O'REILLY®

4ª edición

SQL

Guía de bolsillo

Guía de uso de SQL

Alice Zhao

O "REILLY

Guía de bolsillo SQL

Si utiliza SQL en su trabajo diario como analista de datos, científico de datos o ingeniero de datos, esta popular guía es la referencia ideal para su trabajo. Encontrará muchos ejemplos que abordan las complejidades del lenguaje, junto con aspectos de SQL utilizados en Microsoft SQL Server, MySQL, Oracle Database, PostgreSQL y SQLite.

En esta edición actualizada, la autora Alice Zhao describe cómo estos sistemas de gestión de bases de datos implementan la sintaxis SQL tanto para consultar como para realizar cambios en una base de datos. Encontrará detalles sobre tipos de datos y conversiones, sintaxis de expresiones regulares, funciones de ventana, pivoteo y despivoteo, y mucho más.

- Buscar rápidamente cómo realizar tareas específicas utilizando SQL
- Aplique los ejemplos de sintaxis del libro a sus propias consultas.
- Las consultas SQL de actualización funcionan en cinco bases de datos diferentes
  sistemas de gestión
- Nuevo: Conectar Python y Ra una base de datos relacional
- **Novedad: Consulta de las preguntas más frecuentes sobre SQL en la sección "Cómo".**
  ¿Yo?

DATOS

US $29.

ISBN: 978 -1-49
2-09040-3

99

Twitter: @oreillymedia
facebook.com/oreilly

CUARTA EDICIÓN

Guía de bolsillo SQL

Alice Zhao

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Guía de bolsillo SQL

por Alice Zhao

Publicado por O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Los libros de O'Reilly pueden adquirirse para uso educativo, comercial o promocional. También están disponibles ediciones en línea de la mayoría de los títulos (http://oreilly.com). Para más información, póngase en contacto con nuestro departamento de ventas a empresas e instituciones: 800-998-9938 o corporate@oreilly.com.

Editor de adquisiciones: Andy Kwan
Editor de desarrollo: Amelia Blevins y Jeff Bleiel
Editora de producción: Caitlin Ghegan
Correctora: Piper Editorial Consulting, LLC
Corrector de pruebas: James Fraleigh
Indexadora: Ellen Troutman-Zaig
Diseñador de interiores: David
Futato **Diseñador de cubiertas:**
Karen Montgomery **Ilustradora:**
Kate Dullea

Septiembre de 2021: Cuarta edición

Historial de revisiones de la cuarta edición

2021-08-26: Primera publicación

Consulte https://oreil.ly/sqlpocketerrata para obtener más información.

El logotipo de O'Reilly es una marca registrada de O'Reilly Media, Inc. *SQL Pocket Guide*, la imagen de portada y la imagen comercial relacionada son marcas comerciales de O'Reilly Media, Inc.

978-1-492-09040-3
[LSI]

Índice

Prefacio

¿Por qué SQL?

Desde que se publicó la última edición de *SQL Pocket Guide*, mucho ha cambiado en el mundo de los datos. La cantidad de datos generados y recopilados se ha disparado, y se han creado numerosas herramientas y trabajos para gestionar la afluencia de datos. A pesar d e todos los cambios, SQL ha seguido siendo una parte integral del panorama de los datos.

En los últimos 15 años, he trabajado como ingeniero, consultor, analista y científico de datos, y he utilizado SQL en cada u n a de mis funciones. Incluso si mis responsabilidades principales se centraban en otra herramienta o habilidad, tenía que conocer SQL para poder acceder a los datos de una empresa.

> *Si hubiera un premio de lenguaje de programación al mejor actor secundario, SQL se llevaría el galardón.*

A medida que surgen nuevas tecnologías, SQL sigue siendo lo más importante a la hora de trabajar con datos: . Las soluciones de almacenamiento en la nube como Amazon Redshift y Google BigQuery requieren que los usuarios escriban consultas SQL para extraer datos. Los marcos de trabajo de procesamiento de datos distribuidos como Hadoop y Spark cuentan con Hive y Spark SQL, respectivamente, que proporcionan interfaces similares a SQL para que los usuarios analicen los datos.

SQL existe desde hace casi cinco décadas y no va a desaparecer pronto. Es uno de los lenguajes de programación más antiguos que todavía se utiliza ampliamente hoy en día, y estoy muy contento de compartir lo último y lo mejor con usted en este libro.

Objetivos de este libro

Existen muchos libros sobre SQL, desde los que enseñan a los principiantes a programar en SQL hasta especificaciones técnicas detalladas para administradores de bases de datos. Este libro no pretende abarcar todos los conceptos de SQL en profundidad, sino más bien ser una referencia sencilla para cuando:

- Has olvidado alguna sintaxis SQL y necesitas buscarla rápidamente
- Se ha encontrado con un conjunto ligeramente diferente de herramientas de bases de datos en un nuevo trabajo y necesita buscar las diferencias de matiz
- Lleva un tiempo centrado en otro lenguaje de codificación y necesita un repaso rápido sobre el funcionamiento de SQL.

Si SQL desempeña un importante papel de apoyo en su trabajo, ésta es la guía de bolsillo perfecta para usted.

Actualizaciones de la cuarta edición

La tercera edición de la *Guía de Bolsillo SQL* de Jonathan Gennick se publicó en 2010, y tuvo una buena acogida entre los lectores. En la cuarta edición he realizado las siguientes actualizaciones:

- Se ha actualizado la sintaxis para Microsoft SQL Server, MySQL, Oracle Database y PostgreSQL. Se ha eliminado Db2 de IBM debido a su menor popularidad, y se ha añadido SQLite debido a su mayor popularidad.
- La tercera edición de este libro estaba organizada

alfabéticamente. En la cuarta edición he reorganizado las secciones para agrupar conceptos similares. Sigue habiendo un

índice al final de este libro que enumera los conceptos alfabéticamente.

- Debido al número de analistas de datos y científicos de datos que ahora utilizan SQL en sus trabajos, he añadido secciones sobre cómo utilizar SQL con Python y R (lenguajes de programación de código abierto populares), así como un curso intensivo de SQL para aquellos que necesitan un repaso rápido.

Preguntas más frecuentes (SQL)

El último capítulo de este libro se titula "¿Cómo...?" e incluye las preguntas más frecuentes de los principiantes en SQL o de quienes llevan tiempo sin utilizarlo.

Es un buen punto de partida si no recuerdas exactamente la palabra clave o el concepto que estás buscando. Ejemplos de preguntas

- ¿Cómo encuentro las filas que contienen valores duplicados?
- ¿Cómo selecciono filas con el valor máximo de otra columna?
- ¿Cómo concateno el texto de varios campos en uno solo?

Navegar por este libro

Este libro está organizado en tres secciones.

I. Conceptos básicos

- Los capítulos 1 a 3 presentan palabras clave, conceptos y herramientas básicas para escribir código SQL.
- El capítulo 4 desglosa cada cláusula de una consulta SQL.

II. Objetos de base de datos, tipos de datos y funciones

- El capítulo 5 enumera las formas más comunes de crear y modificar objetos dentro de una base de datos.
- El Capítulo 6 enumera los tipos de datos comunes que se utilizan en SQL.
- El Capítulo 7 enumera los operadores y funciones más comunes en SQL.

III. Conceptos avanzados

- Los capítulos 8 y 9 explican conceptos avanzados de consulta, como uniones, sentencias case, funciones de ventana, etc.
- El capítulo 10 recorre las soluciones a algunas de las preguntas SQL más frecuentes.

Convenciones utilizadas en este libro

En este libro se utilizan las siguientes convenciones tipográficas:

Cursiva
: Indica nuevos términos, URL, direcciones de correo electrónico, nombres de archivo y extensiones de archivo.

`Anchura constante`
: Se utiliza para listados de programas, así como dentro de párrafos para referirse a elementos del programa como nombres de variables o funciones, bases de datos, tipos de datos, variables de entorno, estados y palabras clave.

`Negrita de anchura constante`
: Muestra comandos u otros textos que deben ser tecleados literalmente por el usuario, o valores determinados por el contexto.

CONSEJO

Este elemento significa un consejo o sugerencia.

NOTA

Este elemento significa una nota general.

ADVERTENCIA

Este elemento indica una advertencia o precaución.

Uso de ejemplos de código

Si tiene alguna pregunta técnica o algún problema al utilizar los ejemplos de código, envíe un correo electrónico a bookquestions@oreilly.com.

Este libro está aquí para ayudarle a realizar su trabajo. En general, si se ofrece código de ejemplo con este libro, puede utilizarlo en sus programas y documentación. No es necesario que se ponga en contacto con nosotros para solicitar permiso, a menos que esté reproduciendo una parte significativa del código. Por ejemplo, escribir un programa que utilice varios trozos de código de este libro no requiere permiso. Vender o distribuir ejemplos de libros de O'Reilly sí requiere permiso. Responder a una pregunta citando este libro y el código de ejemplo no requiere permiso. Incor- porar una cantidad significativa de código de ejemplo de este libro en la documentación de su producto sí requiere permiso.

Apreciamos, pero generalmente no exigimos, la atribución. Una atribución suele incluir el título, el autor, la editorial y el ISBN. Por ejemplo: "*SQL Pocket Guide, 4ª ed.* por Alice Zhao (O'Reilly). Copyright 2021 Alice Zhao, 978-1-492-209040-3".

Si cree que el uso que hace de los ejemplos de código no se ajusta al uso legítimo o a los permisos mencionados anteriormente, no dude en ponerse en contacto con nosotros en *permissions@oreilly.com*.

Aprendizaje en línea de O'Reilly

Durante más de 40 años, *O'Reilly Media* ha proporcionado formación tecnológica y empresarial, conocimientos y perspectivas para ayudar a las empresas a alcanzar el éxito.

Nuestra red única de expertos e innovadores comparten sus conocimientos y experiencia a través de libros, artículos y nuestra plataforma de aprendizaje en línea. La plataforma de aprendizaje en línea de O'Reilly le ofrece acceso bajo demanda a cursos de formación en directo, rutas de aprendizaje en profundidad, entornos de codificación interactivos y una amplia colección de textos y vídeos de O'Reilly y de más de 200 editores. Para más información, visite http://oreilly.com.

Cómo ponerse en contacto con nosotros

Por favor, dirija sus comentarios y preguntas sobre este libro al editor:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (en Estados Unidos o Canadá)
707-829-0515 (internacional o local)
707-829-0104 (fax)

Disponemos de una página web para este libro, donde se enumeran erratas, exámenes ples y cualquier información adicional. Puede acceder a esta página en https://oreil.ly/jreAj.

Envíe un correo electrónico *a bookquestions@oreilly.com* para hacer comentarios o preguntas técnicas sobre este libro.

Para noticias e información sobre nuestros libros y cursos, visite http://oreilly.com.

Encuéntrenos en Facebook:

http://facebook.com/oreilly. Síganos en Twitter:

http://twitter.com/oreillymedia. Véanos en YouTube:

http://youtube.com/oreillymedia.

Agradecimientos

Gracias a Jonathan Gennick por crear esta guía de bolsillo desde cero y escribir las tres primeras ediciones, y a Andy Kwan por confiar en mí para continuar con la publicación.

No podría haber terminado este libro sin la ayuda de mis editores Amelia Blevins, Jeff Bleiel y Caitlin Ghegan, y de mis revisores técnicos Alicia Nevels, Joan Wang, Scott Haines y Thomas Nield. Agradezco sinceramente el tiempo que han dedicado a leer cada página de este libro. Sus comentarios han sido inestimables.

A mis padres, gracias por fomentar mi amor por aprender y crear. A mis hijos Henry y Lily, su entusiasmo por este libro me alegra el corazón. Por último, a mi marido, Ali, gracias por todas tus notas sobre este libro, por tus ánimos y por ser mi mayor admirador.

CAPÍTULO 1

Curso intensivo de SQL

Este breve capítulo pretende ponerle rápidamente al día en terminología y conceptos básicos de SQL.

¿Qué es una base de datos?

Empecemos por lo básico. Una *base de datos* es un lugar donde almacenar datos de forma organizada. Hay muchas formas de organizar los datos y, como resultado, hay muchas bases de datos entre las que elegir. Las dos categorías en las que se dividen las bases de datos son *SQL* y *NoSQL*.

SQL

SQL es la abreviatura de *Structured Query Language (lenguaje de consulta estructurado)*. Imagina que tienes una aplicación que recuerda todos los cumpleaños de tus amigos. SQL es el lenguaje más popular que utilizarías para hablar con esa aplicación.

> *Inglés*: "Hey app. ¿Cuándo es el cumpleaños de mi marido?"
> *SQL*: `SELECT * FROM cumpleaños`
> `WHERE persona = 'marido';`

Las bases de datos SQL suelen denominarse *bases de datos relacionales* porque están formadas por relaciones, que más

comúnmente se denominan tablas. Una base de datos está formada por muchas tablas conectadas entre sí. La Figura 1-1 muestra una imagen de una relación en una base de datos SQL .

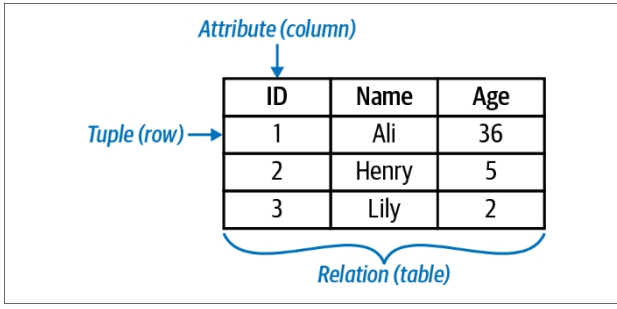

Figura 1-1. Una relación (también conocida como tabla) en una base de datos SQL

Lo principal a tener en cuenta sobre las bases de datos SQL es que requieren *esquemas* predefinidos. Puedes pensar en un esquema como la forma en que se organizan o estructuran los datos en una base de datos. Supongamos que quieres crear una tabla. Antes de cargar cualquier dato en la tabla, primero hay que decidir la estructura de la tabla, incluyendo cosas como qué columnas hay en la tabla, si esas columnas contienen valores enteros o decimales, etc.

Sin embargo, llega un momento en que los datos no pueden organizarse de forma tan estructurada. Puede que tus datos tengan campos variables o que necesites una forma más eficaz de almacenar y acceder a una gran cantidad de datos. Ahí es donde entra NoSQL.

NoSQL

NoSQL *no es sólo SQL*. No se tratará en detalle en este libro, pero quería señalarlo porque el término ha crecido mucho en popularidad desde la década de 2010 y es importante entender que hay formas de almacenar datos más allá de las tablas.

Las bases de datos NoSQL suelen denominarse *bases de datos no relacionales* y las hay de todos los tamaños y formas. Sus principales características son que tienen esquemas dinámicos (lo que significa que el esquema no tiene que estar bloqueado de antemano) y permiten el escalado horizontal (lo que

significa que los datos pueden extenderse por múltiples máquinas).

La base de datos NoSQL más popular es *MongoDB*, que es más específicamente una base de datos de documentos. La Figura 1-2 muestra una imagen de de cómo se almacenan los datos en MongoDB. Observarás que los datos ya no están en una tabla estructurada y que el número de campos (simi- lar a una columna) varía para cada documento (similar a una fila).

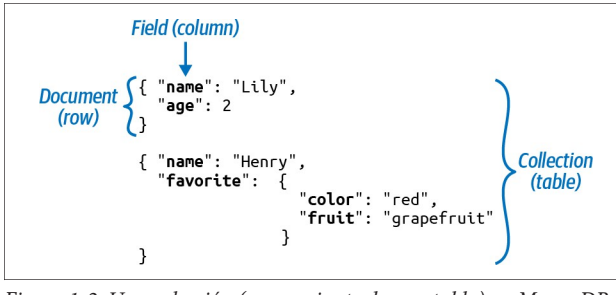

Figura 1-2. Una colección (una variante de una tabla) en MongoDB, una base de datos NoSQL

Dicho todo esto, este libro se centra en las bases de datos SQL. Incluso con la introducción de NoSQL, la mayoría de las empresas siguen almacenando la mayor parte de sus datos en tablas de bases de datos relacionales.

Sistemas de gestión de bases de datos (SGBD)

Es posible que hayas oído términos como *PostgreSQL* o *SQLite* y te preguntes en qué se diferencian de SQL. Se trata de dos tipos de *sistemas de gestión de bases de datos (*SGBD), es decir, software utilizado para trabajar con una base de datos.

Esto incluye aspectos como averiguar cómo importar datos y organizarlos, así como aspectos como gestionar la forma en que los usuarios u otros programas de acceden a los datos. Un *sistema de gestión de bases de datos* relacionales (RDBMS) es un software específico para bases de datos relacionales, o bases de datos formadas por tablas.

Cada RDBMS tiene una implementación diferente de SQL, lo que significa que la sintaxis varía ligeramente de un software a otro. Por ejemplo, así es como se imprimirían 10 filas de datos

en 5 RDBMS diferentes:

MySQL, PostgreSQL y SQLite

```
SELECT * FROM cumpleaños LIMIT 10;
```

Microsoft SQL Server

```
SELECT TOP 10 * FROM cumpleaños;
```

Base de datos Oracle

```
SELECT * FROM cumpleaños WHERE ROWNUM <= 10;
```

Sintaxis SQL en Google

Cuando busques sintaxis SQL en Internet, incluye siempre en la búsqueda el RDBMS con el que estés trabajando. Cuando aprendí SQL por primera vez, no podía entender por qué mi código copiado y pegado de Internet no funcionaba y ¡ésta era la razón!

Haz esto.

