltiples patrones de búsqueda. Por ejemplo:

- El nombre empieza con **A: Nombre LIKE 'A%'**.
- El nombre acaba con **A: Nombre LIKE '%A'**.
- El nombre contiene la letra **A: Nombre LIKE '%A%**.
- El nombre empieza con **A** y después contiene un solo carácter cualquiera: **Nombre LIKE 'A_'**.
- El nombre empieza con una **A**, después cualquier carácter; luego, una **E** y, al final, cualquier cadena de caracteres: **Nombre LIKE 'A_E%'**.

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

expresión LIKE modelo

en donde **expresión** es una cadena modelo o campo contra el que se compara otra expresión. Se puede utilizar el operador **Like** para encontrar valores en los campos que coincidan con el modelo especificado. Por modelo se puede especificar un valor completo (**Ana María**), o se pueden utilizar caracteres comodín como los reconocidos por el sistema operativo para encontrar un rango de valores (**Like An***).

El operador **Like** se puede utilizar en una expresión para comparar un valor de un campo con una expresión de cadena. Por ejemplo, si se introduce **Like C*** en una consulta SQL, la consulta devuelve todos los valores de campo que comiencen por la letra **C**.

El ejemplo siguiente devuelve los datos que comienzan con la letra **P** seguido de cualquier letra entre **A** y **F**, y de tres dígitos:

LIKE 'P[A-F]###'

2. Elementos del lenguaje y sentencias

Este ejemplo devuelve los campos cuyo contenido empieza con una letra de la **A** a la **D** seguidas de cualquier cadena.

Like '[A-D]*'

En la tabla siguiente, se muestra cómo utilizar el operador **LIKE** para comprobar expresiones con diferentes modelos.

Tipo de coincidencia	Modelo Planteado	Coincide	No coincide
Varios caracteres	'a*a'	'aa', 'aBa', 'aBBBBa'	'aBC'
Carácter especial	'a[*]a'	'a*a'	'aaa'
Varios caracteres	'ab*'	'abcdefg', 'abc'	'cab', 'aab'
Un solo carácter	'a?a'	'aaa', 'a3a', 'aBa'	'aBBBBa'
Un solo dígito	'a#a'	'a0a', 'a1a', 'a2a'	'aaa', 'a10a'
Rango de caracteres	'[a-z]'	'f', 'p', 'j'	'2', '&'
Fuera de un rango	'[!a-z]'	'9', '&', '%'	'b', 'a'
Distinto de un dígito	'[!0-9]'	'A', 'a', '&', '~'	'0', '1', '9'
Combinada	'a[!b-m]#'	'An9', 'az0', 'a99'	'abc', 'aj0'

● **BETWEEN:** para un intervalo de valores. Por ejemplo:

- Clientes entre el 30 y el 100: **CodCliente BETWEEN 30 AND 100.**
- Clientes nacidos entre 1970 y 1979: **FechaNac BETWEEN '19700101' AND '19791231'.**

Para indicar que deseas recuperar los registros según el intervalo de valores de un campo, emplea el operador **BETWEEN**, cuya sintaxis es:

```
campo [Not] Between valor1 And valor2 (la condición Not es  
opcional)
```

En este caso, la consulta devolverá los registros que contengan en **campo** un valor incluido en el intervalo **valor1, valor2** (ambos inclusive). Si antepones la condición **Not**, devolverá aquellos valores no incluidos en el intervalo.

```
SELECT * FROM Pedidos WHERE CodPostal Between 28000 And  
28999;
```

```
SELECT IIf(CodPostal Between 28000 And 28999,  
'Provincial', 'Nacional')  
FROM Editores;
```

● **IN()**: para especificar una relación de valores concretos. Por ejemplo: Ventas de los Clientes 10, 15, 30 y 75: **CodCliente IN(10, 15, 30, 75)**. Por supuesto, es posible combinar varias condiciones simples de los operadores anteriores utilizando los operadores lógicos **OR**, **AND** y **NOT**, así como el uso de paréntesis para controlar la prioridad de los operadores (como en matemáticas). Por ejemplo: ... **Cliente = 100 AND Provincia = 30) OR Ventas > 1000**... que sería para los clientes de las provincias 100 y 30 o cualquier cliente cuyas ventas superen 1000. Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los de una lista. Su sintaxis es:

2. Elementos del lenguaje y sentencias

expresión [Not] In(valor1, valor2, . . .)

```
SELECT * FROM Pedidos WHERE Provincia In ('Madrid',  
'Barcelona', 'Sevilla');
```

En la tabla siguiente, se describen las condiciones que puedes utilizar en la consulta SQL.

Condición	Descripción
EQUAL	<p>Comprueba si dos expresiones son o no iguales. Si las expresiones son iguales, la condición es verdadera y se devuelven los registros coincidentes.</p> <p>Cuando se ejecuta la siguiente sentencia SQL para la condición igual y devuelve registros en los que el identificado de cliente es igual a Smith:</p> <pre>SELECT * FROM Customers WHERE CustomerID=1</pre>
BETWEEN	<p>Comprueba valores entre un rango determinado y devuelve los valores coincidentes.</p> <p>La condición BETWEEN es inclusiva. Los valores inicial y final están incluidos.</p> <p>Sintaxis de BETWEEN:</p> <pre>SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;</pre>

LIKE	<p>Realiza y devuelve una coincidencia de patrón utilizando comodines en la cláusula WHERE de una sentencia SELECT.</p> <p>Se utilizan dos comodines junto con el operador LIKE:</p> <ul style="list-style-type: none"> %: el signo de porcentaje representa cero, uno o varios caracteres; _ : el guion bajo representa un único carácter. <p>Sintaxis de LIKE:</p> <pre>SELECT column1, column2, ... FROM table_name WHERE column LIKE pattern</pre>
IN	<p>Igual a cualquier valor de una lista de valores.</p> <p>Sintaxis de IN:</p> <pre>SELECT column_name1, column_name2, etc FROM table_name WHERE column_name1 IN (value1, value2, etc);</pre>
NOT IN	<p>Comprueba si dos expresiones son o no iguales. Si las expresiones no son iguales, la condición es verdadera y devuelve los registros no coincidentes.</p> <p>Sintaxis de NOT IN:</p> <pre>SELECT column_name1, column_name2, etc FROM table_name WHERE column_name1 NOT IN (value1, value2, etc);</pre>

ORDER BY

Define el orden de las filas del conjunto de resultados. Se especifica el campo o los campos (separados por comas) por los cuales quieras ordenar los resultados.

Se puede especificar el orden en que se desean recuperar los registros de

2. Elementos del lenguaje y sentencias

las tablas mediante la cláusula **ORDER BY Lista de Campos**. En donde **Lista de campos** representa los campos por ordenar. Ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;
```

Esta consulta devuelve los campos **CodigoPostal**, **Nombre**, **Telefono** de la tabla **Clientes** ordenados por el campo **Nombre**.

Se pueden ordenar los registros por más de un campo, como por ejemplo:

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;
```

Incluso, es posible especificar el orden de los registros: ascendente mediante la cláusula **ASC**, que se toma por defecto, o descendente: **DESC**.

```
SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre ASC;
```

ASC / DESC

ASC es el valor predeterminado, especifica que la columna indicada en la cláusula **ORDER BY** se ordenará de forma ascendente, o sea, de menor a mayor. Si por el contrario se especifica **DESC**, se ordenará de forma descendente, de mayor a menor.

Por ejemplo, para ordenar los resultados de forma ascendente por ciudad, y los que sean de la misma ciudad de forma descendente por nombre, utilizarás esta cláusula de ordenación:

```
... ORDER BY Ciudad, Nombre DESC ...
```

Como a la columna **Ciudad** no le has puesto **ASC** o **DESC**, se usará para esta el valor predeterminado, que es **ASC**.

Operadores lógicos

Los operadores lógicos soportados por SQL son: **AND**, **OR**, **XOR**, **Eqv**, **Imp**, **Is** y **Not**. A excepción de los dos últimos, todos poseen la siguiente sintaxis:

```
<expresión1>     operador     <expresión2>
```

en donde **expresión1** y **expresión2** son las condiciones por evaluar; el resultado de la operación varía en función del operador lógico.

A continuación se muestra la tabla de verdad.

<expresión1>	Operador	<expresión2>	Resultado
Verdad	AND	Falso	Falso
Verdad	AND	Verdad	Verdad
Falso	AND	Verdad	Falso
Falso	AND	Falso	Falso
Verdad	OR	Falso	Verdad
Verdad	OR	Verdad	Verdad
Falso	OR	Verdad	Verdad
Falso	OR	Falso	Falso

2. Elementos del lenguaje y sentencias

Verdad	XOR	Verdad	Falso
Verdad	XOR	Falso	Verdad
Falso	XOR	Verdad	Verdad
Falso	XOR	Falso	Falso
Verdad	Eqv	Verdad	Verdad
Verdad	Eqv	Falso	Falso
Falso	Eqv	Verdad	Falso
Falso	Eqv	Falso	Verdad
Verdad	Imp	Verdad	Verdad
Verdad	Imp	Falso	Falso
Verdad	Imp	Null	Null
Falso	Imp	Verdad	Verdad
Falso	Imp	Falso	Verdad
Falso	Imp	Null	Verdad
Null	Imp	Verdad	Verdad
Null	Imp	Falso	Null
Null	Imp	Null	Null

Si a cualquiera de las condiciones le antepones el operador **NOT**, el resultado de la operación será el contrario al devuelto sin el operador **NOT**.
El operador **Is** se emplea para comparar dos variables de tipo objeto **<Objeto1> Is <Objeto2>**, este operador devuelve verdad si los dos objetos son iguales

```
SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;  
SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;  
SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';  
SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR  
(Provincia = 'Madrid' AND Estado = 'Casado');
```

Actividades

A continuación verás las preguntas que deberías saber responder para considerar aprendido el capítulo.