Búsqueda: *create table datetime* ***postgresql***

→ Resultado: `marca de tiempo`

Search: *crear tabla datetime* ***microsoft sql server***

→ Resultado: `datetime`

Esto no.

Búsqueda: *create table datetime*

→ Resultado: la sintaxis podría ser para cualquier RDBMS.

Este libro cubre los fundamentos de SQL junto con los matices de cinco populares sistemas de gestión de bases de datos: Microsoft SQL Server, MySQL, Oracle Database, PostgreSQL y SQLite.

Algunos son propietarios, es decir, son propiedad de una empresa y su uso cuesta dinero, y otros son de código abierto, es decir, s u uso es gratuito para cualquiera. La Tabla 1-1 detalla las diferencias entre los RDBMS.

Tabla 1-1. Tabla comparativa de RDBMS

RDBMS	Propietario	Destacados
Microsoft SQL Server	Microsoft	- RDBMS propietario popular - A menudo se utiliza junto con otros productos de Microsoft, como Microsoft Azure y el marco .NET. - Común en la plataforma Windows - También conocido como *MSSQL* o *SQL Server*
MySQL	Abrir Fuente	- SGBDR populares de código abierto - A menudo se utiliza junto con lenguajes de desarrollo web como HTML/CSS/Javascript - Adquirida por Oracle, aunque sigue siendo de código abierto
Base de datos Oracle	Oracle	- SGBDR propietario popular - A menudo se utiliza en grandes empresas, dada la cantidad de funciones, herramientas y asistencia disponibles. - También llamado simplemente *Oracle*
PostgreSQL	Código abierto	- Crece rápidamente su popularidad - A menudo se utiliza junto con tecnologías de código abierto como Docker y Kubernetes. - Eficaz y excelente para grandes conjuntos de datos
SQLite	Abrir Fuente	- El motor de base de datos más utilizado del mundo - Común en plataformas iOS y Android - Ligero y estupendo para una base de datos pequeña

NOTA

Avanzando en este libro:

- Microsoft SQL Server se denominará *SQL Server*.
- La base de datos Oracle se denominará *Oracle*.

Encontrará instrucciones de instalación y fragmentos de código para cada RDBMS en Software RDBMS, en el Capítulo 2.

Una consulta SQL

Un acrónimo común en el mundo SQL es *CRUD*, que significa para "Crear, Leer, Actualizar y Eliminar". Estas son las cuatro operaciones principales que están disponibles dentro de una base de datos.

Declaraciones SQL

Las personas que tienen *acceso de lectura y escritura* a una base de datos pueden realizar las cuatro operaciones. Pueden crear y borrar tablas, actualizar datos en tablas y leer datos de tablas. En otras palabras, tienen todo el poder.

Escriben *sentencias SQL*, que es código SQL general que puede escribirse para realizar cualquiera de las operaciones CRUD. Estas personas suelen tener títulos como *administrador de bases de datos* (DBA) o *ingeniero de bases de datos*.

Consultas SQL

Las personas que tienen *acceso de lectura* a una base de datos sólo pueden realizar la operación de lectura, es decir, pueden consultar los datos de las tablas.

Escriben *consultas SQL*, que son un tipo más específico de sentencia SQL. Las consultas se utilizan para buscar y mostrar datos, lo que también se conoce como "leer" datos. Esta acción se denomina a veces *consulta de tablas*. Estas personas suelen tener títulos como *analista de datos* o *científico de datos*.

Las dos secciones siguientes son una guía rápida para escribir consultas SQL, ya que es el tipo de código SQL más común que verás. Encontrará más detalles sobre la creación y actualización de tablas en el Capítulo 5.

La sentencia SELECT

La consulta SQL más básica (que funcionará en cualquier software SQL) es:

```
SELECT * FROM mi_tabla;
```

que dice, muéstrame todos los datos dentro de la tabla llamada mi_tabla: `todas` las columnas y todas las filas.

Aunque SQL no distingue entre mayúsculas y minúsculas (`SELECT` y `select` son equiva- lentes), observará que algunas palabras están en mayúsculas y otras no.

- Las palabras en mayúsculas de la consulta se denominan *palabras clave*, lo que significa que SQL las ha reservado para realizar algún tipo de operación con los datos en .
- Todas las demás palabras se escriben en minúsculas. Esto incluye nombres de tablas, columnas, etc.

Los formatos de mayúsculas y minúsculas no son obligatorios, pero es una buena convención de estilo para facilitar la lectura.

Volvamos a esta consulta:

```
SELECT * FROM mi_tabla;
```

Digamos que en lugar de devolver todos los datos en su estado actual, quiero:

- Filtrar los datos
- Ordenar los datos

Aquí es donde se modificaría la sentencia `SELECT` para incluir algunas *cláusulas más*, y el resultado sería algo parecido a esto:

```
SELECCIONAR *
FROM mi_tabla
WHERE columna1 > 100
ORDER BY columna2;
```

En el capítulo 4 encontrará más información sobre todas las cláusulas, pero lo principal es que deben enumerarse siempre en el mismo orden.

Memorice esta orden

Todas las consultas SQL contendrán alguna combinación de estas cláusulas. Si no recuerda nada más, ¡recuerde este orden!

```
SELECCIONE    -- columnas a mostrar
DESDE         -- tabla(s) de la(s) que extraer
DONDE         -- filtrar filas
GRUPO POR     -- divide las filas en
grupos HAVING -- filtrar filas
agrupadas ORDER BY     -- columnas a
ordenar
```

NOTA

El `--` es el comienzo de un comentario en SQL, lo que significa que el texto después de él es sólo para la documentación y el código no se ejecutará.

En la mayoría de los casos, las cláusulas `SELECT` y `FROM` son obligatorias y el resto de cláusulas son opcionales. La excepción es si se está seleccionando una función de base de datos en particular, entonces sólo se requiere el SELECT.

La mnemotecnia clásica para recordar el orden de las cláusulas es:

Los pies sudorosos desprenden olores horribles.

Si no quieres pensar en pies sudorosos cada vez que escribes una consulta, aquí tienes una que me he inventado:

Empieza los viernes con la avena casera de la abuela.

Orden de ejecución

El orden en que se ejecuta el código SQL no es algo que se enseñe normalmente en un curso de SQL para principiantes, pero lo incluyo aquí porque es una pregunta común que recibí cuando enseñaba SQL a estudiantes que venían de una formación de codificación en Python.

Una suposición sensata sería que el orden en que se *escriben* las cláusulas es el mismo en que el ordenador *ejecuta* las cláusulas de , pero no es así. Después de ejecutar una consulta, este es el orden en que el ordenador trabaja con los datos:

1. `DESDE`
2. `DONDE`
3. `GRUPO POR`
4. `TENIENDO`
5. `SELECCIONE`
6. `ORDENAR POR`

Comparado con el orden en el que se escriben las cláusulas, notarás que `SELECT se` ha movido a la quinta posición . Lo importante es que SQL funciona en este orden:

1. Recoge todos los datos con el `FROM`
2. Filtra filas de datos con la función `WHERE`
3. Agrupa las filas con `GROUP BY`
4. Filtra las filas agrupadas con `HAVING`
5. Especifica las columnas que se mostrarán con `SELECT`
6. Reordena los resultados con el `ORDER BY`

Un modelo de datos

Me gustaría dedicar la sección final del curso intensivo a repasar u n *modelo de datos* sencillo y señalar algunos términos que a menudo oirás en en divertidas conversaciones sobre SQL en la oficina.

Un modelo de datos es una visualización que resume cómo se relacionan entre sí todas las tablas de una base de datos, junto con algunos detalles sobre cada tabla. La Figura 1-3 es un modelo de datos sencillo de una base de datos de calificaciones de alumnos.

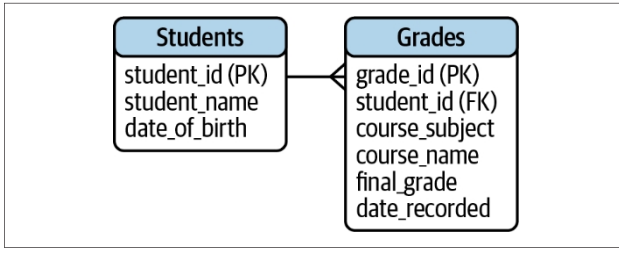

Figura 1-3. Modelo de datos de las calificaciones de los alumnos

La Tabla 1-2 enumera los términos técnicos que describen lo que ocurre en el modelo de datos.

Tabla 1-2. Términos utilizados para describir el contenido de un modelo de datos

Término	Definición	Ejemplo
Base de datos	Una base de datos es un lugar para almacenar datos de forma organizada.	Este modelo de datos muestra todos los datos de la base de datos de calificaciones de alumnos.
Tabla	Una tabla está formada por filas y columnas . En el modelo de datos, s e representan mediante rectángulos.	Existen dos tablas en la base de datos de calificaciones de alumnos: `Alumnos` y `Calificaciones`.

Término	Definición	Ejemplo
ColumnaUna	tabla consta de varias columnas, que a veces se denominan atributos o campos . Cada columna contiene un tipo concreto de datos. En el modelo de datos, todas las columnas de una tabla se enumeran dentro de cada rectángulo.	En la tabla `Alumnos`, las columnas son `student_id`, `student_name` y `date_of_birth`.
Clave primaria	Una *clave primaria* única identifica cada fila de datos de una tabla. Una clave primaria puede estar formada por una o varias columnas en una tabla. En un modelo de datos, se marca como pk o con un icono de clave.	En la tabla `Estudiantes`, la clave principal es la columna `student_id`, lo que significa que el valor `student_id` es diferente para cada fila de datos.
Clave externa	Una *clave externa* en una tabla se refiere a una clave primaria en otra tabla. Las dos tablas pueden estar vinculadas por la columna común. Una tabla puede tener varias claves externas. En un modelo de datos, se marca como fk.	En la tabla `Grados`, `student_id` es una clave externa, lo que significa que los valores de esa columna coinciden con los valores de la columna de clave primaria correspondiente en la tabla `Alumnos`.
Relación	Una *relación* describe cómo los las filas de una tabla corresponden a las filas de otra tabla. En un modelo de datos, se representa mediante una línea con símbolos en los extremos. Los tipos más comunes son las relaciones uno a uno y uno a muchos.	En este modelo de datos, las dos tablas tienen una relación de uno a muchos representada por la horquilla. Un alumno puede tener muchas calificaciones, o una fila de la tabla `Alumnos` se asigna a varias filas de la tabla `Calificaciones`.

Encontrará más información sobre estos términos en "Creación de tablas" en la página 97 del capítulo 5.

Te preguntarás por qué pasamos tanto tiempo leyendo un modelo de datos en lugar de escribir código SQL. La razón es que a menudo escribirás consultas que enlazan varias tablas, por lo que es una buena idea familiarizarse primero con el modelo de datos para saber cómo se conectan todas ellas.

Los modelos de datos suelen encontrarse en los archivos de documentación de las empresas. Es posible que desee imprimir los modelos de datos con los que trabaja con frecuencia, tanto para facilitar su consulta como para decorar su escritorio.

También puede escribir consultas en un RDBMS para buscar información contenida en un modelo de datos, como las tablas de una base de datos , las columnas de una tabla o las restricciones de una tabla.

¡Y ese es tu curso intensivo!

El resto de este libro pretende ser una obra de referencia y no es necesario leerlo en orden. Utilícelo para buscar conceptos, palabras clave y normas.

CAPÍTULO 2

¿Dónde puedo escribir código SQL?

Este capítulo cubre tres lugares donde puedes escribir código SQL:

Software RDBMS
: Para escribir código SQL, primero hay que descargar un RDBMS como MySQL, Oracle, PostgreSQL, SQL Server o SQLite. Los matices de cada RDBMS se destacan en "Software RDBMS" en la página 14.

Herramientas de bases de datos
: Una vez descargado un RDBMS, la forma más básica de escribir código SQL es a través de una *ventana de terminal*, que es una pantalla en blanco y negro de sólo texto. La mayoría de la gente prefiere utilizar una *herramienta de base de datos, que* es una aplicación más fácil de usar que se conecta a un RDBMS entre bastidores.

  Una herramienta de base de datos tendrá una *interfaz gráfica de usuario (*GUI), que permite a los usuarios explorar visualmente las tablas y más editar fácilmente el código SQL. "Herramientas de base de datos" en la página 20 explica cómo conectar una herramienta de base de datos a un RDBMS.

Otros lenguajes de programación
: SQL puede escribirse en muchos otros lenguajes de

programación. Este capítulo se centra en dos de ellos: Python y R. Son populares lenguajes de programación de código abierto.

utilizados por los científicos y analistas de datos, que a menudo también necesitan escribir código SQL.

En lugar de ir y venir entre Python/R y un RDBMS, puedes conectar Python/R directamente a un RDBMS y escribir código SQL dentro de Python/R. "Otros lenguajes de programación" en la página 24 explica cómo hacerlo paso a paso.

Software RDBMS

Esta sección incluye instrucciones de instalación y fragmentos breves de código para los cinco RDBMS que se tratan en este libro.

¿Qué RDBMS elegir?

Si trabaja en una empresa que ya utiliza un RDBMS , tendrá que utilizar el mismo.

Si está trabajando en un proyecto personal, tendrá que decidir qué RDBMS utilizar. Puedes consultar la Tabla 1-1 del Capítulo 1 para repasar los detalles de algunos de los más populares.

Inicio rápido con SQLite

¿Quieres empezar a escribir código SQL cuanto antes? SQLite es el RDBMS más rápido de configurar.

Comparado con los otros RDBMS de este libro, es menos seguro y no puede manejar múltiples usuarios, pero proporciona funcionalidad SQL básica en un paquete compacto.

Por ello, he colocado SQLite al principio de cada sección de este capítulo, ya que su configuración suele ser más sencilla que la de los demás.

¿Qué es una ventana de terminal?

A menudo me referiré a una ventana de terminal en este capítulo, porque una vez que haya descargado un RDBMS, es la forma más básica de interactuar con el RDBMS.

Una *ventana de terminal es* una aplicación de su ordenador que suele tener un fondo negro y sólo permite introducir texto. El nombre de la aplicación varía según el sistema operativo:

- En Windows, utilice la aplicación Símbolo del sistema.
- En macOS y Linux, utilice la aplicación Terminal.

Al abrir una ventana de terminal, verás un *símbolo del sistema, que se* parece a un `>` seguido de un cuadro parpadeante. Este significa que está listo para recibir comandos de texto del usuario.

CONSEJO

Las siguientes secciones incluyen enlaces para descargar los instaladores de RDBMS para Windows, macOS y Linux.

En macOS y Linux, una alternativa a la descarga de un instalador de es utilizar el gestor de paquetes Homebrew. Una vez instalado Homebrew, puedes ejecutar comandos de `instalación brew` sencillos desde el Terminal para realizar todas las instalaciones RDBMS.

SQLite

SQLite es gratuito y la instalación más ligera, lo que significa que no ocupa mucho espacio en tu ordenador y es extremadamente rápido de configurar. Para Windows y Linux, SQLite Tools puede descargarse desde la página de descargas de SQLite. macOS viene con SQLite ya instalado.

CONSEJO

La forma más sencilla de empezar a utilizar SQLite es abrir una ventana termi- nal y escribir **`sqlite3`**. Con este enfoque, sin embargo, todo se hace en la memoria, lo que significa q u e los cambios no se guardarán una vez que cierre SQLite.

```
> sqlite3
```

Si desea que sus cambios se guarden, debe conectarse a una base de datos al abrir con la siguiente sintaxis:

```
> sqlite3 mi_nueva_db.db
```

El símbolo del sistema para SQLite tiene este aspecto:

```
sqlite>
```

Algo de código rápido para probar cosas:

```
sqlite> CREATE TABLE test (id int, num int);
sqlite> INSERT INTO test VALUES (1, 100), (2, 200);
sqlite> SELECT * FROM prueba LIMIT 1;

1|100
```

Para mostrar bases de datos, mostrar tablas y salir:

```
sqlite> .bases de
datos sqlite> .tablas
sqlite> .quit
```

CONSEJO

Si desea mostrar los nombres de las columnas en la salida, escriba:

```
sqlite> .headers on
```

Para ocultarlos de nuevo, teclea:

```
sqlite> .headers off
```

MySQL

MySQL es gratuito, aunque ahora es propiedad de Oracle. MySQL Community Server puede descargarse desde la página MySQL Com- munity Downloads. En macOS y Linux, alternativamente, puede hacer la instalación con Homebrew escribiendo **`brew install mysql`** en el Terminal.

El símbolo del sistema para MySQL tiene este aspecto:

```
mysql>
```

Algo de código rápido para probar cosas:

```
mysql> CREATE TABLE test (id int, num int);
mysql> INSERT INTO test VALUES (1, 100), (2, 200);
mysql> SELECT * FROM prueba LIMIT 1;

+------+ ------+
| id|  num |
+------+ ------+
|1    | 100 |
+------+ ------+
1 fila en juego (0.00 seg)
```

Para mostrar bases de datos, cambiar de base de datos, mostrar tablas y salir:

```
mysql> mostrar bases de
datos; mysql> conectar
otra_db; mysql> mostrar
tablas;
mysql> quit
```

Oracle

Oracle es propietario y funciona en máquinas Windows y Linux . Oracle Database Express Edition, la edición gratuita, puede descargarse de la página de descargas de Oracle Database XE.