Test de autoevaluación

1. *¿Describe los tipos de datos que puedes encontrar en una base de datos?*
 2. *Indica los sinónimos de **LONG**.*
 3. *Menciona las sentencias de manipulación de datos de SQL.*
 4. *¿Qué tipo de sentencias son **DROP TABLE** y **ALTER TABLE**?
¿Cuál es la función de cada una de ellas?*
 5. *Indica las partes más importantes en la sintaxis de una consulta SQL.*
 6. *¿Qué son las condiciones? Indica algunos ejemplos de su uso.*
-

Capítulo 03



Bases de datos y tablas

En este capítulo, comenzarás a trabajar en forma directa con las sentencias que te permitirán manipular bases de datos y tablas. Una vez que lo termines, deberás ser capaz de llevar a cabo muchas de las tareas referidas a las bases de datos.

Al terminar este capítulo, podrás realizar las siguientes tareas:

- ✿ Crear y eliminar una base de datos.
- ✿ Crear y eliminar una tabla.
- ✿ Modificar tablas existentes.

Bases de datos / 53

CREATE DATABASE / 53

Crear una tabla / 54

Estructura de una tabla / 58

CONSTRAINT / 63

Modificar una tabla / 65

Eliminar una tabla / 66

Actividades / 71

Test de autoevaluación / 71

Bases de datos



Dispones de una sentencia que te permite crear un **esquema**, denominada **CREATE SCHEMA**. Gracias a la creación de esquemas, podrás agrupar un conjunto de elementos de la base de datos que son propiedad de un usuario.

La sintaxis es la siguiente:

```
CREATE SCHEMA {[nombre_esquema]} | [AUTHORIZATION  
usuario] [lista_de_elementos_del_esquema];
```

La sentencia de creación de esquemas hace que varias tablas se puedan agrupar bajo un mismo nombre y que tengan un propietario (usuario). Aunque todos los parámetros son opcionales, como mínimo se debe indicar el nombre del esquema o el nombre del usuario propietario de la base de datos.

Para borrar una base de datos, puedes proceder de la misma forma, es decir, utilizando **DROP SCHEMA**, que presenta la siguiente sintaxis:

```
DROP SCHEMA nombre_esquema {RESTRICT|CASCADE};
```

CREATE DATABASE

La sentencia **CREATE DATABASE** se utiliza para crear una nueva base de datos SQL. Su sintaxis es la siguiente:

```
create database nombre_basededatos ON PRIMARY
```

3. Bases de datos y tablas

Ejemplo de funcionamiento:

```
CREATE DATABASE VENTAS ON PRIMARY (
    NAME=VENTAS_data, FILENAME='c:\VENTAS.mdf',
    SIZE=5MB, MAXSIZE=10MB, FILEGROWTH=1MB )
LOG ON
(
    NAME=VENTAS_log, FILENAME='c:\VENTAS.ldf', SIZE=5MB,
    MAXSIZE=10MB, FILEGROWTH=1MB
)
```

Crear una tabla



La sentencia **CREATE TABLE** permite crear una tabla.

Para hacerlo, hay que mencionar como mínimo el nombre de la tabla y un campo.

```
CREATE TABLE <nombre_de_tabla> (
    <campo_uno> <tipo_campo_uno>,
    <campo_dos> <tipo_campo_dos>,
    ...);
```

También puedes crear una tabla sobre la base de una o varias tablas existentes.

```
CREATE TABLE <nombre_de_tabla> AS <SELECT STATEMENT>;
```

Encontrarás que la mayoría de las bases de datos ofrecen editores que permiten realizar la creación de tablas en forma rápida y sencilla. Pero en algunos casos podría ser necesario crearlas utilizando las sentencias SQL que se presentaron en los párrafos anteriores.

En cualquier caso, partir desde una sentencia SQL podría ahorrarte muchos problemas.

Estas sentencias son bastante útiles para bases de datos como MySQL, que trabajan en forma directa con comandos SQL.

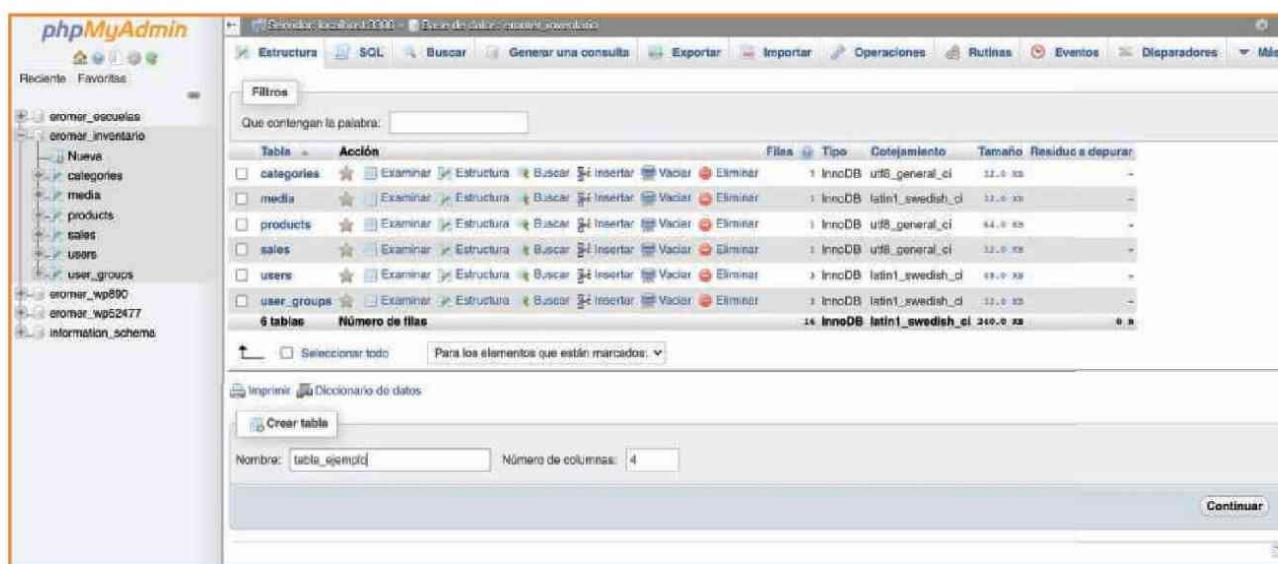


Figura 3.1. Herramienta para crear tablas en phpMyAdmin.

Para crear una tabla mediante una sentencia SQL, debes especificar diversos datos: el nombre que deseas asignar, los nombres de los campos y sus características. Además, puede ser necesario indicar cuáles campos serán índices. Es importante considerar que la sintaxis de creación podría variar ligeramente de una base de datos a otra.

Sintaxis completa:

```
Create Table nombre_tabla
(
```

3. Bases de datos y tablas

```
nombre_campo_1 tipo_1  
nombre_campo_2 tipo_2  
nombre_campo_n tipo_n  
Key(campo_x,...)  
)
```

Ejemplo:

```
Create Table pedidos  
(  
id_pedido INT(4) NOT NULL AUTO_INCREMENT,  
id_cliente INT(4) NOT NULL,  
id_articulo INT(4)NOT NULL,  
fecha DATE,  
cantidad INT(4),  
total INT(4), KEY(id_pedido,id_cliente,id_articulo)  
)
```

Para **id_pedido** se incrementa automáticamente con **AUTO_INCREMENT** y, para evitar un mensaje de error, los campos que van a ser definidos como índices no pueden ser nulos: **NOT NULL**.

El campo **fecha** se almacena con formato de fecha **DATE**, y los índices se definen enumerándolos entre paréntesis precedidos de la palabra **KEY** o **INDEX**.

Otro ejemplo:

```
Create Table articulos  
(
```

```
id_articulo INT(4) NOT NULL AUTO_INCREMENT,  
titulo VARCHAR(50),  
autor VARCHAR(25),  
editorial VARCHAR(25),  
precio REAL,  
KEY(id_articulo)  
)
```

En este caso, los campos alfanuméricos son introducidos de la misma forma que los numéricos. Ahora verás otros ejemplos prácticos:

Creación de **Tabla Cliente**:

```
CREATE TABLE CLIENTE  
(NUMCLI INT not null,  
NOMCLI CHAR(30) not null, DIRCLI char(30), FAX INT, E_MAIL  
CHAR(30) DEFAULT ('Desconocido'),  
SALD_0_30 DECIMAL (10,2), SALD_31_60 DECIMAL (10,2),  
SALD_61_90 DECIMAL (10,2), primary key (NUMCLI) )
```

Creación de **Tabla Vendedor**:

```
CREATE TABLE VENDEDOR (CODVEND INT not null, NOMVEND  
char(20) not null,APELLVEND char(20) not null,  
DIRVEND char(30), TELVEND INT, E_MAIL CHAR(30)  
DEFAULT('Desconocido'),  
CUOTA DECIMAL (10,2), VENTAS DECIMAL (10,2), primary key  
(CODVEND) )
```

3. Bases de datos y tablas

Creación de **Tabla Artículo**:

```
CREATE TABLE ARTICULO (NUMART char(4) not null PRIMARY KEY, DESCRIPCION CHAR(30), PRECIO DECIMAL (10,2) NOT NULL CHECK (PRECIO >= 0.00), EXISTENCIA INT, CATEGORIA_ART CHAR (15))
```

Creación de **Tabla Pedido**:

```
CREATE TABLE PEDIDO (NUMPED INT not null PRIMARY KEY, NUMCLI INT not null, FECHA_PED DATETIME, TOT_DESC DECIMAL (10,2), FOREIGN KEY (NUMCLI) REFERENCES CLIENTE(NUMCLI) , FOREIGN KEY (CODVEND) REFERENCES VENDEDOR(CODVEND))
```

Creación de **Tabla Detalle_Ped**:

```
CREATE TABLE DETALLE_PED (NUMPED INT not null, NUMART char (4) not null, CANTIDAD INT CHECK (CANTIDAD >= 0), PRIMARY KEY (NUMPED,NUMART), FOREIGN KEY (NUMPED) REFERENCES PEDIDO(NUMPED), FOREIGN KEY (NUMART) REFERENCES ARTICULO (NUMART) )
```

Estructura de una tabla

Las **tablas** son los objetos de una base de datos donde se almacenan los datos. A continuación se muestra un ejemplo de una tabla típica:

EMP_NO	APPELLIDO	OFICIO	DIRECTOR	FECHA_AL	SALARIO	COMISION	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	140000	40000	30
7521	LÓPEZ	EMPLEADO	7782	08/05/81	135000		10
7654	MARTÍN	VENDEDOR	7698	28/09/81	150000	160000	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	385000		30
7782	MARTÍNEZ	DIRECTOR	7839	09/06/81	245000		10
7839	REY	PRESIDENTE		17/11/81	600000		10
7844	CALVO	VENDEDOR	7698	08/09/81	180000	0	30
7876	GIL	ANALISTA	7782	06/05/82	335000		20
7900	JIMÉNEZ	EMPLEADO	7782	24/03/83	140000		20

Las tablas están formadas por **filas** y **columnas**. Por una parte, cada fila representa una ocurrencia de la entidad, por ejemplo, un empleado si se trata de una tabla de empleados; un departamento si estás frente a una tabla de departamentos, o un cliente si se trata de una tabla de clientes, entre otros.

Por otra parte, cada columna representa un atributo o una característica de la entidad. Tiene un nombre y puede incluir un conjunto de valores.

Por ejemplo, la tabla de empleados puede tener como columnas o atributos los siguientes: número de empleado, nombre, fecha de alta, salario, etcétera.

Esta es la sintaxis de creación presentada en la sección anterior:

```
CREATE TABLE tabla (
campo1 tipo (tamaño) índice1,
campo2 tipo (tamaño) índice2, ... ,
índice multicampo , ... )
```

3. Bases de datos y tablas

- **tabla**: es el nombre de la tabla que se va a crear.
- **campo1, campo2**: es el nombre del campo o de los campos que se van a crear en la nueva tabla, que debe contener, al menos, un campo.
- **tipo**: es el tipo de datos de campo en la nueva tabla.
- **tamaño**: es el tamaño del campo; solo se aplica para campos de tipo texto.
- **índice1, índice2**: es una cláusula **CONSTRAINT** que define el tipo de índice por crear. Esta cláusula es opcional.
- **índice multicampos**: es una cláusula **CONSTRAINT** que define el tipo de índice multicampos por crear. Ten en cuenta que un índice multicampo es aquel que está indexado por el contenido de varios campos. Esta cláusula es opcional.

Ejemplo:

```
CREATE TABLE  
    Empleados (  
        Nombre TEXT (25),  
        Apellidos TEXT (50)  
    )
```

Esta sentencia creará una nueva tabla llamada **Empleados** con dos campos, uno llamado **Nombre** de tipo texto y longitud 25, y otro denominado **Apellidos** con longitud 50.

```
CREATE TABLE  
    Empleados (  
        Nombre TEXT (10),  
        Apellidos TEXT,
```

```

FechaNacimiento DATETIME
)
CONSTRAINT
    IndiceGeneral
    UNIQUE (
        Nombre, Apellidos, FechaNacimiento
    )

```

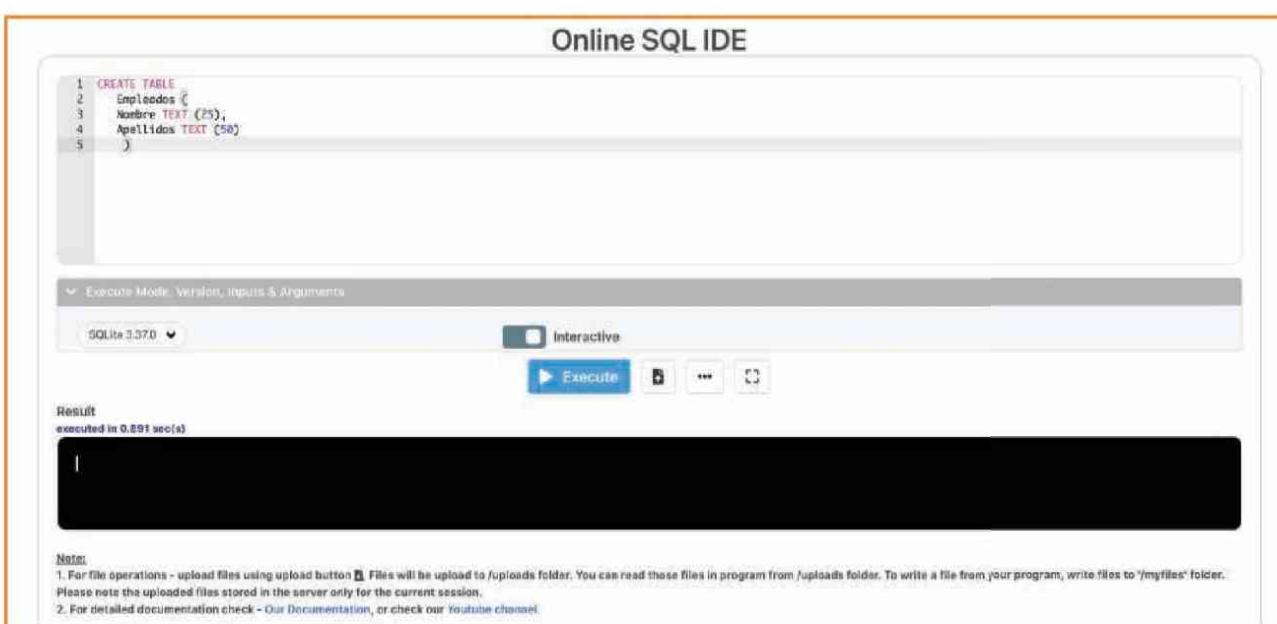


Figura 3.2. Sentencia SQL ejecutada en un entorno web.

La sentencia anterior crea una nueva tabla llamada **Empleados** con un campo **Nombre** de tipo texto y longitud 10; otro denominado **Apellidos**, de tipo texto y longitud predeterminada 50, y uno más llamado **FechaNacimiento** de tipo Fecha/Hora. También crea un índice único que no permite valores repetidos formado por los tres campos.

```

CREATE TABLE
    Empleados (
        IdEmpleado INTEGER CONSTRAINT IndicePrimario PRIMARY,

```

3. Bases de datos y tablas

```
    Nombre TEXT,  
    Apellidos TEXT,  
    FechaNacimiento DATETIME  
)
```

Asimismo, la sentencia anterior crea una tabla llamada **Empleados** con un campo texto de longitud predeterminada 50 llamado **Nombre** y otro igual denominado **Apellidos**; crea otro campo llamado **FechaNacimiento** de tipo Fecha/Hora y el campo **IdEmpleado** de tipo entero al que establece como clave principal.

A continuación se muestra un ejemplo más completo:

```
CREATE TABLE mitabla ( id INT PRIMARY KEY, nombre  
VARCHAR(20) );  
INSERT INTO mitabla VALUES ( 1, 'Will' );  
INSERT INTO mitabla VALUES ( 2, 'Marry' );  
INSERT INTO mitabla VALUES ( 3, 'Dean' );  
SELECT id, nombre FROM mitabla WHERE id = 1;  
UPDATE mitabla SET nombre = 'Willy' WHERE id = 1;  
SELECT id, nombre FROM mitabla;  
DELETE FROM mitabla WHERE id = 1;  
SELECT id, nombre FROM mitabla;
```

Las sentencias anteriores no solo se encargan de crear una tabla con sus respectivos valores, sino que también realizan la tarea de mostrar los valores más adelante.

No te preocupes si no lo entiendes por completo pues, en el próximo capítulo de este e-book, aprenderás a realizar consultas sobre los datos de una tabla ([Figura 3.3.](#)).