El símbolo del sistema para Oracle tiene este aspecto:

```
SQL>
```

Algo de código rápido para probar cosas:

```
SQL> CREATE TABLE test (id int, num int);
SQL> INSERT INTO test VALUES (1, 100);
SQL> INSERT INTO test VALUES (2, 200);
SQL> SELECT * FROM test WHERE ROWNUM <=1;

        ID         NUM
---------- ----------
         1         100
```

Para mostrar bases de datos, mostrar todas las tablas (incluidas las tablas del sistema), mostrar tablas creadas por el usuario y salir:

```
SQL> SELECT * FROM nombre_global;
SQL> SELECT table_name FROM all_tables;
SQL> SELECT table_name FROM user_tables;
SQL> quit
```

PostgreSQL

PostgreSQL es gratuito y se utiliza a menudo junto con otras tecnologías de código abierto. PostgreSQL puede descargarse desde la página de descargas de PostgreSQL. En macOS y Linux, como alternativa, puede realizar la instalación con Homebrew escribiendo **`brew install postgresql`** en el Terminal.

El símbolo del sistema para PostgreSQL tiene este aspecto:

```
postgres=#
```

Algo de código rápido para probar cosas:

```
postgres=# CREATE TABLE test (id int, num int);
postgres=# INSERT INTO test VALUES (1, 100),
  (2, 200);
postgres=# SELECT * FROM prueba LIMIT 1;

 id | num
----+-----
```

```
  1 | 100
(1 fila)
```

Para mostrar bases de datos, cambiar de base de datos, mostrar tablas y salir:

```
postgres=# \l
postgres=# \c otra_db
postgres=# \d
postgres=# \q
```

CONSEJO

Si alguna vez ves `postgres-#`, significa que has olvidado-ten un punto y coma al final de una sentencia SQL. Escriba `;` y debería volver a ver `postgres=#`.

Si alguna vez ves `:`, significa que has sido cambiado automáticamente al editor de texto vi, y puedes salir tecleando `q`.

Servidor SQL

SQL Server es propietario (propiedad de Microsoft) y funciona en máquinas Windows y Linux. También puede instalarse a través de Docker. SQL Server Express, la edición gratuita, puede descargarse desde la página de descargas de Microsoft SQL Server.

El símbolo del sistema para SQL Server tiene el siguiente aspecto:

```
1>
```

Algo de código rápido para probar cosas:

```
1> CREAR TABLA test (id int, num int);
2> INSERT INTO test VALUES (1, 100), (2, 200);
3> ir
1> SELECT TOP 1 * FROM prueba;
2> ir
```

```
id          número
----------- -----------
          1         100

(1 fila afectada)
```

Para mostrar bases de datos, cambiar de base de datos, mostrar tablas y salir:

```
1> SELECT name FROM master.sys.databases;
2> go
1> USE otra_db;
2> ir
1> SELECT * FROM esquema_informacion.tablas;
2> go
1> salir
```

NOTA

En *SQL Server*, el código SQL no se ejecuta hasta que se teclea la tecla go en una nueva línea.

Herramientas de bases de datos

En lugar de trabajar con un RDBMS directamente, la mayoría de la gente utiliza una herramienta de base de datos para interactuar con una base de datos. Una herramienta de base de datos viene con una interfaz gráfica de usuario agradable que le permite apuntar, hacer clic y escribir código SQL en un entorno fácil de usar.

Entre bastidores, una herramienta de base de datos utiliza un *controlador de base de datos*, que es un software que ayuda a la herramienta de base de datos a comunicarse con la base de datos. La Figura 2-1 muestra las diferencias visuales entre acceder a una base de datos directamente a través de una ventana de terminal o indirectamente a través de una herramienta de base de datos.

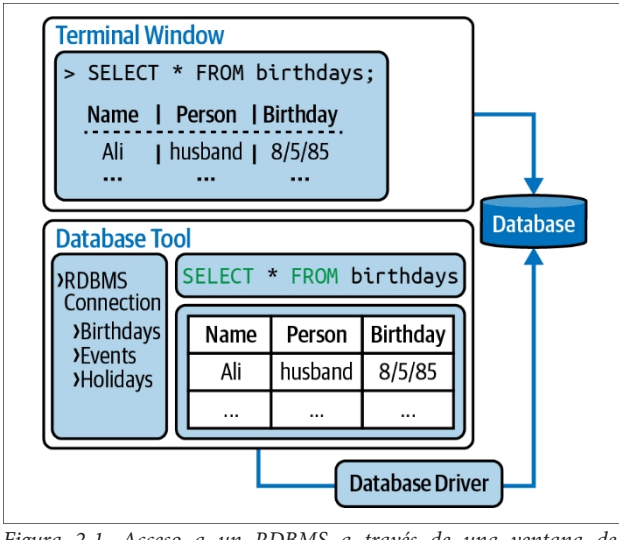

Figura 2-1. Acceso a un RDBMS a través de una ventana de terminal frente a una herramienta de base de datos

Existen varias herramientas de bases de datos. Algunas trabajan específicamente con un único RDBMS, y otras trabajan con múltiples RDBMS. La Tabla 2-1 enumera cada RDBMS junto con una de las herramientas de base de datos más populares para ese RDBMS en particular. Todas las herramientas de bases de datos de la tabla se pueden descargar y utilizar de forma gratuita, , y también existen muchas otras propietarias.

Tabla 2-1. Tabla comparativa de herramientas de bases de datos

RDBMSHerramienta de base de datos	Detalles
Navegador SQLiteDB para SQLite	- Desarrollador distinto de SQLite - Una de las muchas opciones de herramientas para SQLite
MySQLMySQLWorkbench	Mismo desarrollador que
MySQL OracleOracle	SQL Developer -
Desarrollado por Oracle	
PostgreSQL	pgAdmin- Colaboradores diferentes de PostgreSQL

- Incluido con la instalación de PostgreSQL

RDBMS	Herramienta de base de datos	Detalles
Servidor SQL	Servidor SQL Estudio de gestión	- Desarrollado por Microsoft
Múltiples conectar	DBeaver-	Una de las muchas opciones de herramientas para a diversos RDBMS (incluido cualquiera de los cinco anteriores)

Conectar una herramienta de base de datos a una base de datos

Al abrir una herramienta de base de datos, el primer paso es conectarse a una base de datos. Esto puede hacerse de varias maneras:

Opción 1: Crear una nueva base de datos
: Puede crear una base de datos nueva escribiendo un comando `CREATE` declaración:

```
CREAR BASE DE DATOS mi_nueva_db;
```

Después, puede crear tablas para poblar la base de datos. Encontrará más detalles en "Creación de tablas" en la página 97 del Capítulo 5.

Opción 2: Abrir un archivo de base de datos
: Es posible que haya descargado o recibido un archivo con *extensión .db* extensión:

```
mi_nueva_db.db
```

Este archivo *.db* ya contendrá una serie de tablas. Basta con abrirlo en una herramienta de bases de datos y empezar a interactuar con la base de datos.

Opción 3: Conectarse a una base de datos existente
: Puede que quieras trabajar con una base de datos que esté en tu ordenador o en un *servidor remoto*, lo que significa que los datos de están en un ordenador situado en otro lugar. Esto es muy común hoy en día con la *computación en*

la nube, donde la gente utiliza servidores propiedad de empresas como Amazon, Google o Microsoft.

Campos de conexión a la base de datos

Para conectarte a una base de datos, tendrás que rellenar los siguientes campos en una herramienta de base de datos:

Anfitrión
: Dónde se encuentra la base de datos.

  - Si la base de datos está en su ordenador, debe ser *localhost* o *127.0.0.1*.
  - Si la base de datos está en un servidor remoto, debe ser la dirección IP de ese ordenador, por ejemplo: *123.45.678.90*.

Puerto
: Cómo conectarse al RDBMS.

  Ya debería haber un número de puerto por defecto en este campo, y no deberías cambiarlo. Será diferente para cada RDBMS.

  - MySQL: *3306*
  - Oráculo: *1521*
  - PostgreSQL: *5432*
  - SQL Server: *1433*

Base de datos
: El nombre de la base de datos a la que desea conectarse.

Nombre de usuario
: Tu nombre de usuario para la base de datos.

  Puede que ya exista un nombre de usuario por defecto en este campo. Si no recuerda haber configurado un nombre de usuario, mantenga el valor predeterminado.

Contraseña
: Tu contraseña asociada al nombre de usuario.

  Si no recuerdas haber establecido una contraseña para tu nombre de usuario, prueba a dejar este campo en blanco.

NOTA

En el caso de *SQLite,* en lugar de rellenar estos cinco campos de conexión con la base de datos, deberá introducir la ruta del archivo de base de datos *.db* al que intenta conectarse.

Una vez rellenados correctamente los campos de conexión a la base de datos, deberías tener acceso a la misma. Ahora puede utilizar la herramienta de base de datos para encontrar las tablas y los campos que le interesan, y empezar a escribir código SQL.

Otros lenguajes de programación

SQL puede escribirse con otros lenguajes de programación ( ). Este capítulo se centra en dos lenguajes populares de código abierto: Python y R.

Como científico de datos o analista de datos, es probable que realice sus análisis en Python o R, y que también necesite escribir consultas SQL para extraer datos de una base de datos.

Un flujo de trabajo básico para el análisis de datos

1. Escribir una consulta SQL en una herramienta de base de datos.
2. Exporte los resultados como archivo *.csv*.
3. Importe el archivo *.csv* a Python o R.
4. Seguir haciendo análisis en Python o R.

El método anterior está bien para realizar una exportación rápida y única. Sin embargo, si necesita editar continuamente su consulta SQL o está trabajando con múltiples consultas, esto puede volverse molesto muy rápidamente.

Un mejor flujo de trabajo para el análisis de datos

1. Conectar Python o R a una base de datos.
2. Escribir consultas SQL en Python o R.
3. Seguir haciendo análisis en Python o R.

Este segundo enfoque le permite realizar todas sus consultas y análisis en una sola herramienta, lo que resulta útil si necesita modificar sus consultas a medida que realiza el análisis. El resto de este capítulo proporciona código para cada paso de este segundo flujo de trabajo.

Conectar Python a una base de datos

Se necesitan tres pasos para conectar Python a una base de datos:

1. Instalar un controlador de base de datos para Python.
2. Configurar una conexión de base de datos en Python.
3. Escribir código SQL en Python.

Paso 1: Instalar un controlador de base de datos para Python

Un controlador de base de datos es un software que ayuda a Python a comunicarse con una base de datos, y hay muchas opciones d e controladores para elegir. La Tabla 2-2 incluye el código para instalar un controlador popular para cada RDBMS.

Esta es una instalación de una sola vez que tendrá que hacer a través de una **`instalación pip`** o una **`instalación conda`**. El siguiente código debe ejecutarse en una ventana de terminal.

Tabla 2-2. Instalar un controlador para Python usando pip o conda

RDBMS	Código	de
[redacted]	opciónSQLite	
	n/aNo necesita instalación (Python 3 viene con sqlite3)	
MySQL	pip `pip install mysql-conector-python`	
	conda `conda install -c conda-forge mysql-connector-python`	
Oracle	pip `pip install cx_Oracle`	
	conda `conda install -c conda-forge cx_oracle`	
PostgreSQL	pip `pip install psycopg2`	
	conda `conda install -c conda-forge psycopg2`	
Servidor SQL	pip `pip install pyodbc`	
	conda `conda install -c conda-forge pyodbc`	

Paso 2: Configurar una conexión de base de datos en Python

Para configurar una conexión a una base de datos, primero debe conocer la ubicación y el nombre de la base de datos a la que intenta conectarse, , así como su nombre de usuario y contraseña. Encontrará más detalles en "Campos de conexión a la base de datos" en la página 23.

La Tabla 2-3 contiene el código Python que necesitas ejecutar cada vez que planees escribir código SQL en Python. Puedes incluirlo en la parte superior de tu script Python.

Tabla 2-3. Código Python para establecer una conexión a una base de datos

RDBMSCode	SQLite
	`importar sqlite3` `conn = sqlite3.connect('mi_nueva_db.db')`

MySQL	`import mysql.conector`
	`conn = mysql.connector.connect(`
	`host='localhost', base`
	`de datos='mi_nueva_db',`
	`usuario='alice',`
	`contraseña='contraseña`
	`')`

Oracle	`# Conexión a Oracle Express Edition`
	`import cx_Oracle`
	`conn = cx_Oracle.connect(dsn='localhost/XE',`
	`usuario='alice',`
	`contraseña='contraseña')`

PostgreSQL	`import psycopg2`
	`conn = psycopg2.connect(host='localhost',`
	`database='mi_nueva_db',`
	`user='alice',`
	`password='contraseña')`

SQL Server	`# Conexión a SQL Server Express import`
	`pyodbc`
	`conn = pyodbc.connect(driver='{SQL Server}',`
	`host='localhost\SQLEXPRESS',`
	`database='mi_nueva_db',`
	`user='alice',`
	`password='contraseña')`

CONSEJO

No todos los argumentos son obligatorios. Si excluye un argumento por completo, se utilizará el valor predeterminado. En por ejemplo, el host por defecto es *localhost*, que es su ordenador. Si no se ha configurado ningún nombre de usuario ni contraseña, estos argumentos pueden omitirse.

Protección de contraseñas en Python

El código anterior está bien para probar una conexión a una base de datos, pero en realidad, no deberías guardar tu contraseña dentro de un script para que todo el mundo la vea.

Hay múltiples formas de evitarlo, entre ellas:

- generar una clave SSH
- configuración de variables de entorno
- creación de un archivo de configuración

Sin embargo, todas estas opciones requieren conocimientos adicionales de informática o formatos de archivo.

El método recomendado: crear un archivo Python independiente.

El enfoque más directo, en mi opinión, es guardar tu nombre de usuario y contraseña en un archivo Python separado, y luego llamar a ese archivo dentro de tu script de conexión a la base de datos. Aunque esto es menos seguro que las otras opciones, es el comienzo más rápido.

Para utilizar este enfoque, comience por crear un archivo *db_config.py* con el siguiente código:

```
usr = "alice"
pwd =
"password"
```

Importe el archivo db_config.*py* cuando configure la conexión a la base de datos. El siguiente ejemplo modifica el código Oracle de la Tabla 2-3 para utilizar los valores de db_config.py en lugar de los valores de usuario y contraseña codificados (los

cambios están en negrita):

```
importar cx_Oracle
importar db_config

conn = cx_Oracle.connect(dsn='localhost/XE',
       user=db_config.usr, password=db_config.pwd)
```

Paso 3: Escribir código SQL en Python

Una vez establecida la conexión con la base de datos, puedes empezar a escribir consultas SQL dentro de tu código Python.

Escriba una consulta sencilla para probar su conexión a la base de datos:

```
cursor = conn.cursor()
cursor.execute('SELECT * FROM test;')
result = cursor.fetchall()
print(result)
```

```
[(1, 100),
 (2, 200)]
```

ADVERTENCIA

Cuando utilice `cx_Oracle` en Python, elimine el punto y coma (`;`) al final de todas las consultas para evitar obtener un error.

Guarda los resultados de una consulta como un marco de datos pandas:

```
# pandas debe estar ya instalado
import pandas as pd

df = pd.read_sql('''SELECT * FROM test;''', conn)
print(df)
print(tipo(df))
```

```
    id num
    01 100
```

```
    12 200
<clase 'pandas.core.frame.DataFrame'>
```

Cierre la conexión cuando haya terminado de utilizar la base de datos:

```
cursor.close()
conn.close()
```

Siempre es una buena práctica cerrar la conexión a la base de datos para ahorrar recursos.

SQLAlchemy para los amantes de Python

Otra forma popular de conectarse a una base de datos es utilizar el paquete SQL- Alchemy en Python. Se trata de un *mapeador relacional de objetos* (ORM) que convierte los datos de la base de datos en objetos de Python, lo que permite codificar en Python puro en lugar de utilizar sintaxis SQL.

Imagina que quieres ver todos los nombres de las tablas de una base de datos. (El siguiente código es específico de PostgreSQL, pero SQLAlchemy funcionará con cualquier RDBMS).

Sin SQLAlchemy:

```
pd.read_sql("""SELECT nombre_tabla
            FROM pg_catalog.pg_tables
            WHERE schemaname='public'""", conn)
```

Con SQLAlchemy:

```
conn.nombres_tabla()
```

Cuando se utiliza SQLAlchemy, el objeto conn viene con un método Python `table_names()`, que puede resultarte más fácil de recordar que la sintaxis SQL. Aunque SQLAlchemy proporciona un código Python más limpio, ralentiza el rendimiento debido al tiempo adicional que emplea en convertir los datos en objetos Python.

Para utilizar SQLAlchemy en Python:

1. Debe tener ya instalado un controlador de base de datos (como `psycopg2`).
2. En una ventana de terminal, escribe **`pip install sqlalchemy`** o un **`conda install -c conda-forge sqlalchemy`** para instalar SQLAlchemy.
3. Ejecuta el siguiente código en Python para configurar una

conexión SQLAlchemy. (El siguiente código es específico de PostgreSQL).

La documentación de SQLAlchemy proporciona código para otras
RDBMS y controladores:

```
from sqlalchemy import crear_motor
conn = create_engine('postgresql+psycopg2://
       alice:password@localhost:5432/mi_nueva_db')
```

Conectar R a una base de datos

Se necesitan tres pasos para conectar R a una base de datos:

1. Instalar un controlador de base de datos para R
2. Establecer una conexión de base de datos en R
3. Escribir código SQL en R

Paso 1: Instalar un controlador de base de datos para R

Un controlador de base de datos es un software que ayuda a R a comunicarse con una base de datos, y hay muchas opciones de controladores para elegir. La Tabla 2-4 incluye el código para instalar un controlador popular para cada RDBMS.

Se trata de una instalación única. El siguiente código debe ejecutarse en R.