Online SQL IDE

```

1: CREATE TABLE mitabla ( id INT PRIMARY KEY, nombre VARCHAR(20) );
2: INSERT INTO mitabla VALUES ( 1, "Will" );
3: INSERT INTO mitabla VALUES ( 2, "Marry" );
4: INSERT INTO mitabla VALUES ( 3, "Dean" );
5: SELECT id, nombre FROM mitabla WHERE id = 1;
6: UPDATE mitabla SET nombre = "Willy" WHERE id = 1;
7: SELECT id, nombre FROM mitabla;
8: DELETE FROM mitabla WHERE id = 1;
9: SELECT id, nombre FROM mitabla;

```

Execute Mode: Version, Inputs & Arguments

SQLite 3.37.0 Interactive

Execute

Result
executed in 0.028 secs)

```

1|Will
1|Willy
2|Marry
3|Dean
2|Marry
3|Dean

```

Notes:
 1. For file operations - upload files using upload button . Files will be upload to /uploads folder. You can read those files in program from /uploads folder. To write a file from your program, write files to /myfiles/ folder.
 Please note the uploaded files stored in the server only for the current session.
 2. For detailed documentation check - Our Documentation, or check our YouTube channel.

Figura 3.3. Creación de una tabla y posterior consulta de los datos almacenados.

CONSTRAINT

Se utiliza la cláusula **CONSTRAINT** en las instrucciones **ALTER TABLE** y **CREATE TABLE** para crear o eliminar índices. Existen dos sintaxis para esta cláusula dependiendo de la necesidad de crear o eliminar un índice de un único campo o si se trata de un campo multiíndice. Si se utiliza el motor de datos de Microsoft, solo se podrá usar esta cláusula con las bases de datos propias de dicho motor. Para los índices de campos únicos:

```

CONSTRAINT nombre {PRIMARY KEY | UNIQUE | REFERENCES tabla
externa
[(campo externo1, campo externo2)]}

```

Para los índices de campos múltiples:

```

CONSTRAINT nombre {PRIMARY KEY (primario1[, primario2
[,...]]) | }

```

3. Bases de datos y tablas

```
UNIQUE (único1[, único2 [, ...]]) |  
FOREIGN KEY (ref1[, ref2 [,...]]) REFERENCES tabla externa  
[(campo externo1 ,campo externo2 [,...])]
```

- **nombre**: es el nombre del índice que se va a crear.
- **primarioN**: es el nombre del campo o de los campos que forman el índice primario.
- **únicoN**: es el nombre del campo o de los campos que forman el índice de clave única.
- **refN**: es el nombre del campo o de los campos que forman el índice externo (hacen referencia a campos de otra tabla).
- **tabla externa**: es el nombre de la tabla que contiene el campo o los campos referenciados en **refN**.
- **campos externos**: es el nombre del campo o de los campos de la tabla externa especificados por **ref1, ref2, refN**.

Si se desea crear un índice para un campo cuando se utilizan las instrucciones **ALTER TABLE** o **CREATE TABLE**, la cláusula **CONSTRAINT** debe aparecer inmediatamente después de la especificación del campo indexado.

Si necesitas crear un índice con múltiples campos cuando se están utilizando las instrucciones **ALTER TABLE** o **CREATE TABLE**, la cláusula **CONSTRAINT** debe aparecer fuera de la cláusula de creación de tabla.

- **UNIQUE**: genera un índice de clave única. Lo que implica que los registros de la tabla no pueden tener el mismo valor en los campos indexados.
- **PRIMARY KEY**: genera un índice primario del campo o los campos especificados. Todos los campos de la clave principal deben ser únicos y no nulos, y cada tabla solo puede tener una única clave principal.

- **FOREIGN KEY:** genera un índice externo (toma como valor del índice campos contenidos en otras tablas). Si la clave principal de la tabla externa consta de más de un campo, se debe utilizar una definición de índice de múltiples campos, listando todos los campos de referencia, el nombre de la tabla externa y los nombres de los campos referenciados en la tabla externa en el mismo orden que los campos de referencia listados. Si los campos referenciados son la clave principal de la tabla externa, no se deben especificar estos campos, ya que, predeterminado por valor, el motor Jet se comporta como si la clave principal de la tabla externa estuviera formada por los campos referenciados.

Modificar una tabla



La sentencia **ALTER TABLE** permite modificar una tabla que hayas creado antes. Para modificar una tabla deberás mencionar, como mínimo, el nombre de la tabla, el tipo de modificación por ejecutar y el campo afectado.

```
ALTER TABLE <nombre_de_tabla> <acción> <campo> <tipo_campo>;
```

Las acciones posibles son **modify** y **add**. Ciertas bases de datos, no todas, permiten borrar campos de una tabla.

Es posible modificar los campos o los índices existentes en una tabla. La sintaxis completa es la siguiente:

```
ALTER TABLE tabla {ADD {COLUMN tipo de campo[(tamaño)]  
[CONSTRAINT índice]}
```

3. Bases de datos y tablas

```
CONSTRAINT índice multicampo} |  
DROP {COLUMN campo I CONSTRAINT nombre del índice}}
```

- **tabla**: es el nombre de la tabla que se desea modificar.
- **campo**: es el nombre del campo que se va a añadir o eliminar.
- **tipo**: es el tipo de campo que se va a añadir.
- **tamaño**: es el tamaño del campo que se añadirá.
- **índice**: es el nombre del índice del campo (cuando se crean campos) o el nombre del índice de la tabla que se desea eliminar.
- **índice multicampo**: es el nombre del índice del campo multicampo o el nombre del índice de la tabla que se desea eliminar.

Las operaciones posibles son las siguientes:

- **ADD COLUMN**: se utiliza para añadir un nuevo campo a la tabla, indicando el nombre, el tipo de campo y, opcionalmente, el tamaño para campos de tipo texto.
- **ADD**: se utiliza para agregar un índice de multicampos o de un único campo.
- **DROP COLUMN**: se utiliza para borrar un campo. Se especifica únicamente el nombre del campo.
- **DROP**: se utiliza para eliminar un índice. Se especifica solo el nombre del índice a continuación de la palabra reservada **CONSTRAINT**.

Eliminar una tabla



La sentencia **DROP TABLE** permite borrar una tabla existente. Para borrar una tabla hay que mencionar su nombre.

```
DROP TABLE <nombre_de_tabla>;
```

DROP TABLE elimina una tabla de una base de datos, pero debes considerar que solo el propietario de la tabla, el propietario del esquema, un superusuario, un usuario o un grupo que tengan el privilegio **DROP** pueden realizar esta acción.

Si necesitas borrar las filas de una tabla sin eliminar la tabla, deberás utilizar el comando **DELETE** o **TRUNCATE**. Es posible eliminar varias tablas con un solo comando **DROP TABLE**.

La sintaxis completa de la sentencia es la siguiente:

```
DROP TABLE [ IF EXISTS ] name [, ...] [ CASCADE | RESTRICT ]
```

Estos son sus parámetros:

- **IF EXISTS**: cláusula que indica que, si la tabla especificada no existe, el comando no debe realizar cambios y devolverá un mensaje en el que se indique que la tabla no existe, en lugar de terminar con un error. Esta cláusula es útil cuando se realiza scripting, para que el script no produzca un error si **DROP TABLE** se ejecuta contra una tabla que no existe.
- **name**: nombre de la tabla que se eliminará.
- **CASCADE**: cláusula que indica que se deben eliminar automáticamente los objetos que dependen de la vista, como otras vistas.
- **RESTRICT**: cláusula que indica que no se debe eliminar la tabla si tiene objetos dependientes. Esta acción es la predeterminada.

A continuación se muestran algunos ejemplos.

3. Bases de datos y tablas

Para eliminar una tabla sin dependencias:

```
create table feedback(a int);

drop table feedback;
```

Para eliminar dos tablas en forma simultánea:

```
create table feedback(a int);

create table buyers(a int);

drop table feedback, buyers;
```

Para eliminar una tabla con una dependencia:

```
create table feedback(a int);

create view feedback_view as select * from feedback;

drop table feedback cascade;
```

Si necesitas ver las dependencias de una tabla, primero es necesario crearlas:

```
create view find_depend as
select distinct c_p.oid as tbloid,
```

```
n_p.nspname as schemaname, c_p.relname as name,  
n_c.nspname as refbyschemaname, c_c.relname as refbyname,  
c_c.oid as viewoid  
from pg_catalog.pg_class c_p  
join pg_catalog.pg_depend d_p  
on c_p.relfilenode = d_p.refobjid  
join pg_catalog.pg_depend d_c  
on d_p.objid = d_c.objid  
join pg_catalog.pg_class c_c  
on d_c.refobjid = c_c.relfilenode  
left outer join pg_namespace n_p  
on c_p.relnamespace = n_p.oid  
left outer join pg_namespace n_c  
on c_c.relnamespace = n_c.oid  
where d_c.deptype = 'i'::"char"  
and c_c.relkind = 'v'::"char";
```

A continuación:

```
create view sales_view as select * from sales;
```

Y luego realiza la consulta:

```
select * from find_depend  
where refbyschemaname='public'  
order by name;
```

El resultado será algo similar a lo siguiente:

3. Bases de datos y tablas

```
tbloid | schemaname |      name       | viewoid |
refbyschemaname |  refbyname
-----+-----+-----+-----+
-----+
100241 | public      | find_depend | 100241 | public
| find_depend
100203 | public      | sales        | 100245 | public
| sales_view
100245 | public      | sales_view   | 100245 | public
| sales_view
(3 rows)
```

Por último, para eliminar una tabla solo en el caso de que exista:

```
drop table if exists feedback;
```

Actividades

A continuación verás las preguntas que deberías saber responder para considerar aprendido el capítulo.

Test de autoevaluación

1. *¿Para qué sirve **CREATE SCHEMA**?*
 2. *¿Cómo debes utilizar **CREATE DATABASE**?*
 3. *Indica los pasos que debes seguir para crear una tabla y, posteriormente, eliminarla.*
 4. *Menciona lo más importante de la estructura de una tabla.*
 5. *¿Qué sentencias puedes utilizar para modificar una tabla?*
 6. *¿Cómo puedes eliminar una tabla?*
-

Capítulo 04



Insertar y modificar datos

Has llegado al corazón de este e-book y, en este capítulo, revisarás las formas más importantes que ofrece SQL para trabajar con los datos existentes en una tabla, es decir, insertar, eliminar, modificar y también consultar información.

Al terminar este capítulo, podrás realizar las siguientes tareas:

- ✿ Insertar y eliminar datos de una tabla.
- ✿ Realizar consultas mediante SELECT.
- ✿ Utilizar diferentes funciones en conjunto a SELECT.

Insertar datos / 73

Modificar datos / 74

Eliminar datos / 75

 Commit y rollback / 76

Visualizar datos / 77

 WHERE / 84

 COUNT / 86

 Sum, Avg, Min, Max / 87

 DISTINCT / 87

 ORDER BY / 88

 UNION / 89

Actividades / 91

 Test de autoevaluación / 91

Insertar datos



La sentencia **INSERT** permite insertar datos en una tabla.

```
INSERT INTO <nombre_de_tabla> (<campo_1>,<campo_2>,<...>)
VALUES
(<valor_campo_1>,<valor_campo_2>,<valor_...>);
```

También existe:

```
INSERT INTO <nombre_de_tabla> (<campo_1>,<campo_2>,<...>)
<SELECT STATEMENT>;
```

La instrucción **INSERT** permite crear o insertar nuevos registros en una tabla, esta es su sintaxis:

```
insert into ALUMNOS (ID_ALUMNO , NOMBRE , APELLIDOS , F_
NACIMIENTO)
values (1 , 'Pablo' , 'Hernandez Mata' , '1995-03-14')
```

Su sintaxis general es la siguiente:

```
INSERT INTO nombre_tabla (lista de campos separados por
comas)
VALUES (lista de datos separados por comas)
```

Cada dato de la lista **VALUES** se corresponde y se asigna a cada campo de

4. Insertar y modificar datos

la tabla en el mismo orden de aparición de la sentencia **INSERT**. Si la clave primaria que identifica el registro que se pretende insertar ya la usa un registro existente, el SGBD rechazará la operación y devolverá un error de clave primaria duplicada.

Modificar datos



La sentencia **UPDATE** permite modificar el valor de uno o de varios datos en una tabla.

```
UPDATE <nombre_de_tabla> SET <campo_1>=<valor_campo_1>,<campo_2>=<valor_campo_2>,<...>;
```

Por lo general, se limita el cambio a ciertos registros mencionados, utilizando una cláusula **WHERE**.

```
UPDATE <nombre_de_tabla> SET <campo_1>=<valor_campo_1>,<campo_2>=<valor_campo_2>,<...>
WHERE <cláusula_where>;
```

La instrucción **UPDATE** permite actualizar registros de una tabla. Se debe indicar qué registros se quieren actualizar mediante la cláusula **WHERE**, y qué campos, mediante la cláusula **SET**. Además, se deberá indicar qué nuevo dato va a guardar cada campo.

Ahora se muestra un ejemplo:

```
update CURSOS
```

```
set ID_PROFE = 2  
where ID_CURSO = 5
```

La instrucción anterior asigna un **2** en el campo **ID_PROFE** de la tabla **CURSOS** en los registros cuyo valor en el campo **ID_CURSO** es **5**. El campo **ID_CURSO** es la clave primaria de la tabla, tan solo se modificará un registro si es que existe. Dado que el campo que se pretende actualizar es clave foránea de la tabla **PROFESORES**, si no existe un registro en dicha tabla con identificador **2**, el SGBD devolverá un error de clave no encontrada.

Eliminar datos



La sentencia **DELETE** permite borrar uno o varios registros en una tabla.

```
DELETE FROM <nombre_de_tabla> ;
```

Por lo general, se limita el borrado a ciertos registros mencionados, utilizando una cláusula **WHERE**.

```
DELETE FROM <nombre_de_tabla> WHERE <cláusula_where>;
```

La instrucción **DELETE** posee una sintaxis simple, puesto que solo debes indicar qué registros deseas eliminar mediante la cláusula **WHERE**. La siguiente consulta elimina todos los registros de la tabla **mascotas** que están de baja:

```
delete from MASCOTAS
```

4. Insertar y modificar datos

```
where ESTADO = 'B'
```

Por ejemplo, la siguiente instrucción eliminará todos los registros de la tabla **VEHICULOS**:

```
delete  
from VEHICULOS
```

Commit y rollback

Si la base de datos permite la gestión de transacciones, se puede utilizar **commit** para confirmar **insert** y **update**, o **delete** y **rollback** para cancelarlos. Ciertas base de datos pueden ser configuradas para autocommit, que hace **commit** en forma automática después de cada instrucción, a menos que se haya iniciado una transacción de manera explícita. Mientras **commit** no está ejecutada, las modificaciones no están inscritas de manera permanente en la base de datos y solo son visibles para la sesión en curso del usuario autor de las acciones. Después de **commit**, los cambios son definitivos y visibles para todos.

Por ejemplo:

```
SELECT emp_no,job_grade FROM employee where emp_no=45;  
START TRANSACTION;  
update employee set job_grade=5 where emp_no=45;  
SELECT emp_no,job_grade FROM employee where emp_no=45;  
rollback;  
SELECT emp_no,job_grade FROM employee where emp_no=45;
```

```
START TRANSACTION;  
update employee set job_grade=5 where emp_no=45;  
SELECT emp_no,job_grade FROM employee where emp_no=45;  
commit;  
SELECT emp_no,job_grade FROM employee where emp_no=45;
```

Visualizar datos



SELECT permite seleccionar datos en la base de datos y visualizarlos. Es posible utilizar un alias para que el campo se pueda llamar con otro nombre.

```
SELECT <campo_1>,<campo_2>,<...> FROM <nombre_tabla>;  
SELECT <campo_1> as <alias1>,<campo_2>,<...> FROM <nombre_tabla>;
```

Para seleccionar todos los campos de la tabla, se utiliza el asterisco (*) en vez de los nombres de campo.

```
SELECT * FROM <nombre_tabla>;
```

Por ejemplo:

```
SELECT emp_no,job_grade as nivel FROM employee;  
SELECT * FROM employee;
```

4. Insertar y modificar datos

La selección de datos es una de las partes más importantes del trabajo con SQL, para ello utiliza **SELECT**. La selección total o parcial de una tabla se realiza mediante la instrucción **SELECT**, en la que es necesario especificar lo siguiente:

- los campos que deseas seleccionar;
- la tabla en la que realizarás la selección.

Por ejemplo:

```
Select nombre, dirección From clientes
```

Si necesitas seleccionar todos los campos, es decir, toda la tabla, debes utilizar el comodín ***** del siguiente modo:

```
Select * From clientes
```

También resulta muy útil filtrar los registros mediante condiciones que vienen expresadas después de la cláusula **WHERE**. Si necesitas mostrar los clientes de una determinada ciudad, usa la siguiente expresión:

```
Select * From clientes Where poblacion Like 'Madrid'
```

También podrías ordenar los resultados en función de uno o varios de sus campos. En este último ejemplo, es posible ordenarlos por nombre:

```
Select * From clientes Where poblacion Like <Madrid> Order By nombre
```

Puede haber más de un cliente con el mismo nombre, podrías dar un segundo criterio como ser el apellido:

```
Select * From clientes Where poblacion Like 'Madrid' Order By nombre, apellido
```

Si inviertes el orden **nombre, apellido** por **apellido, nombre**, el resultado será distinto. Verás los clientes ordenados por apellido, y aquellos que tuviesen apellidos idénticos aparecerán subclasificados por el nombre. Si necesitas ver los clientes por orden de pedidos realizados con los mayores en primer lugar, escribe lo siguiente:

```
Select * From clientes Order By pedidos Desc
```

Por otra parte, si buscas saber en qué ciudades se encuentran tus clientes sin necesidad de que para ello aparezca varias veces la misma ciudad, usa una sentencia de esta clase:

```
Select Distinct poblacion From clientes Order By poblacion
```

La sintaxis completa de la instrucción **SELECT** es compleja; a continuación se muestra un resumen:

```
SELECT nombres de las columnas  
[INTO nueva Tabla destino para resultados del select_]  
FROM origenTabla  
[WHERE condición de Búsqueda]
```

4. Insertar y modificar datos

```
[GROUP BY nombres de columnas por la cual Agrupar] [HAVING  
condiciónBúsqueda para Group By ]  
[ORDER BY nombre de columnas [ASC | DESC] ]
```

Asimismo, se pueden unir estas sentencias con otras por el operador **UNION** entre consultas, para combinar sus resultados en una sola tabla de resultados sin nombre.