Tabla 2-4. Instalar un controlador para R

RDBMS	Code
SQLite	`install.packages("RSQLite")`
MySQL	`install.packages("RMySQL")`

Oracle	El paquete `ROracle` puede descargarse de la página de descargas de Oracle ROracle.
	`setwd("carpeta_donde_descargaste_ROracle")`
	`# Actualizar el nombre del archivo .zip en base a la última versión`
	`install.packages("ROracle_1.3-2.zip", repos=NULL)`

RDBMSCode PostgreSQL `install.packages("RPostgres")`

SQL Server En Windows, el paquete odbc (Open Database Connectivity) está preinstalado. En macOS y Linux, puede descargarse de la página ODBC de Microsoft.

```
install.packages("odbc")
```

Paso 2: Configurar una conexión de base de datos en R

Para configurar una conexión a una base de datos, primero debe conocer la ubicación y el nombre de la base de datos a la que intenta conectarse, así como su nombre de usuario y contraseña. Encontrará más detalles en "Campos de conexión a la base de datos" en la página 23.

La Tabla 2-5 contiene el código R que necesita ejecutar cada vez que planee escribir código SQL en R. Puede incluirlo en la parte superior de su script R.

Tabla 2-5. Código R para establecer una conexión a una base de datos

RDBMSCode SQLite

```
biblioteca(DBI)
con <- dbConnect(RSQLite::SQLite(),
                 "mi_nueva_db.db")
```

MySQL

```
library(RMySQL)
con <- dbConnect(RMySQL::MySQL(),
                 host="localhost",
                 dbname="mi_nueva_db",
                 user="alice",
                 password="contraseña")
```

Oracle

```
biblioteca(ROracle)
drv <- dbDriver("Oracle")
con <- dbConnect(drv, "alice", "password",
                 dbname="mi_nueva_db")
```

RDBMSCode Biblioteca PostgreSQL(RPostgres)

```
con <- dbConnect(RPostgres::Postgres(),
                 host="localhost",
                 dbname="mi_nueva_db",
                 user="alice",
                 password="contraseña")
```

Biblioteca SQL Server (DBI)

```
con <- DBI::dbConnect(odbc::odbc(),
                      Driver="SQL Server",
                      Server="localhost\\SQLEXPRESS",
                      Database="mi_nueva_db",
                      User="alice",
                      Password="contraseña",
                      Trusted_Connection="True")
```

CONSEJO

No todos los argumentos son obligatorios. Si excluye un argumento por completo, se utilizará el valor predeterminado.

- Por ejemplo, el host por defecto es *localhost*, que es tu ordenador.
- Si no se ha configurado ningún nombre de usuario ni contraseña, estos argumentos pueden omitirse.

Proteja sus contraseñas con R

El código anterior está bien para probar una conexión a una base de datos, pero en realidad, no deberías guardar tu contraseña en un script para que todo el mundo la vea.

Hay múltiples formas de evitarlo, entre ellas:

- cifrado de credenciales con el paquete de `llaveros`

- crear un archivo de configuración con el paquete `config`
- configuración de variables de entorno con un archivo *.renviron*
- registrar el usuario y la contraseña como una opción global en R con el comando `options`

El enfoque recomendado: solicitar al usuario una contraseña.

En mi opinión, lo más sencillo es que RStudio te pida la contraseña.

En lugar de esto:

```
con <- dbConnect(...,
    password="contraseña",
    ...)
```

Haz esto:

```
install.packages("rstudioapi")
con <- dbConnect(...,
    password=rstudioapi::askForPassword("¿Contraseña?"),
    ...)
```

Paso 3: Escribir código SQL en R

Una vez establecida la conexión con la base de datos, puede empezar a escribir consultas SQL dentro de su código R.

Mostrar todas las tablas de la base de datos:

```
dbListTables(con)
```

```
[1] "test"
```

CONSEJO

Para *SQL Server*, incluya el nombre del esquema para limitar el número de tablas mostradas-dbListTables(`con, schema="dbo"`). `dbo` significa propietario de la base de datos y es el esquema por defecto en SQL Server.

Eche un vistazo a la tabla de prueba de la base de datos:

```
dbReadTable(con, "test")
```

```
  id num
 1 1 100
2  2 200
```

NOTA

En *Oracle*, el nombre de la tabla distingue entre mayúsculas y minúsculas. Dado que Oracle convierte automáticamente los nombres de tabla a mayúsculas, es probable que tenga que ejecutar lo siguiente en su lugar: `dbRead Table(con, "TEST")`.

Escriba una consulta sencilla y obtenga un marco de datos:

```
df <- dbGetQuery(con, "SELECT * FROM prueba
                       WHERE id = 2")
print(df); class(df)
```

```
  id num
1 2 200
[1] "data.frame"
```

Cierre la conexión cuando haya terminado de utilizar la base de datos.

```
dbDisconnect(con)
```

Siempre es una buena práctica cerrar la conexión a la base de datos para ahorrar recursos.

CAPÍTULO 3

El lenguaje SQL

Este capítulo cubre los fundamentos de SQL, incluyendo sus estándares, términos clave y sublenguajes, junto con las respuestas a las siguientes preguntas:

- ¿Qué es ANSI SQL y en qué se diferencia de SQL?
- ¿Qué es una palabra clave frente a una cláusula?
- ¿Importan las mayúsculas y los espacios en blanco?
- ¿Qué hay más allá de la sentencia `SELECT`?

Comparación con otras lenguas

Algunas personas del ámbito tecnológico no consideran que SQL sea un verdadero lenguaje de programación.

Aunque SQL son las siglas de "Structured Query *Language" (lenguaje de* consulta estructurado), no se puede utilizar del mismo modo que otros lenguajes de programación populares como Python, Java o C++. Con estos lenguajes, puedes escribir código para especificar los pasos exactos que debe seguir un ordenador para realizar una tarea. Esto se llama *programación imperativa*.

En Python, si quieres sumar una lista de valores, puedes decirle a exactamente *cómo* quieres h a c e r l o . Lo siguiente

recorre una lista, elemento por elemento, y añade cada valor a un total para calcular finalmente la suma total:

```
calorías = [90, 240, 165]
total = 0
para c en calorías:
    total += c
print(total)
```

Con SQL, en lugar de decirle a un ordenador exactamente cómo quieres hacer algo, te limitas a describir *lo que* quieres que haga, que en este caso es calcular la suma. Entre bastidores, SQL calcula cómo ejecutar el código de forma óptima. Esto se llama *declarative programming*.

```
SELECT SUMA(calorías)
DE los entrenamientos;
```

Lo más importante es que SQL no es un *lenguaje de programación de uso general* como Python, Java o C++, que pueden utilizarse para una gran variedad de aplicaciones. Por el contrario, SQL es un *lenguaje de programación especial*, creado específicamente para gestionar datos en una base de datos relacional.

Extensiones para SQL

En esencia, SQL es un lenguaje declarativo, pero existen extensiones que le permiten hacer más cosas:

- *Oracle* dispone del *lenguaje de procedimiento SQL* (PL/SQL)
- *SQL Server* dispone de *SQL transaccional* (T-SQL)

Con estas extensiones, puedes hacer cosas como agrupar el código SQL de en procedimientos y funciones, y mucho más. La sintaxis no sigue los estándares ANSI, pero hace que SQL sea mucho más potente.

Normas ANSI

El *Instituto Nacional Estadounidense de Normalización* (ANSI) es una organización con sede en Estados Unidos que elabora normas sobre todo tipo de temas, desde el agua potable hasta las tuercas y tornillos.

SQL se convirtió en norma ANSI en 1986. En 1989, publicaron un documento muy detallado de especificaciones (cientos de páginas) sobre lo que un lenguaje de bases de datos debía ser capaz de hacer y cómo debía hacerlo. Cada pocos años, las normas se actualizan, por eso oirás términos como ANSI-89 y ANSI-92, que eran diferentes conjuntos de normas SQL que se añadieron en 1989 y 1992, respectivamente. La norma más reciente es ANSI SQL2016.

SQL Versus ANSI SQL Versus MySQL Versus ...

SQL es el término general para referirse al lenguaje de consulta estructurado.

ANSI SQL se refiere al código SQL que sigue los estándares ANSI y que se ejecutará en cualquier software de sistema de gestión de bases de datos relacionales (RDBMS).

MySQL es una de las muchas opciones RDBMS. En MySQL, puede escribir tanto código ANSI como código SQL específico de MySQL.

Otras opciones de RDBMS incluyen *Oracle*, *PostgreSQL*, *SQL Server*, *SQLite* y otros.

Incluso con los estándares, no hay dos RDBMS exactamente iguales. Aunque algunos pretenden ser totalmente compatibles con ANSI, todos lo son sólo parcialmente. Cada proveedor acaba eligiendo qué estándares implantar y qué características adicionales crear que sólo funcionen dentro de su software.

¿Debo seguir las normas?

La mayor parte del código SQL básico que escribes se adhiere a las normas ANSI ards. Si encuentra código que hace algo complejo utilizando palabras clave sencillas pero desconocidas, es muy probable que esté fuera de los estándares.

Si trabaja únicamente con un RDBMS, como *Oracle* o *SQL Server, no hay ningún* problema en no seguir las normas ANSI y aprovechar todas las funciones del software.

El problema surge cuando se tiene código funcionando en un RDBMS que se quiere utilizar en otro RDBMS. Es probable que el código que no sea ANSI no funcione en el nuevo RDBMS y haya que reescribirlo.

Supongamos que tiene la siguiente consulta que funciona en *Oracle*. No cumple las normas ANSI porque la función `DECODE` sólo está disponible en *Oracle* y no en otro software. Si copio la consulta en *SQL Server,* el código no se ejecutará:

```
-- Código específico de Oracle
SELECT item, DECODE (flag, 0, 'No', 1, 'Yes')
               AS Sí_o_No
DE artículos;
```

La siguiente consulta tiene la misma lógica, pero en su lugar utiliza un estado CASE- ment, que es un estándar ANSI. Por ello, funcionará en *Oracle*, *SQL Server* y otros programas:

```
-- Código que funciona en cualquier RDBMS
SELECT item, CASE WHEN flag = 0 THEN 'No'
                  ELSE 'Sí' END AS Sí_o_No
DE artículos;
```

¿Qué norma elegir?

Los dos bloques de código siguientes realizan una unión utilizando dos estándares diferentes. ANSI-89 fue la primera norma ampliamente adoptada, seguida de ANSI-92, que incluyó algunas revisiones importantes.

```
-- ANSI-89
SELECT c.id, c.name, o.date
FROM cliente c, pedido o
WHERE c.id = o.id;

-- ANSI-92
SELECT c.id, c.name, o.date
FROM cliente c INNER JOIN pedido
o ON c.id = o.id;
```

Si estás escribiendo código SQL nuevo, te recomiendo que utilices el estándar más reciente (que actualmente es ANSI SQL2016) o la sintaxis proporcionada en la documentación del RDBMS en el que estés trabajando.

Sin embargo, es importante conocer las normas anteriores, ya que es probable que se encuentre con código más antiguo si su empresa tiene varias décadas de existencia.

Términos SQL

Este es un bloque de código SQL que muestra el número de ventas que cada empleado cerró en 2021. Utilizaremos este bloque de código para resaltar una serie de términos SQL.

```
-- Ventas cerradas en 2021
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id =
s.emp_id WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

Palabras clave y funciones

Las palabras clave y las funciones son términos integrados en SQL.

Palabras clave

Una palabra *clave* es un texto que ya tiene algún significado en SQL. Todas las palabras clave en el bloque de código están en negrita aquí:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id = s.emp_id
WHERE AÑO(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

> **SQL distingue entre mayúsculas y minúsculas**
>
> Las palabras clave suelen escribirse en mayúsculas para facilitar la lectura. Sin embargo, SQL no distingue entre mayúsculas y minúsculas, lo que significa que un `WHERE` en mayúsculas y un `where` en minúsculas significan lo mismo cuando se ejecuta el código.

Funciones

Una *función es un* tipo especial de palabra clave. Toma cero o más entradas, hace algo con las entradas y devuelve una salida. En SQL, una función suele ir seguida de paréntesis, pero no siempre. Las dos funciones en el bloque de código están en negrita aquí:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id = s.emp_id
WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

Existen cuatro categorías de funciones: numéricas, cadena, fecha-hora y otras:

- `COUNT()` es una función numérica. Toma una columna y devuelve el número de filas no nulas (filas que tienen un valor).
- `YEAR()` es una función de fecha. Toma una columna de un tipo de datos fecha o fecha-hora, extrae los años y devuelve los val- ues como una nueva columna.

En la Tabla 7-2 encontrará una lista de las funciones más comunes.

Identificadores y alias

Los identificadores y alias son términos que define el usuario.

Identificadores

Un *identificador* es el nombre de un objeto de la base de datos, como una tabla o una columna. Todos los identificadores del bloque de código aparecen en negrita:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id =
s.emp_id WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

Los identificadores deben empezar por una letra (`a-z` o `A-Z`), seguida de cualquier combinación de letras, números y guiones bajos (_). Algunos programas permiten caracteres adicionales como `@`, `#` y `$`.

Para facilitar la lectura, los identificadores suelen ir en minúsculas, mientras que las palabras clave van en mayúsculas, aunque el código se ejecutará con independencia de mayúsculas y minúsculas.

CONSEJO

Como práctica recomendada, los identificadores no deben tener el mismo nombre que una palabra clave existente. Por ejemplo, no querrás nombrar una columna `COUNT` porque ya es una palabra clave en SQL.

Si aún así decide hacerlo, puede evitar confusiones encerrando el identificador entre comillas dobles en . Así, en lugar de llamar a una columna `CONTAR`, puede llamarla `"CONTAR"`, pero es mejor utilizar un nombre completamente distinto, como `num_ventas`.

MySQL utiliza puntos suspensivos (``) para encerrar identificadores en lugar de comillas dobles (`""`).

Alias

Un *alias* cambia el nombre de una columna o una tabla temporalmente, sólo durante el tiempo que dure la consulta en . En otras palabras, los nuevos nombres de alias aparecerán en los resultados de la consulta, pero los nombres originales de las columnas permanecerán inalterados en las tablas desde las que se realiza la consulta. Todos los alias del bloque de código aparecen en negrita:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id =
s.emp_id WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

La norma es utilizar `AS` cuando se renombran columnas (`AS num_ventas`) y ningún texto adicional cuando se renombran tablas (`e`). Sin embargo, técnicamente, cualquiera de las dos sintaxis sirve tanto para columnas como para tablas.

Además de las columnas y tablas, los alias también son útiles si desea asignar un nombre temporal a una subconsulta.

Declaraciones y cláusulas

Son formas de referirse a subconjuntos de código SQL.

Declaraciones

Una *sentencia* comienza con una palabra clave y termina con un punto y coma. Todo este bloque de código se denomina sentencia `SELECT` porque empieza por la palabra clave `SELECT`.

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id =
s.emp_id WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

CONSEJO

Muchas herramientas de bases de datos que proporcionan una interfaz gráfica de usuario no requieren el punto y coma (`;`) al final de una sentencia.

La sentencia `SELECT` es el tipo de sentencia SQL más popular, y a menudo se denomina consulta porque busca datos en una base de datos. Otros tipos de sentencias se tratan en "Sublan- guajes" en la página 50.

Cláusulas

Una *cláusula* es una forma de referirse a una sección concreta de una sentencia. Esta es nuestra sentencia `SELECT` original:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id =
s.emp_id WHERE YEAR(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

Esta declaración contiene cuatro cláusulas principales:

- Cláusula SELECT
  ```
  SELECT e.name, COUNT(s.sale_id) AS num_sales
  ```
- Cláusula FROM
  ```
  FROM empleado e
    LEFT JOIN ventas s ON e.emp_id = s.emp_id
  ```
- Cláusula WHERE
  ```
  WHERE YEAR(s.fecha_venta) =
    2021 AND s.cerrado IS NOT
    NULL
  ```
- Cláusula GROUP BY
  ```
  GROUP BY e.name;
  ```

En una conversación, a menudo oirás a la gente referirse a una sección de una sentencia como "echa un vistazo a las tablas de la cláusula FROM". Es una forma útil de ampliar una sección concreta del código.

NOTA

En realidad, esta frase tiene más cláusulas que las cuatro enumeradas. En gramática, una cláusula es una parte de una frase que contiene un sujeto y un verbo. Así que podrías referirte a lo siguiente:

```
LEFT JOIN ventas s ON e.emp_id = s.emp_id
```

como la cláusula LEFT JOIN si desea especificar aún más la sección del código a la que se refiere.

Las seis cláusulas más populares empiezan por SELECT, FROM, WHERE, GROUP BY, HAVING y ORDER BY, y se tratan en detalle en el Capítulo 4.

Expresiones y predicados

Se trata de combinaciones de funciones, identificadores, etc.

Expresiones

Una *expresión* puede considerarse como una fórmula que da como resultado un valor . Una expresión en el bloque de código era:

```
COUNT(s.venta_id)
```

Esta expresión incluye una función (`COUNT`) y un identificador (`s.sale_id`). Juntos, forman una expresión que dice contar el número de ventas.

Otros ejemplos de expresiones son:

- `s.sale_id + 10` es una expresión numérica que incorpora operaciones matemáticas básicas.
- `CURRENT_DATE` es una expresión datetime, simplemente una función , que devuelve la fecha actual.

Predicados

Un *predicado* es una comparación lógica que da como resultado uno de estos tres valores `VERDADERO`/`FALSO`/`DESCONOCIDO`. A veces se denominan *sentencias condicionales*. Los tres predicados del bloque de código están en negrita:

```
SELECT e.name, COUNT(s.sale_id) AS num_sales
FROM empleado e
  LEFT JOIN ventas s ON e.emp_id = s.emp_id
WHERE AÑO(s.fecha_venta) = 2021
  AND s.closed IS NOT NULL
GROUP BY e.name;
```

Algunas cosas que notarás en estos ejemplos son:

- El signo igual (=) es el operador más popular para compare valores.

- `NULL` significa sin valor. Cuando se comprueba si un campo no tiene valor, en lugar de escribir `= NULL`, se escribiría `IS NULL`.

Comentarios, citas y espacios en blanco

Son signos de puntuación con significado en SQL.

Comentarios

Un *comentario* es un texto que se ignora cuando se ejecuta el código, como el siguiente .

```
-- Ventas cerradas en 2021
```

Es útil incluir comentarios en tu código para que otros revisores del mismo (¡incluido tu futuro yo!) puedan entender rápidamente la intención del código sin leerlo todo.

Para comentar:

- Una sola línea de texto:

```
-- Estos son mis comentarios
```

- Varias líneas de texto:

```
/* Estos son
mis
comentarios */
```

Citas

Hay dos tipos de comillas que se pueden utilizar en SQL, la comilla simple y la comilla doble.

```
SELECT "Esta columna"
FROM mi_tabla
WHERE nombre = 'Bob';
```

Citas sueltas: Cuerdas

Eche un vistazo a `'Bob'`. Las comillas simples se utilizan cuando se hace referencia a un valor de cadena. En la práctica verás muchas más comillas simples que dobles.

Comillas dobles: Identificadores

Eche un vistazo a `"Esta columna"`. Las comillas dobles se utilizan cuando se hace referencia a un identificador. En este caso, como hay un espacio entre `This` y `column`, las comillas dobles son necesarias para `que This column se` interprete como un nombre de columna. Sin las comillas dobles, SQL arrojaría un error debido al espacio. Dicho esto, es una buena práctica utilizar `_` en lugar de espacios al nombrar columnas para evitar el uso de las comillas dobles.

NOTA

MySQL utiliza puntos suspensivos (``) para encerrar identificadores en lugar de comillas dobles (`""`).

Espacio en blanco

A SQL no le importa el número de espacios entre términos. Ya sea un espacio, un tabulador o una nueva línea, SQL ejecutará la consulta desde la primera palabra clave hasta el punto y coma al final de la sentencia. Las dos consultas siguientes son equivalentes.

```
SELECT * FROM mi_tabla;

SELECT *
  FROM mi_tabla;
```

NOTA

En el caso de consultas SQL sencillas, es posible que veas todo el código escrito en una sola línea. En el caso de consultas más largas, con docenas o incluso cientos de líneas, verás nuevas líneas para nuevas cláusulas, pestañas al enumerar muchas columnas o tablas, etc.

El objetivo final es tener un código legible, por lo que tendrás que decidir cómo quieres espaciar el código (o seguir las directrices de tu empresa) para que tenga un aspecto limpio y se pueda hojear rápidamente.

Sublenguas

Existen muchos tipos de sentencias que pueden escribirse en SQL. Todos ellos caen bajo uno de los cinco sublenguajes, que se detallan en la Tabla 3-1.

Tabla 3-1. Sublenguajes SQL

Sublengua	Descripción	Común	Comando
Lenguaje de consulta de datos (DQL)	Este es el lenguaje con el que la mayoría de la gente está familiarizada . Estas sentencias se utilizan para recuperar información de un objeto de base de datos, como una tabla, y suelen denominarse consultas SQL.	SELECT	La mayor parte de este libro está dedicada a DQL
Lenguaje de definición de datos (DDL)	manipulación de datos (DML)	Es el lenguaje utilizado para definir o	crear un objeto de base de datos, como una tabla o un índice. Es el lenguaje utilizado para manipular o modificar los datos de una base de
Lenguaje de			

datos.

CREAR ALTERAR
SOLTAR

INSERTAR
ACTUALIZAR
SUPRIMIR

Crear, actualizar y borrar

Crear, actua

Sublengua	Descripción	Común	Comando
Lenguaje de control de datos (DCL)	Es el lenguaje utilizado para controlar el acceso a los datos de una base de datos, lo que a veces se denomina permisos o privilegios.	REVOCACIÓN DE LA SUBVENCIÓN	No cubierto
Lenguaje de control de transacciones (TCL)	Es el lenguaje utilizado para gestionar transacciones en una base de datos, o aplicar cambios permanentes a una base de datos.	COMETER RETROCESO	Gestión de transacciones

Aunque la mayoría de los analistas y científicos de datos escribirán sentencias DQL `SELECT` para consultar tablas, es importante saber que los administradores de bases de datos y los ingenieros de datos también escribirán código en estos otros sublenguajes para mantener una base de datos.

Resumen del lenguaje SQL

- ANSI SQL es un código SQL estandarizado que funciona en todos los programas de bases de datos. Muchos RDBMS tienen extensiones que no cumplen los estándares pero añaden funcionalidad a su software .
- Las palabras clave son términos reservados en SQL que tienen un significado especial.
- Las cláusulas se refieren a secciones concretas de una sentencia. Las cláusulas más comunes son `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `HAVING` y `ORDER BY`.
- Las mayúsculas y los espacios en blanco no importan en SQL para la ejecución de , pero hay mejores prácticas para la legibilidad.
- Además de las sentencias `SELECT`, existen comandos para definir objetos, manipular datos, etc.

CAPÍTULO 4

Conceptos básicos de consulta

Una *consulta* es un apodo para una sentencia SELECT, que consta de seis cláusulas principales. Cada sección de este capítulo cubre una cláusula en detalle:

1. SELECCIONE
2. DESDE
3. DONDE
4. GRUPO POR
5. TENIENDO
6. ORDENAR POR

La última sección de este capítulo cubre la cláusula LIMIT, soportada por *MySQL*, *PostgreSQL* y *SQLite*.

Los ejemplos de código de este capítulo hacen referencia a cuatro tablas:

`cascada`
: Cascadas de la Península Superior de Michigan

`propietario`
: propietarios de las cascadas

condado
: comarcas donde se encuentran las cascadas

visita
: recorridos con múltiples paradas en cascadas

A continuación se presenta un ejemplo de consulta que utiliza las seis cláusulas principales. Le siguen los resultados de la consulta, que también se conocen como *conjunto de resultados*.

```
-- Recorridos con 2 o más cascadas públicas
SELECT   t.name AS tour_name,
         COUNT(*) AS num_waterfalls
FROM     tour t LEFT JOIN cascada w
         ON t.parada = w.id
DONDE    w.open_to_public = 'y'
GROUP BY t.name
HAVING   CONTAR(*) >= 2
ORDER BY tour_name;

tour_name numero_cascadas
---------- ---------------
M-28                     6
Munising                 6
US-2                     4
```

Consultar una base de datos significa obtener datos de una base de datos, normalmente de una tabla o varias tablas.

NOTA

También es posible consultar una *vista* en lugar de una tabla. Las vistas se parecen a las tablas y derivan de ellas, pero no contienen datos. Encontrará más información sobre las vistas en "Vistas", en la página 133 del Capítulo 5.

La cláusula SELECT

La cláusula SELECT especifica las columnas que desea que devuelva un estado.

En la cláusula SELECT, la palabra clave SELECT va seguida de una lista de nombres de columnas y/o expresiones separadas por com- mas. Cada nombre de columna y/o expresión se convierte entonces en una columna de los resultados.

Selección de columnas

La cláusula SELECT más sencilla enumera uno o más nombres de columnas de las tablas de la cláusula FROM:

```
SELECT id, nombre
DEL propietario;

identificador   nombre
----- ---------------
    1 Rocas Pictured
    2 Naturaleza de Michigan
    3 AF LLC
    4 MI DNR
    5 Cataratas Horseshoe
```

Seleccionar todas las columnas

Para devolver todas las columnas de una tabla, puede utilizar un único asterisco en lugar de escribir el nombre de cada columna:

```
SELECCIONAR *
DEL propietario;

identificador           nombre         teléfono     tipo
----- ---------------- -------------- --------
    1 Pictured Rocks   906.387.2607  público
    2 Naturaleza de    517.655.5655   privado
    Michigan
    3 AF LLC                          privado
    4 MI DNR           906.228.6561  público
    5 Cataratas        906.387.2635   privado
    Horseshoe
```

ADVERTENCIA

El asterisco es un atajo útil cuando se prueban consultas, ya que puede ahorrarle bastante tiempo de escritura. Sin embargo, es arriesgado utilizar el asterisco en código de producción porque las columnas de una tabla pueden cambiar con el tiempo, haciendo que tu código falle cuando haya menos o más columnas de las esperadas.

Selección de expresiones

Además de enumerar simplemente las columnas, también puede enumerar expresiones más complejas dentro de la cláusula `SELECT` para que aparezcan como columnas en los resultados.

El siguiente enunciado incluye una expresión para calcular un descenso del 10% de la población, redondeado a cero decimales:

```
SELECT nombre, ROUND(población * 0,9, 0)
DESDE el condado;

nombreROUND(población * 0,9, 0)
---------- --------------------------
Alger                                8876
Baraga                               7871
Ontonagon                            7036
...
```

Selección de funciones

Las expresiones de la lista `SELECT` suelen hacer referencia a columnas de las tablas de las que se está extrayendo información, pero hay excepciones. Por ejemplo, una función común que no hace referencia a ninguna tabla es la que devuelve la fecha actual:

```
SELECT FECHA_ACTUAL;

FECHA_ACTUAL
------------
2021-12-01
```

El código anterior funciona en *MySQL*, *PostgreSQL* y *SQLite*. El código equivalente que funciona en otros RDBMSs puede encontrarse en "Funciones Datetime" en la página 218 del Capítulo 7.

NOTA

La mayoría de las consultas incluyen una cláusula `SELECT` y una cláusula `FROM`, pero sólo es necesaria la cláusula `SELECT` cuando se utilizan funciones concretas de la base de datos, como `CURRENT_DATE`.

También es posible incluir expresiones dentro de la cláusula `SELECT` que sean *subconsultas* (una consulta anidada dentro de otra consulta). Encontrará más detalles en "Selección de subconsultas" en la página 61.

Aliasing Columns

El propósito de un *alias de columna* es dar un nombre temporal a cualquier columna o expresión listada en la cláusula `SELECT`. Ese nombre temporal, o alias de columna, se muestra como nombre de columna en los resultados.

Tenga en cuenta que no se trata de un cambio de nombre permanente, ya que los nombres de las columnas de las tablas originales no cambian. El alias sólo existe dentro de la consulta.

Este código muestra tres columnas.

```
SELECCIONE id, nombre,
       ROUND(población * 0,9, 0)
DESDE el condado;

id    nombreROUND(población * 0,9, 0)
----- ---------- --------------------------
    2 Alger                            8876
    6 Baraga                           7871
    7 Ontonagon                        7036
...
```

Supongamos que queremos renombrar los nombres de las columnas de los resultados. `id` es demasiado ambiguo y nos gustaría darle un nombre más descriptivo. `ROUND(population * 0.9, 0)` es demasiado largo y nos gustaría darle un nombre más sencillo.

Para crear un alias de columna, a continuación del nombre o la expresión de una columna, debe aparecer (1) un nombre de alias o (2) la palabra clave `AS` y un nombre de alias.

```
-- alias_name
SELECT id county_id, nombre,
       ROUND(población * 0,9, 0) población_estimada
DESDE el condado;
```

o:

```
-- AS alias_name
SELECT id AS county_id, name,
       ROUND(población * 0,90, 0) AS población_estimada
DESDE el condado;

county_id nombre      población_estimada
---------- ---------- -------------
         2 Alger                 8876
         6 Baraga                7871
         7 Ontonagon             7036
...
```

Ambas opciones se utilizan en la práctica al crear alias. Dentro de la cláusula `SELECT`, la segunda opción es más popular porque la palabra clave `AS` facilita visualmente la diferenciación de nombres de columnas y alias entre una larga lista de nombres de columnas.

NOTA

Las versiones antiguas de *PostgreSQL* requieren el uso de `AS` al crear un alias de columna.

Aunque los alias de columna no son obligatorios, son muy recomendables cuando se trabaja con expresiones para dar nombres razonables a las columnas de los resultados.

Alias con distinción entre mayúsculas y minúsculas y puntuación

Como puede verse con los alias de columna `county_id` y `estimated_pop`, la convención es utilizar letras minúsculas con guiones bajos en lugar de espacios al nombrar los alias de columna.

También puede crear alias que contengan letras mayúsculas, espacios y signos de puntuación utilizando la sintaxis de comillas dobles, como se muestra en este ejemplo:

```
SELECT id AS "Waterfall #",
  name AS "Waterfall Name"
DESDE la cascada;

Cascada # Nombre de la cascada
----------- --------------
          1 Munising Falls
          2 Cascadas Tannery
          3 Cascadas de Alger
...
```

Columnas de calificación

Supongamos que escribes una consulta que extrae datos de dos tablas y ambas contienen una columna llamada `nombre`. Si sólo incluyeras `nombre` en la cláusula `SELECT`, el código no sabría a qué tabla te estás refiriendo.

Para resolver este problema, puede *calificar* un nombre de columna por su nombre de tabla . En otras palabras, puedes dar a una columna un prefijo para especificar a qué tabla pertenece utilizando *la notación por puntos*, como en `nombre_tabla.nombre_columna`.

El siguiente ejemplo consulta una única tabla, por lo que, aunque no es necesario calificar las columnas, se muestra a modo de demostración. Así es como se califica una columna por su nombre de tabla:

```
SELECT propietario.id, propietario.nombre
DEL propietario;
```

CONSEJO

Si se produce un error en SQL al hacer referencia a un *nombre de columna ambiguo*, significa que varias tablas de la consulta tienen una columna con el mismo nombre y que no se ha especificado en a qué combinación de tabla y columna se hace referencia. Puede resolver el error calificando el nombre de la columna.

Tablas de clasificación

Si califica un nombre de columna por su nombre de tabla, también puede calificar ese nombre de tabla por su nombre de base de datos o esquema. La siguiente consulta recupera datos específicamente de la tabla `owner` dentro del esquema `sqlbook`:

```
SELECT sqlbook.owner.id, sqlbook.owner.name
FROM sqlbook.owner;
```

El código anterior es largo ya que `sqlbook.owner` se repite varias veces. Para ahorrar tiempo, puede proporcionar un *alias de tabla*. El siguiente ejemplo da el alias `o` a la tabla `owner`:

```
SELECT o.id, o.name
FROM sqlbook.owner o;
```

o:

```
SELECT o.id, o.name
FROM propietario o;
```

Alias de columna frente a alias de tabla

Los alias de columna se definen dentro de la cláusula `SELECT` para renombrar una columna en los resultados. Es habitual incluir `AS`, aunque no es obligatorio.

```
-- Alias de columna
SELECT num AS nuevo_col
FROM mi_tabla;
```

Los alias de tabla se definen dentro de la cláusula `FROM` para crear un apodo temporal para una tabla. Es habitual excluir `AS`, aunque incluir `AS` también funciona.

```
-- Alias de tabla
SELECT *
FROM mi_tabla mt;
```

Selección de subconsultas

Una *subconsulta* es una consulta anidada dentro de otra consulta. Las subconsultas pueden ubicarse dentro de varias cláusulas, incluida la cláusula `SELECT de` .

En el siguiente ejemplo, además `del id`, el `nombre` y la `población`, digamos que también queremos ver la población media de todos los condados. Al incluir una subconsulta, estamos creando una nueva columna en los resultados para la población media.

```
SELECT id, nombre, población,
       (SELECT AVG(población) FROM condado)
       AS average_pop
DESDE el condado;

id               nombrepoblación población_media
----- ---------- ----------- ------------
    2 Alger             9862        18298
    6 Baraga            8746        18298
    7 Ontonagon         7818        18298
...
```

Hay que tener en cuenta algunas cosas:

- Una subconsulta debe ir rodeada de paréntesis.
- Al escribir una subconsulta dentro de la cláusula `SELECT`, es muy recomendable especificar un alias de columna,

que en este caso era `average_pop`. De esta forma, la columna tiene un nombre sencillo en los resultados.

- Sólo hay un valor en la columna `población_media` que se repite en todas las filas. Cuando se incluye una subconsulta dentro de la cláusula `SELECT`, el resultado de la subconsulta debe devolver una única columna y cero o una fila, como se muestra en la siguiente subconsulta para calcular la población media.

```
SELECT AVG(población) FROM condado;

 AVG(población)
 ---------------
          18298
```

- Si la subconsulta devolviera cero filas, la nueva columna se rellenaría con valores `NULL`.

Subconsultas no correlacionadas frente a subconsultas correlacionadas

El ejemplo anterior es una subconsulta *no correlacionada*, lo que significa que la subconsulta no hace referencia a la consulta externa. La subconsulta puede ejecutarse por sí sola independientemente de la consulta externa.

El otro tipo de subconsulta se denomina *subconsulta correlacionada*, , que hace referencia a valores de la consulta externa. Esto a menudo ralentiza significativamente el tiempo de procesamiento, por lo que es mejor reescribir la consulta utilizando un `JOIN` en su lugar. A continuación se muestra un ejemplo de subconsulta correlacionada junto con un código más eficiente.

Problemas de rendimiento con subconsultas correlacionadas

La siguiente consulta devuelve el número de cascadas de cada propietario de . Observe que el paso `o.id = w.owner_id` de la subconsulta hace referencia a la tabla `owner` de la consulta

externa, lo que la convierte en una subconsulta correlacionada.