Ahora verás las funciones de cada una de las cláusulas de la sentencia **SELECT**:

- La cláusula **SELECT**: se usa para listar las columnas de las tablas que se desean ver en el resultado de una consulta. Además de las columnas, se pueden listar columnas para calcular por el SQL cuando actúe la sentencia. Esta cláusula no puede omitirse.
- La cláusula **FROM**: lista las tablas que deben ser analizadas en la evaluación de la expresión de la cláusula **WHERE** y de dónde se listarán las columnas enunciadas en **SELECT**. Esta cláusula no puede omitirse.
- La cláusula **WHERE**: establece criterios de selección de ciertas filas en el resultado de la consulta gracias a las condiciones de búsqueda. Si no se requieren condiciones de búsqueda, puede omitirse, y el resultado de la consulta serán todas las filas de las tablas enunciadas en **FROM**.
- La cláusula **GROUP BY**: especifica una consulta sumaria. En vez de producir una fila de resultados por cada fila de datos de la base de datos, una consulta sumaria agrupa todas las filas similares y, luego, produce una fila sumaria de resultados para cada grupo de los nombres de columnas enunciados en esta cláusula. En otras palabras, esta cláusula permitirá agrupar un conjunto de columnas con valores repetidos y utilizar las funciones de agregación sobre las columnas con valores no repetidos. Esta cláusula puede omitirse.

- La cláusula **HAVING**: le dice al SQL que incluya en los resultados de la consulta solo ciertos grupos producidos por la cláusula **GROUP BY**. Al igual que la cláusula **WHERE**, utiliza una condición de búsqueda para especificar los grupos deseados. La cláusula **HAVING** es la encargada de condicionar la selección de los grupos sobre la base de los valores resultantes en las funciones agregadas que se utilizan, debido a que la cláusula **WHERE** condiciona solo para la selección de filas individuales. Esta cláusula puede omitirse.
- La cláusula **ORDER BY**: permitirá establecer la columna o las columnas sobre las cuales las filas resultantes de la consulta deberán ser ordenadas.

Al utilizar la sentencia **SELECT** con las cláusulas **SELECT - FROM**, verás como resultado todas las filas existentes en las tablas especificadas en **FROM**.

Ejemplo: seleccionar todas las columnas y filas de la tabla **CLIENTE**:

```
SELECT * FROM CLIENTE
```

Obtendrás como resultado un total de ocho filas con las siete columnas que posee la tabla **CLIENTE**.

Ejemplo: seleccionar columnas **nomcli** y **e_mail** de la tabla **CLIENTE**:

```
SELECT NUMCLI, NOMCLI, E-MAIL FROM CLIENTE
```

Verás como resultado un total de ocho filas con solo tres columnas.

Ejemplo: seleccionar las columnas y mostrarlas con el título especificado en **AS**, es decir, con un alias. A **NUMCLI** lo muestra como **NUMERO DE CLIENTE**

4. Insertar y modificar datos

y a **NOMCLI** lo muestra con el nombre especificado en el **AS** como **NOMBRE DEL CLIENTE**. Esto permite ver una columna con encabezados más familiares a los usuarios finales.

```
SELECT NUMCLI AS <NUMERO DE CLIENTE>, NOMCLI AS <NOMBRE DE CLIENTE> FROM CLIENTE
```

Verás como resultado un total de ocho filas.

Es importante mencionar que la cláusula **AS** puede omitirse, y el resultado será el mismo.

```
SELECT NUMCLI ‘NUMERO DE CLIENTE’, NOMCLI ‘NOMBRE DE CLIENTE’ FROM CLIENTE
```

A continuación, se muestra un caso práctico para abordar el uso de **SELECT**. Piensa en una tabla de empleados con la que necesites saber quiénes tienen un salario mayor a 1350.

ID_EMPLEADO	NOMBRE	APPELLIDOS	F_NACIMIENTO	SEXO	CARGO	SALARIO
1	Carlos	Jiménez Clarín	1985-05-03	H	Mozo	1500
2	Elena	Rubio Cuestas	1978-09-25	M	Secretaria	1300
3	José	Calvo Sisman	1990-11-12	H	Mozo	1400
4	Margarita	Rodríguez Garcés	1992-05-16	M	Secretaria	1325.5

SQL permite responder a estas preguntas de forma rápida y fiable, salvo si hubo error al construir la consulta o errores en los propios datos.

Para construir una consulta SQL, debes hacer como mínimo tres preguntas:

- ¿Qué datos se piden? En este caso, el nombre y los apellidos de los empleados.
- ¿Dónde están esos datos? Obviamente están en la tabla **Empleados**.
- ¿Qué requisitos deben cumplir los registros? En este caso, que el sueldo del empleado sea superior a 1350.

Cómo formalizar la consulta:

Seleccióname el NOMBRE y los APELLIDOS
del archivo EMPLEADOS
cuyo SALARIO sea mayor a 1350

En SQL la forma de operar es similar, esta información se obtiene mediante la siguiente consulta:

```
select NOMBRE , APELLIDOS  
      from EMPLEADOS  
     where SALARIO > 1350
```

En la consulta, los nombres de los objetos de base de datos (tabla y campos) se escriben en mayúsculas, mientras que las palabras reservadas de la consulta SQL (**select**, **from**, **where**) se hace en minúsculas; esto tiene únicamente un propósito estético, con intención de hacer el código más ordenado y legible.

4. Insertar y modificar datos

El resultado que devuelve el SGBD es:

NOMBRE	APELLIDOS
Carlos	Jiménez Clarín
José	Calvo Sisman

En general una consulta SQL simple tendrá la siguiente forma:

```
select CAMPOS(separados por comas)
      from TABLA
     where CONDICION
```

SQL permite al usuario desentenderse de cómo el SGBD ejecuta la consulta. Dicho de otro modo, basta con saber cómo pedir la información y no cómo esta se reunirá.

WHERE

La cláusula **WHERE** permite limitar la encuesta a ciertos datos.

Se utiliza evaluando un campo versus una condición. Se pueden utilizar varias condiciones con el uso de **or**, **and** o paréntesis.

Para comparar números, se utilizan los signos **=**, **<**, **>**, **<=**, **>=**, **between ... and ...**

Para comparar caracteres, se utiliza la palabra **like**. El wildcard es **%**.

Para comparar fechas, se utilizan los signos **=**, **<**, **>**, **<=**, **>=**, **between ... and ...**

```
SELECT * FROM <nOMBRE_tabla>
WHERE <campo_1> <operation> <condición> AND <campo_2>
```

```
<operation> <condición>;
```

Ejemplo:

```
SELECT emp_no, job_grade FROM employee where emp_no>45;
SELECT emp_no, job_grade FROM employee where emp_no=45 or
emp_no=41;
SELECT * FROM employee where emp_no between 40 and 45;
SELECT * FROM employee where last_name like 'P%';
```

Este es un ejemplo común. ¿Qué empleados tienen un sueldo comprendido entre 1350 y 1450?

ID_EMPLEADO	NOMBRE	APPELLIDOS	F_NACIMIENTO	SEXO	CARGO	SALARIO
1	Carlos	Jiménez Clarín	1985-05-03	H	Mozo	1500
2	Elena	Rubio Cuestas	1978-09-25	M	Secretaria	1300
3	José	Calvo Sisman	1990-11-12	H	Mozo	1400
4	Margarita	Rodríguez Garcés	1992-05-16	M	Secretaria	1325.5

Las preguntas iniciales son:

- ¿Qué datos se piden?
- ¿Dónde están los datos?
- ¿Qué requisitos deben cumplir los registros?

4. Insertar y modificar datos

Las cláusulas **SELECT** y **FROM** no van a cambiar respecto al caso expuesto en la sección anterior, pero sí lo hará la cláusula **WHERE**.

```
select NOMBRE , APELLIDOS  
      from EMPLEADOS  
     where SALARIO > 1350
```

El salario debe estar comprendido entre 1350 y 1450, ambos inclusive. La cláusula **WHERE** queda de la siguiente forma:

```
where SALARIO >= 1350 and SALARIO <= 1450
```

donde el salario sea mayor o igual a 1350 y menor o igual a 1450.

La consulta quedaría de la siguiente forma:

```
select NOMBRE , APELLIDOS  
      from EMPLEADOS  
     where SALARIO >= 1350 and SALARIO <= 1450
```

COUNT

Para contar un número de registros, se utiliza la palabra **COUNT**.

```
SELECT COUNT(<campo_1>) FROM <nombre_tabla>;
```

A continuación puedes ver un ejemplo:

```
SELECT count(*) FROM employee where job_grade=4;
```

Sum, Min, Max

Para una suma, mínimo, máximo de un campo, se utilizan las palabras **Sum**, **Min**, **Max**.

```
SELECT SUM(<campo_1>) FROM <nombre_tabla>;
```

Ejemplo:

```
SELECT avg(salary) FROM employee where job_grade=2;
```

DISTINCT

Para tener la lista de valores distintas de un campo, se utiliza la palabra **DISTINCT**.

```
SELECT DISTINCT(<campo_1>) FROM <nombre_tabla>;
```

Ejemplo:

```
SELECT distinct(job_grade) FROM employee;
```

Omite los registros que contienen datos duplicados en los campos seleccionados.