```
    SELECT o.id, o.name,
           (SELECT COUNT(*) FROM cascada w
           WHERE o.id = w.owner_id) AS num_waterfalls
    DEL propietario o;

identificador           nombre      número_cascadas
       ----- ---------------- --------------
          1 Pictured Rocks                 3
          2 Naturaleza de Michigan         3
          3 AF LLC                         1
          4 MI DNR                         1
          5 Cataratas Horseshoe            0
```

Un enfoque mejor sería reescribir la consulta con un `JOIN`. De esta forma, primero se unen las tablas y luego se ejecuta el resto de la consulta, lo que es mucho más rápido que volver a ejecutar una subconsulta para cada fila de datos. Encontrará más información sobre las uniones en "Unir tablas" en la página 270 del Capítulo 9.

```
    SELECTo  .id, o.name,
             COUNT(w.id) AS num_waterfalls
    FROM     owner o LEFT JOIN cascadas w
             ON o.id = w.owner_id
    GROUP BY o.id, o.name

identificador           nombre      número_cascadas
       ----- ---------------- --------------
          1 Pictured Rocks                 3
          2 Naturaleza de Michigan         3
          3 AF LLC                         1
          4 MI DNR                         1
          5 Cataratas Horseshoe            0
```

DISTINTO

Cuando se enumera una columna en la cláusula `SELECT`, por defecto se devuelven todas las filas. Para ser más explícito, puede incluir la palabra clave `ALL`, pero es puramente opcional. Las siguientes consultas devuelven cada combinación `tipo`/`open_to_public`.

```
SELECT o.type, w.open_to_public
FROM owner o
JOIN cascada w ON o.id = w.owner_id;
```

o:

```
SELECT ALL o.type, w.open_to_public
FROM owner o
JOIN cascada w ON o.id = w.owner_id;

tipo     open_to_public
-------- --------------
público  y
público  y
público  y
privado  y
privado  y
privado  y
privado  y
público y
```

Si desea eliminar filas duplicadas de los resultados, puede utilizar la palabra clave `DISTINCT`. La siguiente consulta devuelve una lista de combinaciones únicas de `tipo`/`open_to_public`.

```
SELECT DISTINCT o.type, w.open_to_public
FROM owner o
JOIN cascada w ON o.id = w.owner_id;

tipo     open_to_public
-------- --------------
public   y
privado y
```

COUNT y DISTINCT

Para contar el número de valores únicos dentro de una *única columna*, puede combinar las palabras clave `COUNT` y `DISTINCT` dentro de la cláusula `SELECT de` . La siguiente consulta devuelve el número de valores únicos de `tipo`.

```
SELECT COUNT(DISTINCT type) AS unique
FROM owner;

 único
-------
     2
```

Para contar el número de combinaciones únicas de *varias columnas*, puede envolver una consulta `DISTINCT` como una subconsulta y, a continuación, realizar un `COUNT` en la subconsulta. La siguiente consulta devuelve el número de combinaciones únicas de `tipo/open_to_public`.

```
SELECT COUNT(*) AS num_unique
FROM (SELECT DISTINCT o.type, w.open_to_public
      FROM owner o JOIN waterfall w
      ON o.id = w.owner_id) my_subquery;

 num_único
-----------
         2
```

MySQL y *PostgreSQL* admiten el uso de la sintaxis `COUNT(DISTINCT)` en varias columnas. Las dos consultas siguientes son equivalentes a la consulta anterior, sin necesidad de una subconsulta:

```
-- Equivalente de MySQL
SELECT COUNT(DISTINCT o.type, w.open_to_public)
       AS num_unique
       FROM propietario o JOIN
            cascada w ON o.id =
            w.owner_id;

-- Equivalente de PostgreSQL
SELECT COUNT(DISTINCT (o.type, w.open_to_public))
       AS num_unique
       FROM propietario o JOIN
            cascada w ON o.id =
            w.owner_id;

 num_único
-----------
```

2

La cláusula FROM

La cláusula `FROM` se utiliza para especificar la fuente de los datos que desea recuperar . El caso más sencillo es nombrar una única tabla o vista en la cláusula `FROM` de la consulta.

```
SELECT nombre
FROM cascada;
```

Puede calificar una tabla o una vista con un nombre de base de datos o de esquema utilizando la notación de puntos. La siguiente consulta recupera datos específicamente de la tabla `waterfall` dentro del esquema `sqlbook`:

```
SELECCIONAR nombre
FROM sqlbook.waterfall;
```

A partir de varias tablas

En lugar de recuperar datos de una tabla, a menudo querrá reunir datos de varias tablas. La forma más común de hacerlo es utilizando una cláusula `JOIN` dentro de la cláusula `FROM`. La siguiente consulta recupera datos de las tablas `waterfall` y `tour` y muestra una única tabla de resultados.

```
SELECCIONAR *
FROM cascada w JOIN gira t
     ON w.id = t.parada;

id    nombre           ... nombre    para...
----- --------------      ---------  -----
    1 Cataratas           M-28           1
      Munising
    1 Cataratas           Munising       1
      Munising
    2 Cascadas            Munising       2
      Tannery
    3 Cataratas de        M-28           3
      Alger
    3 Cataratas de        Munising       3
      Alger
...
```

Desglosemos cada parte del bloque de código.

Alias de tabla

```
cascada w JOIN recorrido t
```

Las tablas `waterfall` y `tour` reciben los alias de tabla `w` y `t`, que son nombres temporales para las tablas dentro de la consulta. Los alias de tabla no son necesarios en una cláusula `JOIN`, pero son muy útiles para acortar los nombres de tabla a los que hay que hacer referencia en las cláusulas `ON` y `SELECT`.

UNIRSE A ... EN ...

```
cascada w JOIN recorrido t
ON w.id = t.stop
```

Estas dos tablas se unen mediante la palabra clave `JOIN`. Una cláusula `JOIN` siempre va seguida de una cláusula `ON`, que especifica cómo deben vincularse las tablas. En este caso, el `id de la cascada` en la `tabla cascada` debe coincidir con la `parada` con la cascada en la tabla `recorrido`.

NOTA

Es posible que vea las cláusulas `FROM`, `JOIN` y `ON` en líneas diferentes o con sangría. Esto no es necesario, pero resulta útil para facilitar la lectura, especialmente cuando se unen varias tablas.

Tabla de resultados

Una consulta siempre da como resultado una única tabla. La `tabla cascada` tiene 12 columnas y la tabla `recorrido` tiene 3 columnas. Tras unir estas tablas, la tabla de resultados tiene 15 columnas.

```
id    nombre           ... nombre    para...
----- ---------------      ---------  -----
    1 Cataratas            M-28           1
      Munising
    1 Cataratas            Munising       1
      Munising
    2 Cascadas             Munising       2
      Tannery
    3 Cataratas de         M-28           3
      Alger
    3 Cataratas de         Munising       3
      Alger
...
```

Observará que hay dos columnas llamadas `nombre` en la tabla de resultados. La primera es de la tabla `cascada` y la segunda es de la tabla `recorrido`. Para referirse a ellas en la cláusula `SELECT`, tendría que calificar los nombres de las columnas.

```
SELECCIONE w.nombre, t.nombre
FROM cascada w JOIN gira t
     ON w.id = t.parada;

nombre           nombre
---------------- ---------
Munising Falls M-28
Munising Falls Munising
Tannery Falls   Munising
...
```

Para diferenciar las dos columnas, también puede utilizar el alias para los nombres de las columnas.

```
SELECT w.name AS waterfall_name,
       t.name AS tour_name
FROM cascada w JOIN gira t
     ON w.id = t.parada;

nombre_cascada nombre_viaje
---------------- ----------
Munising Falls M-28
Munising Falls Munising
Tannery
                FallsMun
```

```
ising Alger     FallsM-
28
Cataratas de Alger      Munising
...
```

En el ejemplo anterior, si una cascada no aparece en ningún recorrido, no aparecerá en la tabla de resultados. Si quisiera ver todas las cascadas en los resultados, tendría que utilizar o t r o tipo de unión.

JOIN Por defecto INNER JOIN

Este ejemplo utiliza una simple palabra clave `JOIN` para reunir los datos de dos tablas, aunque es recomendable indicar explícitamente el tipo de unión que está utilizando en . `JOIN` en sí mismo es por defecto un `INNER JOIN`, lo que significa que sólo los registros que están en ambas tablas se devuelven en los resultados.

Existen varios tipos de unión utilizados en SQL, que se describen con más detalle en "Unir tablas" en la página 270 del Capítulo 9.

A partir de subconsultas

Una subconsulta es una consulta anidada dentro de otra consulta. Las subconsultas dentro de la cláusula `FROM` deben ser sentencias `SELECT` independientes, lo que significa que no hacen referencia a la consulta externa en absoluto y pueden ejecutarse por sí solas.

NOTA

Una subconsulta dentro de la cláusula `FROM` también se conoce como *tabla derivada* porque la subconsulta acaba actuando esencialmente como una tabla mientras dura la consulta.

La siguiente consulta enumera todas las cascadas de titularidad pública, con la parte de la subconsulta en negrita.

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM (SELECT * FROM owner WHERE type = 'public') o
     JOIN cascada w
     ON o.id = w.owner_id;

cascada_nombre propietario_nombre
-------------- --------------
Little Miners Pictured Rocks
Miners Falls Pictured Rocks
Munising Falls Pictured Rocks
Wagner Falls MI DNR
```

Es importante comprender el orden en que se ejecuta la consulta.

Paso 1: Ejecutar la subconsulta

Primero se ejecuta el contenido de la subconsulta. Puede ver que esto da como resultado una tabla de sólo propietarios públicos:

```
SELECT * FROM owner WHERE type = 'public';

identificador          nombre         teléfono      tipo
----- --------------- ------------- -------
    1 Pictured Rocks 906.387.2607 público
    4 MI DNR906       .228.6561 público
```

Volviendo a la consulta original, observará que la subconsulta va seguida inmediatamente de la letra `o`. Éste es el nombre temporal, o alias, que estamos asignando a los resultados de la subconsulta .

NOTA

Los alias son necesarios para las subconsultas dentro de la cláusula `FROM` en *MySQL*, *PostgreSQL* y *SQL Server*, pero no en Oracle y SQLite.

Paso 2: Ejecutar la consulta completa

A continuación, puede pensar en la letra `o` ocupando el lugar de la subconsulta. La consulta se ejecuta ahora como de costumbre.

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM o JOIN cascada w
     ON o.id = w.owner_id;

cascada_nombre propietario_nombre
-------------- --------------
Little Miners Pictured Rocks
Miners Falls Pictured Rocks
Munising Falls Pictured Rocks
Wagner Falls MI DNR
```

Subconsultas frente a la cláusula WITH

Una alternativa a escribir una subconsulta es escribir una expresión común de tabla (CTE) utilizando en su lugar una cláusula `WITH`. La ventaja de la cláusula `WITH` es que la `subconsulta` se nombra desde el principio, lo que hace que el código sea más limpio y también permite hacer referencia a la subconsulta varias veces.

```
WITH o AS (SELECT * FROM owner
           WHERE type = 'public')

SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM o JOIN cascada w
     ON o.id = w.owner_id;
```

La cláusula `WITH` es compatible con *MySQL 8.0+* (2018 y posterior), *PostgreSQL*, *Oracle*, *SQL Server* y *SQLite*. "Expresiones de tabla comunes" en la página 291 del Capítulo 9 incluye más ejemplos de esta técnica.

¿Por qué utilizar una subconsulta en la cláusula FROM?

La principal ventaja de utilizar subconsultas es que se puede convertir un problema de grandes dimensiones de en otros más pequeños. He aquí dos ejemplos:

Ejemplo 1: Múltiples pasos para llegar a los resultados

Supongamos que desea calcular el número medio de paradas de una ruta. En primer lugar, tendrías que calcular el número de paradas de cada recorrido y, a continuación, promediar los resultados.

La siguiente consulta busca el número de paradas de cada recorrido:

```
SELECT name, MAX(stop) as num_stops
FROM tour
GROUP BY nombre;

nombre    número_paradas
--------- ---------
- M-28             11
Munising           6
US-2              14
```

A continuación, podría convertir la consulta en una subconsulta y escribir otra consulta en torno a ella para hallar la media:

```
SELECT AVG(num_stops) FROM
(SELECT name, MAX(stop) as num_stops
FROM tour
GROUP BY nombre) tour_stops;

AVG(número_paradas)
----------------
10.3333333333333
```

Ejemplo 2: La tabla de la cláusula FROM es demasiado grande

El objetivo original era listar todas las cascadas de titularidad pública. En realidad, esto se puede hacer sin una subconsulta y con un `JOIN` en su lugar:

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
DESDE  propietario o
```

```
       JOIN waterfall w ON o.id = w.owner_id
WHERE o.type = 'public';
```

```
cascada_nombre propietario_nombre
-------------- --------------
Little Miners Pictured Rocks
Miners Falls Pictured Rocks
Munising Falls Pictured Rocks
Wagner Falls MI DNR
```

Supongamos que la consulta tarda mucho tiempo en ejecutarse. Esto puede ocurrir cuando se unen tablas de gran tamaño (decenas de millones de filas). Hay varias maneras de reescribir la consulta para acelerarla, y una de ellas es utilizar una subconsulta.

Como sólo nos interesan los propietarios públicos, podemos escribir primero una subconsulta que filtre todos los propietarios privados. La tabla de `propietarios` más pequeña se uniría entonces con la tabla de `cascadas`, lo que llevaría menos tiempo y produciría los mismos resultados.

```
SELECT w.name AS waterfall_name,
       o.name AS owner_name
FROM ( SELECT * FROM propietario
       WHERE type = 'public') o
       JOIN cascada w ON o.id = w.owner_id;

cascada_nombre propietario_nombre
-------------- --------------
Little Miners Pictured Rocks
Miners Falls Pictured Rocks
Munising Falls Pictured Rocks
Wagner Falls MI DNR
```

Estos son sólo dos de los muchos ejemplos de cómo pueden utilizarse las subconsultas para dividir una consulta más amplia en pasos más pequeños.

La cláusula WHERE

La cláusula `WHERE` se utiliza para restringir los resultados de la consulta a sólo las filas de interés de , o dicho de otro modo, es el lugar donde filtrar los datos. Rara vez querrá mostrar todas las filas de una tabla, sino más bien las filas que coincidan con criterios específicos.

CONSEJO

Al explorar una tabla con millones de filas, nunca querrás hacer un `SELECT * FROM mi_tabla;` porque tardará un tiempo innecesariamente largo en ejecutarse.

En su lugar, conviene filtrar los datos. Dos formas comunes de hacerlo son:

Filtrar por una columna dentro de la cláusula `WHERE`

Mejor aún, filtre por una columna que ya esté indexada para que la recuperación sea aún más rápida.

```
SELECCIONAR *
FROM mi_tabla
WHERE year_id = 2021;
```

Mostrar las primeras filas de datos con la cláusula `LIMIT` *(o* `WHERE ROWNUM <= 10` *en Oracle o* `SELECT TOP 10 *` *en SQL Server)*

```
SELECCIONAR *
FROM mi_tabla
LÍMITE 10;
```

La siguiente consulta encuentra todas las cascadas que no contienen *Cataratas* en el nombre. Encontrará más información sobre la palabra clave `LIKE` en el Capítulo 7.

```
SELECT id, name
FROM cascada
WHERE name NOT LIKE '%Falls%';

identificador   nombre
----- ----------------
    7 pequeños mineros
   14 Rapid River Fls
```

La sección en negrita suele denominarse estado condicional o predicado. El predicado realiza una comparación lógica para cada fila de datos que da como resultado TRUE/FALSE/UNKNOWN.

La tabla de cascadas tiene 16 filas. Para cada fila, se comprueba si el nombre de la cascada contiene "Cataratas" o no. Si no contiene *Falls*, el predicado `name NOT LIKE '%Falls%'` es TRUE, y la fila se devuelve en los resultados, como en el caso de las dos filas anteriores.

Predicados múltiples

También es posible combinar varios predicados con *operadores* como AND u OR. El siguiente ejemplo muestra cascadas sin *Falls* en su nombre y que tampoco tienen propietario:

```
SELECT id, name
FROM cascada
WHERE name NOT LIKE '%Falls%'
      AND owner_id IS NULL;

identificador   nombre
----- ---------------
   14 Rapid River Fls
```

Encontrará más información sobre los operadores en Operadores, en el Capítulo 7.

Filtrado de subconsultas

Una subconsulta es una consulta anidada dentro de otra, y la cláusula WHERE de es un lugar habitual para encontrar una. El siguiente ejemplo recupera las cascadas de acceso público situadas en el condado de Alger:

```
SELECCIONE w.nombre
       DESDE cascada w
WHERE w.open_to_public = 'y'
       AND w.county_id IN (
           SELECT c.id FROM condado c
           WHERE c.name = 'Alger');
```

```
nombre
--------------
Cascadas Munising
Cascadas
Tannery
Cascadas Alger
...
```

NOTA

A diferencia de las subconsultas dentro de la cláusula `SELECT` o la cláusula `FROM`, las subconsultas en la cláusula `WHERE` no requieren un alias. De hecho, recibirá un error si incluye un alias.

¿Por qué utilizar una subconsulta en la cláusula WHERE?

El objetivo original era recuperar las cascadas de acceso público situadas en el condado de Alger. Si tuviera que escribir esta consulta desde cero, probablemente empezaría con lo siguiente:

```
SELECCIONE w.nombre
       DESDE cascada w
WHERE w.open_to_public = 'y';
```

Llegados a este punto, ya tiene todas las cascadas accesibles al público. El toque final es encontrar las que están específicamente en el condado de Alger. Ya sabes que la tabla de cascadas no tiene una columna con el nombre del condado, pero la tabla de `condados` sí.

Tiene dos opciones para incluir el nombre del condado en los resultados. Puede (1) escribir una subconsulta dentro de la cláusula `WHERE` que extraiga específicamente la información del condado de Alger o (2) unir las tablas de `cascada` y `condado`:

```
-- Subconsulta en la cláusula
WHERE SELECT w.name
       DESDE cascada w
WHERE w.open_to_public = 'y'
```

```
    AND w.county_id IN (
```

```
            SELECT c.id FROM condado c
            WHERE c.name = 'Alger');
```

o:

```
-- Cláusula JOIN
SELECT w.name
FROM   waterfall w INNER JOIN county c
       ON w.county_id = c.id
WHERE w.open_to_public = 'y'
       AND c.name = 'Alger';

nombre
--------------
Cascadas Munising
Cascadas
Tannery
Cascadas Alger
...
```

Las dos consultas producen los mismos resultados. La ventaja del primer enfoque es que las subconsultas suelen ser más fáciles de entender que las uniones. La ventaja del segundo enfoque es que las uniones suelen ejecutarse más rápido que las subconsultas.

Trabajar > Optimizar

Al escribir código SQL, a menudo hay varias formas de hacer lo mismo.

Tu máxima prioridad debe ser escribir código *que funcione*. Si tarda mucho en ejecutarse o es feo, no importa... ¡funciona!

El siguiente paso, si tienes tiempo, es *optimizar* el código mejorando el rendimiento quizás reescribiéndolo con un JOIN, haciéndolo más legible con sangrías y mayúsculas, , etc.

No te estreses por escribir el código más optimizado desde el principio, , sino por escribir código que funcione. Escribir código elegante se consigue con la experiencia.

Otras formas de filtrar datos

La cláusula `WHERE` no es el único lugar dentro de un estado SELECT- ment para filtrar filas de datos.

- `cláusula FROM`: Al unir tablas, la cláusula `ON` especifica cómo deben vincularse. Aquí es donde puede incluir condiciones para restringir las filas de datos devueltas por la consulta. Para más información, consulte la sección Unir tablas en el Capítulo 9.
- `Cláusula HAVING`: Si hay agregaciones dentro de la sentencia `SELECT`, la `cláusula HAVING` es donde se especifica cómo se deben filtrar las agregaciones de . Para obtener más información, consulte "Cláusula HAVING" en la página 83.
- `Cláusula LIMIT`: Para mostrar un número específico de filas, puede utilizar la `cláusula` LIMIT. En *Oracle*, esto se hace con `WHERE ROWNUM` y en *SQL Server*, se hace con `SELECT TOP`. Ver "La Cláusula LIMIT" en la página 88 de este capítulo para más detalles.

Cláusula GROUP BY

El propósito de la cláusula `GROUP BY` es reunir las filas en grupos y resumir las filas dentro de los grupos de alguna manera , devolviendo finalmente sólo una fila por grupo. En ocasiones, esto se denomina "dividir" las filas en grupos y "agrupar" las filas de cada grupo.

La siguiente consulta cuenta el número de cascadas a lo largo de cada uno de los recorridos:

```
SELECTt  .name COMO tour_name,
         COUNT(*) AS num_waterfalls
FROM     cascada w INNER JOIN gira t
         ON w.id = t.parada
GROUP BY t.name;

tour_name numero_cascadas
```

```
---------- ---------------
M-28                       6
Munising                   6
US-2                       4
```

Hay dos partes en las que centrarse aquí:

- *La recopilación de filas*, que se especifica en la cláusula `GROUP BY`
- *La integración de filas* dentro de grupos, que se especifica en la cláusula `SELECT`

Paso 1: Recogida de filas

En la cláusula `GROUP BY`:

```
GROUP BY t.name
```

decimos que nos gustaría mirar todas las filas de datos y poner las cascadas del tour M-28 en un grupo, todas las cascadas del tour Munising en un grupo, y así sucesivamente. Entre bastidores, los datos se agrupan así:

```
tour_name nombre_cascada
---------- ---------------
M-28Cascadas de         Munising
           M-28Alger Falls
           M-28Scott Falls
           M-28Canyon Falls
           M-28Agate Falls
           M-28Bond Falls

           MunisingCascada
s
           MunisingCascada
s Tannery
           MunisingCascada
s Alger
           MunisingCascada
s Wagner MunisingCascadas
           Horseshoe
           MunisingCascada
```

```
s Miners
```

```
                US-2Bond Falls
                US-2Fumee Falls
                US-2Kakabika Falls
US-2Rapid       River Fls
```

Paso 2: Integración de filas

En la cláusula `SELECT`,

```
SELECT t.name AS tour_name,
       COUNT(*) AS num_waterfalls
```

decimos que para cada grupo, o cada recorrido, queremos contar el número de filas de datos en el grupo. Como cada fila representa una cascada, el resultado sería el número total de cascadas a lo largo de cada recorrido.

La función `COUNT()` se conoce más formalmente como una *función de agregación*, o una función que resume muchas filas de datos en un único valor. Encontrará más funciones de agregación en "Funciones de agregación" en la página 191 del Capítulo 7.

ADVERTENCIA

En este ejemplo, `COUNT(*)` devuelve el número de cascadas de cada recorrido. Sin embargo, esto se debe únicamente a que cada fila de datos de las tablas de `cascadas` y `recorridos` representa una única cascada.

Si una misma cascada apareciera en varias filas, `COUNT(*)` daría como resultado un valor mayor del esperado. En este caso, podría utilizar `COUNT(DISTINCT nombre_cascada)` para encontrar las cascadas únicas. Encontrará más información en en COUNT y DISTINCT.

La clave es que es importante volver a comprobar manualmente los resultados de la función de agregación para asegurarse de que resume los datos de la forma prevista.

Ahora que se han creado los grupos con la cláusula `GROUP BY`, la función de agregado se aplicará una vez a cada grupo:

```
tour_name COUNT(*)
---------- ---------
M-28               6
M-28
M-28
M-28
M-28
M-28

Munising           6
Munising
Munising
Munising
Munising
Munising

US-2               4
US-2
US-2
US-2
```

Todas las columnas a las que no se ha aplicado una función de agregado, que en este caso es la columna `tour_name`, se contraen ahora en un solo valor:

```
tour_name COUNT(*)
---------- ---------
M-28               6
Munising           6
US-2               4
```

NOTA

Este colapso de muchas filas detalladas en una fila agregada significa que cuando se utiliza una cláusula `GROUP BY`, la cláusula `SELECT` *sólo* debe contener:

- Todas las columnas de la cláusula `GROUP BY`: `t.name`
- Agregaciones: `CONTAR(*)`

```
SELECT t.name AS tour_name,
       COUNT(*) AS num_waterfalls
...
GROUP BY t.name;
```

De lo contrario, podría aparecer un mensaje de error o devolver valores inexactos.

GROUP BY En la práctica

Estos son los pasos que debe seguir cuando utilice un `GROUP BY`:

1. Averigüe qué columna o columnas desea utilizar para separar o agrupar los datos (por ejemplo, el nombre del circuito).
2. Calcula cómo te gustaría resumir los datos dentro de cada grupo (por ejemplo, contar las cascadas dentro de cada recorrido).

Cuando te hayas decidido por ellos:

1. En la cláusula `SELECT`, enumere la columna o columnas por las que desea agrupar (por ejemplo, nombre del recorrido) y la agregación o agregaciones que desea calcular dentro de cada grupo (por ejemplo, recuento de cascadas).
2. En la cláusula `GROUP BY`, enumere todas las columnas que no sean agregaciones (es decir, nombre del recorrido).

Para situaciones de agrupación más complejas, incluyendo `ROLLUP`, `CUBE` y `GROUPING SETS`, vaya a "Agrupar y Resumir" en

página 242 en el Capítulo 8.

La cláusula HAVING

La cláusula HAVING impone restricciones a las filas devueltas a partir de una consulta GROUP BY. En otras palabras, permite filtrar en los resultados después de aplicar un GROUP BY.

NOTA

Una cláusula HAVING siempre sigue inmediatamente a una cláusula GROUP BY . Sin una cláusula GROUP BY, no puede haber cláusula HAVING.

Se trata de una consulta que enumera el número de cascadas de cada recorrido utilizando una cláusula GROUP BY:

```
SELECTt  .name COMO tour_name,
         COUNT(*) AS num_waterfalls
         FROMwaterfall w INNER JOIN
tour t
         ON w.id = t.stop
GROUP BY t.name;


tour_name numero_cascadas
---------- ---------------
M-28                    6
Munising                6
US-2                    4
```

Supongamos que sólo queremos listar los circuitos que tienen exactamente seis paradas. Para ello, añada una cláusula HAVING después de la cláusula GROUP BY:

```
SELECTt  .name COMO tour_name,
         COUNT(*) AS num_waterfalls
         FROMwaterfall w INNER JOIN
tour t
         ON w.id = t.stop
GROUP BY t.name
HAVINGCOUNT    (*) = 6;
```

```
tour_name numero_cascadas
---------- ---------------
M-28                     6
Munising                 6
```

DONDE frente a TENER

La finalidad de ambas cláusulas es filtrar datos. Si usted está tratando de:

- Para filtrar columnas concretas, escriba las condiciones en la cláusula `WHERE`
- Filtro sobre agregaciones, escriba sus condiciones dentro del campo

  Cláusula `HAVING`

  El contenido de las cláusulas `WHERE` y `HAVING` no puede intercambiarse:

- Nunca ponga una condición con una agregación en el `WHERE` cláusula. Aparecerá un error.
- Nunca ponga una condición en la cláusula `HAVING` que no implique una agregación. Esas condiciones se evalúan de forma mucho más eficiente en la cláusula `WHERE`.

Observará que la cláusula `HAVING` se refiere a la agregación `CONTAR(*)`,

```
SELECT COUNT(*) AS num_waterfalls
...
TENIENDO COUNT(*) = 6;
```

y no el alias,

```
# el código no se ejecutará
SELECT COUNT(*) AS num_waterfalls
...
HAVING num_waterfalls = 6;
```

Esto se debe al orden de ejecución de las cláusulas . La cláusula `SELECT` se escribe antes que la cláusula `HAVING`. Sin embargo, la cláusula `SELECT` se ejecuta *después de* la cláusula `HAVING`.

Esto significa que el alias `num_waterfalls` de la cláusula `SELECT` no existe en el momento en que se ejecuta la cláusula `HAVING`. En su lugar, la cláusula `HAVING` debe hacer referencia a la agregación bruta `COUNT(*)`.

NOTA

MySQL y *SQLite* son excepciones, y permiten los alias (`num_waterfalls`) en la cláusula `HAVING`.

La cláusula ORDER BY

La cláusula `ORDER BY` se utiliza para especificar cómo desea que se ordenen los resultados de una consulta.

La siguiente consulta devuelve una lista de propietarios y cascadas, sin ordenar:

```
SELECT COALESCE(o.name, 'Desconocido') AS
       owner, w.name AS waterfall_name
       DESDE cascada w
       LEFT JOIN owner o ON w.owner_id = o.id;

propietario       nombre_cascada
---------------- --------------
Pictured Rocks   MunisingFalls
Michigan NatureTannery Falls AF
LLC              Alger Falls
MI               DNRWagner Falls
                 DesconocidoHorseshoe Falls
...
```

La función COALESCE

La función COALESCE sustituye todos los valores NULL de una columna por un valor diferente. En este caso, convirtió los valores NULL de la columna o.name en el texto Desconocido.

Si no se hubiera utilizado aquí la función COALESCE, todas las cascadas sin propietario habrían quedado fuera de los resultados. En cambio, ahora se marcan como de propietario Desconocido, y se pueden clasificar e incluir en los resultados.

Encontrará más detalles en el capítulo 7.

La siguiente consulta devuelve la misma lista, pero primero ordenada alfabéticamente por propietario y luego por cascada:

```
SELECTCOALESCE (o.name, 'Desconocido') COMO
        owner, w.name COMO waterfall_name
        DESDE cascada w
        LEFT JOIN owner o ON w.owner_id = o.id
ORDER BY owner, waterfall_name;

propietario      nombre_cascada
---------------- --------------
AF               LLCAlger Falls
MI DNR
                 FallsNaturale
za de Michigan Tannery Falls
Naturaleza de Michigan Twin
Falls nº 1 Naturaleza de
Michigan Twin Falls nº 2
...
```

La ordenación por defecto e s ascendente, lo que significa que el texto irá de la A a la Z y los números irán de menor a mayor. Puede utilizar las palabras clave ASCENDING y DESCENDING (que pueden abreviarse como ASC y DESC) para controlar la ordenación en cada columna de .

Lo siguiente es una modificación de la ordenación anterior, pero esta vez , ordena los nombres de los propietarios en orden inverso:

```
SELECT COALESCE(o.name, 'Desconocido') AS
       owner, w.name AS waterfall_name
...
ORDER BY owner DESC, waterfall_name ASC;

propietario      nombre_cascada
---------------- --------------
                 DesconocidoAgate Falls
                 DesconocidoBond Falls
                 DesconocidoCanyon Falls
...
```

Puede ordenar por columnas y expresiones que no estén en su Lista SELECT:

```
SELECTCOALESCE (o.name, 'Desconocido') COMO
         owner, w.name COMO waterfall_name
         DESDE cascada w
         LEFT JOIN owner o ON w.owner_id = o.id
ORDER BY o.id DESC, w.id;

propietario      nombre_cascada
---------------- --------------
MI               DNRWagner Falls
AF LLC           Alger Falls
MichiganNaturaleza Tannery
Falls
...
```

También puede ordenar por posición numérica de columna:

```
SELECT COALESCE(o.name, 'Desconocido') AS
       owner, w.name AS waterfall_name
...
ORDENADO POR 1 DESC, 2 ASC;

propietario      nombre_cascada
---------------- --------------
                 DesconocidoAgate Falls
                 DesconocidoBond Falls
```

```
                    DesconocidoCanyon Falls
...
```

Dado que las filas de una tabla SQL no están ordenadas, si no incluye una cláusula `ORDER BY` en una consulta, cada vez que ejecute la consulta, los resultados podrían mostrarse en un orden diferente.

ORDER BY no puede utilizarse en una subconsulta

De las seis cláusulas principales, sólo la cláusula `ORDER BY` no puede utilizarse en una subconsulta. Lamentablemente, no se puede forzar el orden de las filas de una s u b c o n s u l t a .

Para evitar este problema, tendría que reescribir su consulta con una lógica diferente para evitar el uso de una cláusula ORDER `BY` dentro de la subconsulta, y sólo incluir una cláusula `ORDER BY` en la consulta externa .

La cláusula LIMITE

Cuando se visualiza rápidamente una tabla, es una buena práctica devolver un número limitado de filas de en lugar de la tabla completa.

MySQL, *PostgreSQL* y *SQLite* admiten la cláusula `LIMIT`. *Ora- cle* y *SQL Server* utilizan una sintaxis diferente con la misma funcionalidad:

```
-- MySQL, PostgreSQL y SQLite SELECT
*
DEL propietario
LÍMITE 3;

-- Oracle
SELECT *
DEL propietario
DONDE ROWNUM <= 3;

-- SQL Server
```

```
SELECT TOP 3 *
DEL propietario;
```

```
id nombre             teléfono       tipo
--- ----------------- -------------- --------
  1 Pictured Rocks    906.387.2607  público
  2 Naturaleza de     517.655.5655   privado
  Michigan
  3 AF LLC                           privado
```

Otra forma de limitar el número de filas devueltas es filtrar en una columna dentro de la cláusula `WHERE`. El filtrado se ejecutará aún más rápido si la columna está indexada.

Crear, actualizar y borrar

La mayor parte de este libro trata sobre cómo leer datos de una base de datos- con consultas SQL. La lectura es una de las cuatro operaciones básicas de las bases de datos: crear, leer, actualizar y eliminar (CRUD).

Este capítulo se centra en las tres operaciones restantes para Bases de Datos, Tablas, Índices y Vistas. Además, la sección Gestión de Transacciones cubre cómo ejecutar múltiples comandos como una sola unidad.

Bases de datos

Una *base de datos* es un lugar donde almacenar datos de forma organizada.

Dentro de una base de datos, puede crear *objetos de base de datos*, que son cosas que almacenan o hacen referencia a datos. Los objetos de base de datos comunes incluyen tablas, restricciones, índices y vistas.

Un *modelo de datos* o un *esquema* describe cómo se organizan los objetos de una base de datos .

La Figura 5-1 muestra una base de datos que contiene muchas tablas. Los detalles sobre cómo se definen las tablas (por ejemplo, la tabla `Ventas` contiene cinco columnas) y cómo se conectan entre sí (por ejemplo, la columna `customer_id` de la

tabla `Ventas` coincide con la columna

`customer_id` de la tabla `Customer`) forman parte de la columna *esquema* de la base de datos.