4. Insertar y modificar datos

Para que los valores de cada campo listado en la instrucción **SELECT** se incluyan en la consulta, deben ser únicos.

Por ejemplo, varios empleados listados en la tabla **Empleados** pueden tener el mismo apellido. Si dos registros contienen López en el campo **Apellido**, la siguiente instrucción SQL devuelve un único registro:

```
SELECT DISTINCT Apellido FROM Empleados;
```

Con otras palabras, el predicado **DISTINCT** devuelve aquellos registros cuyos campos indicados en la cláusula **SELECT** posean un contenido diferente. El resultado de una consulta que utiliza **DISTINCT** no es actualizable y no refleja los cambios subsiguientes realizados por otros usuarios.

ORDER BY

Para ordenar los registros regresados, hay que utilizar la frase **ORDER BY**.

```
SELECT * FROM <nombre_tabla>
ORDER BY <campo_1>,<....>;
```

Ejemplo:

```
SELECT first_name,last_name FROM employee order by first_
name,last_name;
```

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **ORDER BY Lista de campos**.

En donde **Lista de campos** representa los campos por ordenar. Ejemplo:

```
SELECTCodigoPostal, Nombre, Telefono FROM Clientes ORDER  
BY Nombre;
```

Esta consulta devuelve los campos **CodigoPostal**, **Nombre**, **Telefono** de la tabla **Clientes** ordenados por el campo **Nombre**.

Se pueden ordenar los registros por más de un campo, por ejemplo:

```
SELECTCodigoPostal, Nombre, Telefono FROM Clientes ORDER  
BY  
CodigoPostal, Nombre;
```

Incluso, se puede especificar el orden de los registros: ascendente, mediante la cláusula **ASC** (se toma este valor por defecto), o descendente, usando **DESC**.

```
SELECTCodigoPostal, Nombre, Telefono FROM Clientes ORDER  
BY  
CodigoPostal DESC , Nombre ASC;
```

UNION

UNION permite unir los resultados de dos consultas. Para poder unirlas, tienen que tener los mismos campos.

```
SELECT <campo_1>,<campo_2>,<...> FROM <nombre_tabla_1>
```

4. Insertar y modificar datos

UNION

```
SELECT <campo_1>,<campo_2>,<...> FROM <nombre_tabla_2>;
```

Ejemplo:

```
select t.first_name,t.last_name from employee t where job_
grade=5
union
select t2.fname,t2.lname from usuario t2;
```

Actividades

A continuación verás las preguntas que deberías saber responder para considerar aprendido el capítulo.

Test de autoevaluación

1. *Indica de qué forma se pueden insertar y eliminar datos de una tabla.*
 2. *¿Para qué sirve **SELECT**?*
 3. *Menciona algunos casos de uso para **SELECT**.*
 4. *¿Cómo es posible ordenar los datos extraídos en una consulta?*
 5. *¿Para qué sirve **WHERE**?*
 6. *¿Cómo puedes utilizar **UNION**?*
-

Capítulo 05



Subconsultas y uniones

En el último capítulo de este e-book, revisarás la sintaxis y algunos ejemplos de consultas más avanzados que podrás realizar sobre una base de datos: las subconsultas y las uniones.

Al terminar este capítulo, podrás realizar las siguientes tareas:

- ✿ *Conocer la sintaxis de una subconsulta.*
- ✿ *Conocer la forma en que se realizan las uniones.*
- ✿ *Identificar en qué casos se pueden aplicar las subconsultas y las uniones.*

Subconsultas / 93

Agrupaciones y combinaciones / 99

Autocombinación / 102

Actividades / 104

Test de autoevaluación / 104

Glosario / 105

Subconsultas



Las subconsultas son consultas sobre otras consultas. La subconsulta se puede utilizar en la cláusula **FROM** o en la condición de la cláusula **WHERE**, y se pone entre paréntesis. En MySQL, las subconsultas deben tener sus propios alias. Esta es su sintaxis:

```
SELECT t3.<campo_1>, t3.<campo_2> FROM (SELECT  
t.<campo_1>, t.<campo_2> FROM <nombre_tabla> t <where  
cluse>) t3  
WHERE t3.<campo_1> IN (SELECT t2.<campo_1> FROM <nombre_>  
tabla_2> t2);
```

Ejemplo:

```
SELECT t3.first_name,t3.last_name FROM  
(  
select t.first_name,t.last_name from employee t where job_>  
grade=5  
union  
select t2.fname,t2.lname from usuario t2  
) t3 where t3.last_name like '%o%';  
SELECT t3.first_name,t3.last_name FROM employee t3  
where t3.job_country IN  
(select t.country from country t where t.currency='Euro');
```

Las subconsultas son, en esencia, una instrucción **SELECT** anidada dentro de **SELECT**, **SELECT...INTO**, **INSERT...INTO**, **DELETE**, **UPDATE** o también dentro de otra subconsulta:

5. Subconsultas y uniones

comparación [ANY | ALL | SOME] (instrucción sql)
expresión [NOT] IN (instrucción sql) [NOT] EXISTS
(instrucción sql)

Donde:

- **comparación**: es una expresión y un operador de comparación que compara la expresión con el resultado de la subconsulta.
- **expresión**: es una expresión por la que se busca el conjunto resultante de la subconsulta.
- **instrucción SQL**: es una instrucción **SELECT**, que sigue el mismo formato y las reglas que cualquier otra instrucción **SELECT**. Como ya sabes, debe ir entre paréntesis.

Se puede utilizar una subconsulta en lugar de una expresión en la lista de campos de una instrucción **SELECT**, o en una cláusula **WHERE** o **HAVING**.

Debes tener en cuenta que, en una subconsulta, se utiliza una instrucción **SELECT** para proporcionar un conjunto de valores especificados para evaluar en la expresión de la cláusula **WHERE** o **HAVING**.

Ahora verás un ejemplo común. Imagina que se devuelven todos los productos cuyo precio es mayor que el de cualquier producto vendido con un descuento igual o mayor al 25%:

```
SELECT *
FROM
    Productos
WHERE
    PrecioUnidad
    ANY
```

```
(  
SELECT  
PrecioUnidad  
FROM  
DetallePedido  
WHERE  
Descuento = 0 .25  
)
```

ALL se usa para recuperar aquellos registros de la consulta principal que satisfacen la comparación con todos los registros recuperados en la subconsulta. Si cambias **ANY** por **ALL**, la consulta devolverá los productos cuyo precio unitario sea mayor que el de todos los productos vendidos con un descuento igual o mayor al 25%.

IN se utiliza para recuperar aquellos registros de la consulta principal para los que algunos registros de la subconsulta contienen un valor igual. El código a continuación mostrará los productos vendidos con un descuento igual o mayor al 25%:

```
SELECT *  
FROM  
    Productos  
WHERE  
    IDProducto  
        IN  
(  
    SELECT  
    IDProducto  
    FROM
```

5. Subconsultas y uniones

```
DetallePedido  
WHERE  
Descuento = 0.25  
)
```

EXISTS (con **NOT** opcional) se usa en comparaciones de verdad/falso para determinar si la subconsulta devuelve algún registro. Por ejemplo, para recuperar los clientes que hayan realizado al menos un pedido:

```
SELECT  
Clientes.Compañía, Clientes.Teléfono  
FROM  
    Clientes  
WHERE EXISTS (  
    SELECT  
        FROM  
        Pedidos  
        WHERE  
        Pedidos.IdPedido = Clientes.IdCliente  
)
```

La consulta anterior es equivalente a la siguiente:

```
SELECT  
    Clientes.Compañía, Clientes.Teléfono  
FROM  
    Clientes  
WHERE
```

```
IdClientes
IN
(
SELECT
Pedidos.IdCliente
FROM
Pedidos
)
```

La consulta siguiente devuelve los nombres de los empleados cuyo salario es igual o mayor que el salario medio de todos los empleados con el mismo título.

```
SELECT
    Apellido, Nombre, Titulo, Salario
FROM
    Empleados AS T1
WHERE
    Salario =
(
SELECT
    Avg(Salario)
FROM
    Empleados
WHERE
    T1.Titulo = Empleados.Titulo
)
ORDER BY Titulo
```

5. Subconsultas y uniones

A continuación se ofrecen otros ejemplos. Para obtener una lista con el nombre, cargo y salario de todos los agentes de ventas cuyo salario es mayor que el de todos los jefes y directores:

```
SELECT
    Apellidos, Nombre, Cargo, Salario
FROM
    Empleados
WHERE
    Cargo LIKE 'Agente Ven*'
    AND
    Salario ALL
    (
        SELECT
            Salario
        FROM
            Empleados
        WHERE
            Cargo LIKE '*Jefe*'
        OR
            Cargo LIKE '*Director*'
    )
```

Para obtener una lista de las compañías y los contactos de todos los clientes que han realizado un pedido en el segundo trimestre de 2010:

```
SELECT DISTINCT
    NombreContacto, NombreCompania, CargoContacto,
    Telefono
```

```
FROM
    Clientes
WHERE
    IdCliente IN (
        SELECT DISTINCT IdCliente
        FROM Pedidos
        WHERE FechaPedido <#07/01/2010#
    )
```

Para elegir el nombre de todos los empleados que han reservado al menos un pedido:

```
SELECT
    Nombre, Apellidos
FROM
    Empleados AS E
WHERE EXISTS
(
    SELECT *
    FROM
        Pedidos AS O
    WHERE O.IdEmpleado = E.IdEmpleado
)
```

Agrupaciones y combinaciones

Las **agrupaciones** permiten reunir datos y saber cuántos existen de cada valor. Se pueden filtrar utilizando la cláusula **HAVING**. Su sintaxis es la siguiente:

5. Subconsultas y uniones

```
SELECT <campo_1>, <campo_2>, COUNT(*) FROM <nombre_tabla>
GROUP BY <campo_1>, <campo_2>;
```

Ejemplo:

```
SELECT job_grade, count(*) FROM employee
where emp_no>45
group by job_grade;
SELECT job_grade, sum(salary) FROM employee
where emp_no>45
group by job_grade
having sum(salary)<1000000;
```

Las consultas de combinación entre tablas se realizan mediante la cláusula **INNER** que se encarga de combinar registros de dos tablas, pero siempre que exista concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON
tb1.campo1 comparador tb2.campo2
```

Donde:

- **tb1, tb2**: son los nombres de las tablas desde las que se combinan los registros.
- **campo1, campo2**: son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.
- **comparador**: es cualquier operador de comparación relacional: **=, <, <>, <=, =>, >**.

Se puede utilizar una operación **INNER JOIN** en cualquier cláusula **FROM**. Esto crea una combinación por equivalencia, conocida también como **unión interna**. Las **combinaciones equivalentes** son las más comunes; estas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas.

A continuación, verás cómo podrías combinar las tablas **Categorías** y **Productos** basándote en el campo **IDCategoria**. **IDCategoria** es el campo combinado, pero no está incluido en la salida de la consulta ya que no está contenido en la instrucción **SELECT**. Para insertar el campo combinado, se debe incluir el nombre del campo en la instrucción **SELECT**, en este caso, **Categorias.IDCategoria**.

```
SELECT
    NombreCategoria, NombreProducto
FROM
    Categorias
INNER JOIN
    Productos
ON
    Categorias.IDCategoria = Productos.IDCategoria
```

Además, se pueden enlazar varias cláusulas **ON** en una instrucción **JOIN**:

```
SELECT campos FROM tabla1 INNER JOIN tabla2
ON (tb1.campo1 comp tb2.campo1 AND ON tb1.campo2 comp tb2.
campo2)
OR ON (tb1.campo3 comp tb2.campo3)
```

También se pueden anidar instrucciones **JOIN** utilizando la siguiente sintaxis:

5. Subconsultas y uniones

```
SELECT campos FROM tb1 INNER JOIN (tb2 INNER JOIN [( ]tb3  
[INNER JOIN [( ]tablax [INNER JOIN ...)]  
ON tb3.campo3 comp tbx.campox)]  
ON tb2.campo2 comp tb3.campo3)  
ON tb1.campo1 comp tb2.campo2
```

Un **LEFT JOIN** o un **RIGHT JOIN** puede anidarse dentro de un **INNER JOIN**, pero un **INNER JOIN** no puede anidarse dentro de un **LEFT JOIN** o un **RIGHT JOIN**:

```
SELECT DISTINCT  
    Sum(PrecioUnitario * Cantidad) AS Sales,  
    (Nombre + ' ' + Apellido) AS Name  
FROM  
    Empleados  
INNER JOIN(  
    Pedidos  
INNER JOIN  
    DetallesPedidos  
    ON  
        Pedidos.IdPedido = DetallesPedidos.IdPedido)  
    ON  
        Empleados.IdEmpleado = Pedidos.IdEmpleado  
GROUP BY  
    Nombre + ' ' + Apellido
```

Autocombinación

La **autocombinación** se utiliza para unir una tabla consigo misma

comparando valores de dos columnas con el mismo tipo de datos. Esta es su sintaxis:

```
SELECT
    alias1.columna, alias2.columna, ...
FROM
    tabla1 as alias1, tabla2 as alias2
WHERE
    alias1.columna = alias2.columna
AND
    otras condiciones
```

Una aplicación es la siguiente, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

```
SELECT
    t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre,
    s.puesto
FROM
    empleados AS t, empleados AS s
WHERE
    t.num_sup = s.num_emp
```

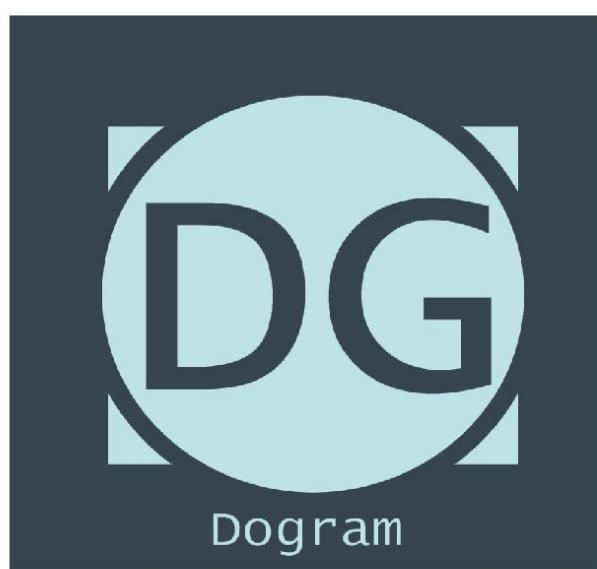
Actividades

A continuación verás las preguntas que deberías saber responder para considerar aprendido el capítulo.

Test de autoevaluación

1. *Identifica la sintaxis de una subconsulta.*
 2. *Indica en qué ocasión podrías utilizar una subconsulta.*
 3. *¿Qué son las agrupaciones?*
 4. *¿Para qué sirven las combinaciones?*
 5. *Indica un ejemplo de autocombinación.*
-

<https://dogramcode.com/bloglibros/libros-bases-de-datos>



Glosario

ADO: mejoramiento del ODBC, para un acceso más directo y con mejor eficiencia a las bases de datos.

Campo: detalle de un dato que es de un tipo específico.

DBMS: *Database Management System* es un sistema que esta desarrollado para manejar una o varias bases de datos, contrario a un simple archivo de texto, que no necesita un sistema específico para manejarlo.

DDL: *Data Definition Language*, instrucciones SQL que permiten definir y modificar la estructura de los datos.

Disparador: es un código compilado en el servidor, que se ejecuta cuando se hacen ciertas acciones sobre una tabla (inserción, modificación, etcétera). Puede disparar antes o después del cambio. Si dispara antes del cambio, permite modificar el valor que se va a agregar/modificar. Si dispara después, puede variar otras tablas.

DML: *Data Modeling Language*, instrucciones SQL que permiten modificar y visionar los datos.

Esquema: es el conjunto de todos los objetos de la base de datos que pertenecen a un mismo usuario.

Funciones: una función es como un procedimiento si no tiene siempre un solo parámetro de salida.

Índice: es un pequeño archivo de uno o varios campos ordenados de una tabla, que permite aumentar el rendimiento de las encuestas utilizando estos campos como filtro.

JDBC: *Java Database Connector*, driver Java que permite conectarse a una base de datos de una manera estándar. El driver JDBC funciona como un puente entre la aplicación llamando y la base de datos.

5. Subconsultas y uniones

ODBC: Open Database Connectivity, API de Windows que permite conectarse a una base de datos de una manera estándar. El driver ODBC funciona como un puente entre la aplicación llamando y la base de datos.

Procedimientos: un procedimiento es un código compilado que permite ejecutar acciones sobre la base de datos (servidor). El procedimiento puede tener cero o varios parámetros de entrada y de salida.

RDBMS: Relational Database Management System es un sistema de bases de datos relacionadas. Es decir, una base de datos que contiene varias tablas relacionadas entre sí, contrario a las bases de datos que contienen una o varias tablas sin relación entre ellas.

Registros: conjunto de campos que pertenecen a un mismo dato.

Secuencias: las secuencias son contadores manejados por el servidor, que los incrementa y se arregla para evitar dobles valores. Ciertos servidores, como MySQL o SQL Server, no tienen secuencias, pero poseen una opción de incremento automático de un campo numérico de una tabla.

Sinónimo: es un nombre que se refiere a un objeto de la base de datos.

Esto permite crear atajos para ciertos objetos de esa base.

SQL: Simple Query Language es un lenguaje utilizado para hacer encuestas y acciones sobre bases de datos. Existe un lenguaje SQL estándar, pero cada base de datos tiene sus excepciones en cuanto a este estándar.

Tabla: conjunto de registros que contienen los mismos campos, es decir, el mismo tipo de información.

Vista: una vista es una definición lógica de una parte o de un conjunto de tablas. El objetivo de una vista puede ser variado: esconder las tablas originales, limitar el acceso a los datos (seguridad), simplificar u optimizar el extracto de datos.