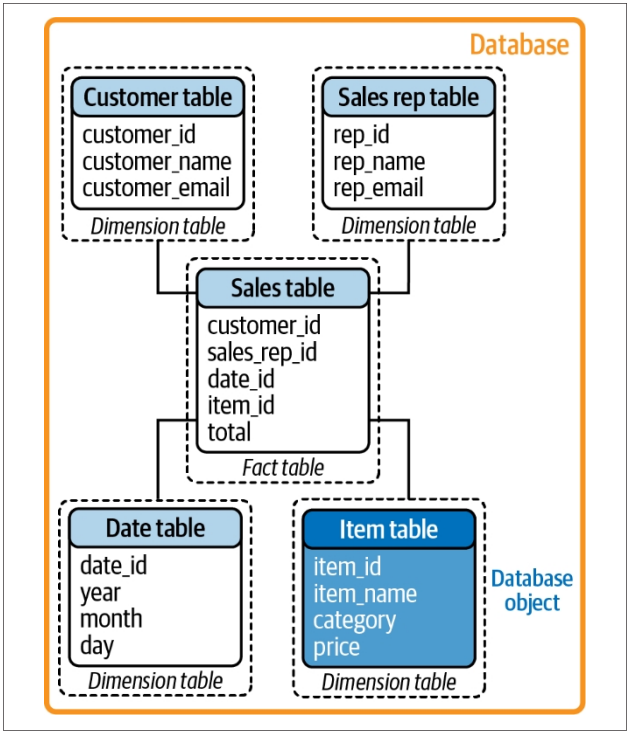

Figura 5-1. Una base de datos que contiene un esquema en estrella

Las tablas de la Figura 5-1 están dispuestas en un *esquema de estrella*, que es una forma básica de organizar tablas en una base de datos. El esquema en estrella incluye una *tabla de hechos* en el centro y está rodeada de *tablas de dimensiones* (también conocidas como *tablas de consulta*). La tabla de hechos registra las transacciones realizadas (ventas en este caso) junto con identificadores de información adicional, que se detallan en las tablas de dimensiones.

Modelo de datos frente a esquema

Cuando se diseña una base de datos, primero se elabora un *modelo de datos*, que es cómo se quiere organizar la base de datos en a alto nivel. Podría parecerse a la Figura 5-1 e incluir los nombres de las tablas, cómo están conectadas entre sí, etc.

Cuando esté listo para pasar a la acción, deberá crear un *esquema*, que es la implementación del modelo de datos en una base de datos. En el programa informático con el que trabajes, deberás especificar las tablas, restricciones, claves primarias y externas, etc.

NOTA

La definición de un esquema varía para algunos RDBMS.

En *MySQL*, un esquema es lo mismo que una base de datos y los dos términos pueden utilizarse indistintamente.

En *Oracle*, un esquema consiste en los objetos de la base de datos propiedad de un usuario concreto, por lo que los términos *esquema* y *usuario* se utilizan indistintamente .

Mostrar nombres de bases de datos existentes

Todos los objetos de base de datos residen en bases de datos, por lo que un buen primer paso es para ver qué bases de datos existen actualmente. La Tabla 5-1 muestra el código para mostrar los nombres de todas las bases de datos existentes en cada RDBMS.

Tabla 5-1. Código para mostrar los nombres de las bases de datos existentes

RDBMSCode	MySQL
	`Mostrar bases de datos;`
Oracle	`SELECT * FROM nombre_global;`
PostgreSQL	
SQL Server	`SELECT name FROM master.sys.databases;`

SQLite	`.database` (o busque archivos *.db* en el explorador de archivos)

NOTA

SQLite: Para la mayoría del software RDBMS, las bases de datos se encuentran dentro del RDBMS. Sin embargo, en el caso de SQLite, las bases de datos se almacenan en fuera de SQLite como archivos *.db*. Para utilizar una base de datos, debe especificar un nombre de archivo .db al iniciar SQLite:

```
> sqlite3 db_existente.db
```

Nombre de la base de datos actual

Es posible que desee confirmar la base de datos en la que se encuentra actualmente antes de escribir cualquier consulta. La Tabla 5-2 muestra el código para mostrar el nombre de la base de datos en la que se encuentra para cada RDBMS.

Tabla 5-2. Código para mostrar el nombre de la base de datos actual

RDBMS	Code
MySQL	`SELECT base de datos();`
Oracle	`SELECT * FROM global_name;`
PostgreSQL	`SELECT current_database();`
SQL Server	`SELECT db_name();`
SQLite	`.base de datos`

NOTA

Te habrás dado cuenta de que el código actual de la base de datos es el mismo que el existente para Oracle y SQLite.

Una instancia de *Oracle* sólo puede conectarse a una única base de datos a la vez, y normalmente no se cambia de base de datos.

Con *SQLite*, sólo puedes abrir y trabajar con un único archivo de base de datos a la vez.

Cambiar a otra base de datos

Es posible que desee utilizar datos de otra base de datos o cambiar a una base de datos recién creada. La Tabla 5-3 muestra el código para cambiar a otra base de datos en cada RDBMS.

Tabla 5-3. Código para cambiar a otra base de datos

RDBMS	Código
MySQL, Servidor SQL	`USE otra_db;`
Oracle	Típicamente no cambias de base de datos (ver nota anterior), pero para cambiar de usuario, escribirías: **`connect another_user`**
PostgreSQL	`\c otra_db`
SQLite	`.abrir otra_db`

Crear una base de datos

Si tiene privilegios CREATE, puede crear una nueva base de datos. Si no, es posible que sólo pueda trabajar con una base de datos existente. La Tabla 5-4 muestra el código para crear una base de datos en cada RDBMS.

Tabla 5-4. Código para crear una base de datos

RDBMS	Código
MySQL, Oracle, PostgreSQL, SQL Server	`CREAR BASE DE DATOS mi_nueva_db;`
SQLite	`> sqlite3 mi_nueva_db.db`

NOTA

Oracle: Existen algunos pasos adicionales (relativos a instancias, variables de entorno, etc.) en torno a la sentencia `CREATE DATABASE` en Oracle, que pueden consultarse en la documentación de Oracle.

SQLite: El símbolo `>` no es un carácter que se escriba realmente. Sólo significa que se trata de código de línea de comandos, no de código SQL.

Eliminar una base de datos

Si tiene privilegios DELETE, puede eliminar una base de datos. La Tabla 5-5 muestra el código para eliminar una base de datos en cada RDBMS.

ADVERTENCIA

Si elimina una base de datos, perderá todos los datos que contenga. *No se puede deshacer*, a menos que se haya creado una copia de seguridad. Recomiendo no ejecutar este comando a menos que esté 100% seguro de que no necesita la base de datos.

Tabla 5-5. Tabla 5-5. Código para borrar una base de datos

RDBMS	Código
MySQL, Oracle, PostgreSQL, SQL Server	`DROP DATABASE mi_nueva_db;`
SQLite	Borra el archivo *.db* en el explorador de archivos

NOTA

Oracle: Existen algunos pasos adicionales (relativos al montaje, etc.) en torno a la sentencia `DROP DATABASE` en Oracle, que pueden encontrarse en la documentación de Oracle.

En algunos RDBMS, no se puede abandonar una base de datos en la que se está actualmente. Tendrías que cambiar primero a otra base de datos, como la predeterminada, antes de soltar la base de datos:

- En *PostgreSQL*, la base de datos por defecto es `postgres`:

```
\c postgres
DROP DATABASE mi_nueva_db;
```

- En *SQL Server*, la base de datos por defecto es `master`:

```
USE master;
go
DROP DATABASE
mi_nueva_db; go
```

Creación de tablas

Las tablas constan de filas y columnas, y almacenan todos los datos de una base de datos. En SQL, existen algunos requisitos adicionales para las tablas:

- Cada fila de una tabla debe ser única
- Todos los datos de una columna deben ser del mismo tipo (entero, texto, etc.).

NOTA

En *SQLite*, los datos de una columna *no* tienen por qué ser todos d e l mismo tipo. SQLite es más flexible en el sentido de que cada valor tiene asociado un tipo de datos, en lugar de una columna entera.

Para ser compatible con otros RDBMSs, SQLite su- porta columnas que tienen asignaciones de tipos de datos. Estas *afinidades de tipo* son tipos de datos recomendados para las columnas, y no son obligatorios.

Crear una tabla simple

La creación de una tabla en SQL requiere dos pasos. Primero debe definir la estructura de una tabla antes de cargar datos en ella:

1. Crea una tabla.

   El siguiente código crea una tabla vacía llamada `mi_simple_tabla` con tres columnas: `id`, `país` y `nombre`. Todos los valores de la primera columna (`id`) deben ser enteros, y las otras dos columnas (`country`, `name`) pueden contener hasta 2 y hasta 15 caracteres:

   ```
   CREATE TABLE mi_tabla_simple (
      id INTEGER,
      país VARCHAR(2),
      nombre VARCHAR(15)
   );
   ```

   En el capítulo 6 se enumeran más tipos de datos además de `INTEGER` y `VARCHAR`.

2. Insertar filas.
   a. Inserta una única fila de datos.

      El siguiente código inserta una fila de datos en las columnas `id`, `country` y `name`:

      ```
      INSERT INTO my_simple_table (id, country, name)
      VALUES (1, 'US', 'Sam');
      ```

   b. Insertar varias filas de datos.

La Tabla 5-6 muestra cómo insertar varias filas de datos en una tabla en cada RDBMS, en lugar de una fila cada vez.

Tabla 5-6. Código para insertar múltiples filas de datos

RDBMS	Código

MySQL, PostgreSQL, SQL Server, SQLite	`INSERT INTO`
	`  mi_tabla_simple`
	`  (id, país, nombre)`
	`VALUES (2, 'US', 'Selena'),`
	`  (3, 'CA', 'Shawn'),`
	`  (4, "US", "Sutton");`

Oracle	`INSERTAR TODO`
	`  INTO mi_tabla_simple (id, país, nombre)`
	`  VALUES (2, 'US', 'Selena')`
	`  INTO mi_tabla_simple (id, país, nombre)`
	`  VALUES (3, 'CA', 'Shawn')`
	`  INTO mi_tabla_simple (id, país, nombre)`
	`  VALUES (4, "US", "Sutton")`
	`SELECT * FROM dual;`

Después de insertar los datos, la tabla tendría el siguiente aspecto:

```
SELECT * FROM mi_tabla_simple;

id país nombre
--- -------- -------
  1 EE.UU.   Sam
  2 EE.UU.   Selena
  3 CA       Shawn
  4 US       Sutton
```

Al insertar filas de datos, el orden de los valores debe coincidir exactamente con el orden de los nombres de las columnas.

Los valores de cualquier columna omitida de la lista de columnas tomarán su valor por defecto de `NULL`, a menos que se especifique otro valor por defecto.

NOTA

Necesita privilegios CREATE para crear una tabla. Si obtiene un error al ejecutar el código anterior, no tiene permiso para hacerlo y necesita hablar con el administrador de su base de datos.

Mostrar nombres de tablas existentes

Antes de crear una tabla, puede querer ver si el nombre de la tabla ya existe. La Tabla 5-7 muestra el código para mostrar los nombres de las tablas existentes en la base de datos para cada RDBMS.

Tabla 5-7. Código para mostrar los nombres de las tablas existentes

RDBMSCode	Code
MySQL	`MOSTRAR tablas;`
Oracle	`-- Todas las tablas, incluidas las tablas del sistema SELECT nombre_tabla FROM todas_las_tablas;` `-- Todas las tablas creadas por el usuario SELECT nombre_tabla FROM tablas_usuario;`
PostgreSQL	`\dt`
SQL Server	`SELECT nombre_tabla FROM esquema_informacion.tablas;`
SQLite	`.tables`

Crear una tabla que aún no existe

En *MySQL*, *PostgreSQL* y *SQLite*, puede comprobar si existen tablas utilizando las palabras clave `IF NOT EXISTS` al crear una tabla:

```
CREATE TABLE IF NOT EXISTS mi_tabla_simple (
   id INTEGER,
   país VARCHAR(2),
   nombre VARCHAR(15)
```

```
);
```

Si el nombre de la tabla no existe, se creará una nueva tabla. Si el nombre de la tabla ya existe, sin el IF NOT EXISTS, recibirá un mensaje de error. Con el IF NOT EXISTS, no se creará ninguna tabla nueva y se evitará el mensaje de error.

Si desea sustituir una tabla existente, puede hacerlo de dos maneras:

- Puede utilizar `DROP TABLE` para eliminar completamente la tabla existente y, a continuación, crear una nueva.
- Puede *truncar* la tabla existente, lo que significa que conserva el esquema (es decir, la estructura) de la tabla, pero borra los datos de que contiene. Esto puede hacerse utilizando `DELETE FROM` para borrar los datos de la tabla.

Crear una tabla con restricciones

Una *restricción* es una regla que especifica qué datos pueden insertarse en una tabla. El código siguiente crea dos tablas y varias restricciones (en negrita):

```
CREAR TABLA otra_tabla ( país
   VARCHAR(2) NOT NULL, nombre
   VARCHAR(15) NOT NULL,
   description VARCHAR(50),
   CONSTRAINT pk_another_table
      PRIMARY KEY (país, nombre)
);

CREATE TABLE mi_tabla (
   id INTEGER NOT NULL,
   country VARCHAR(2) DEFAULT 'CA'
      CONSTRAINT chk_país
      CHECK (country IN ('CA','US')),
   name VARCHAR(15),
   nombre_cap VARCHAR(15),
   CONSTRAINT pk
      PRIMARY KEY (id),
   CONSTRAINT fk1
```

```
        FOREIGN KEY (país, nombre)
        REFERENCIAS otra_tabla (país, nombre), CONSTRAINT
    unq_nombre_país
        UNIQUE (país, nombre),
    CONSTRAINT chk_upper_name
        CHECK (nombre_cap = UPPER(nombre))
);
```

La palabra clave `CONSTRAINT` nombra la restricción para futuras referencias y es opcional. Evite utilizar el mismo nombre para una columna y una restricción.

Para acceder rápidamente a las secciones de restricciones: `NOT NULL`, `DEFAULT`, `CHECK`, `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`.

Restricción: No permitir valores NULL en una columna con NOT NULL

En una tabla SQL, las celdas sin valor se sustituyen por el término NULL. Para cada columna, puede especificar si los valores `NULL` están permitidos o no:

```
CREATE TABLE mi_tabla (
    id INTEGER NOT NULL,
    país VARCHAR(2) NULL,
    nombre VARCHAR(15)
);
```

La restricción `NOT NULL` en la columna `id` significa que la columna no permitirá valores `NULL`. En otras palabras, no puede haber valores perdidos insertados en la columna, o de lo contrario obtendrá un mensaje de error.

La restricción `NULL` en la columna `país` significa que la columna permitirá valores `NULL`. Si inserta datos en la tabla y excluye la `columna de país`, no se insertará ningún valor y la celda se sustituirá por un valor `NULL`.

Si no se especifica `NULL` o `NOT NULL`, la columna `name` será por defecto `NULL`, lo que significa que permitirá valores `NULL`.

Restricción: Establecer valores por defecto en una columna con DEFAULT

Al insertar datos en una tabla, los valores que faltan se sustituyen por el término NULL. Para sustituir los valores que faltan por otro valor de , puede utilizar la restricción DEFAULT. El siguiente código convierte cualquier valor de país que falte en CA:

```
CREATE TABLE mi_tabla (
   id INTEGER,
   country VARCHAR(2) DEFAULT 'CA',
   name VARCHAR(15)
);
```

Restricción: Restricción de valores en una columna con CHECK

Puede restringir los valores permitidos en una columna utilizando la restricción CHECK. El siguiente código sólo permite valores de CA y US en la columna país.

Puede colocar la palabra clave CHECK inmediatamente después del nombre de la columna y el tipo de datos:

```
CREATE TABLE mi_tabla (
   id INTEGER,
   country VARCHAR(2) CHECK
      (country IN ('CA', 'US')),
   nombre VARCHAR(15)
);
```

O puede colocar la palabra clave CHECK después de todos los nombres de columnas y tipos de datos:

```
CREATE TABLE mi_tabla (
   id INTEGER,
   país VARCHAR(2),
   nombre VARCHAR(15),
   CHECK (country IN ('CA','US'))
);
```

También puede incluir lógica que compruebe varias columnas:

```
CREATE TABLE mi_tabla (
   id INTEGER,
   país VARCHAR(2),
```

```
    nombre VARCHAR(15),
    CONSTRAINT chk_id_país
    CHECK (id > 100 AND country IN ('CA','US'))
);
```

Restricción: Exigir valores únicos en una columna con UNIQUE

Puede exigir que los valores de una columna sean únicos utilizando la restricción `UNIQUE`.

Puede colocar la palabra clave `UNIQUE` inmediatamente después del nombre de la columna y del tipo de datos:

```
CREATE TABLE mi_tabla (
    id INTEGER UNIQUE,
    país VARCHAR(2),
    nombre VARCHAR(15)
);
```

O puede colocar la palabra clave `UNIQUE` después de todos los nombres de columnas y tipos de datos:

```
CREATE TABLE mi_tabla (
    id INTEGER,
    país VARCHAR(2),
    nombre VARCHAR(15),
    UNIQUE (id)
);
```

También puede incluir una lógica que obligue a que la combinación de varias columnas sea única. El siguiente código requiere *combinaciones únicas* de `país`/`nombre`, lo que significa que una fila puede incluir `CA`/`Emma` y otra `US`/`Emma`:

```
CREATE TABLE mi_tabla (
    id INTEGER,
    país VARCHAR(2),
    nombre VARCHAR(15),
    CONSTRAINT unq_nombre_país
    UNIQUE (país, nombre)
);
```

Crear una tabla con claves primarias y foráneas

Las claves primarias y las claves externas son tipos especiales de restricciones que identifican de forma exclusiva las filas de datos.

Especificar una clave primaria

Una clave *primaria* identifica de forma exclusiva cada fila de datos de una tabla. Una clave primaria puede estar formada por una o varias columnas de una tabla. Todas las tablas deben tener una clave primaria.

Puede colocar las palabras clave `PRIMARY KEY` inmediatamente después del nombre y el tipo de datos de la columna:

```
CREAR TABLA mi_tabla (
   id INTEGER PRIMARY KEY,
   país VARCHAR(2),
   nombre VARCHAR(15)
);
```

También puede colocar las palabras clave `PRIMARY KEY` después de todos los nombres de columnas y tipos de datos:

```
CREATE TABLE mi_tabla (
   id INTEGER,
   país VARCHAR(2),
   nombre VARCHAR(15),
   PRIMARY KEY (id)
);
```

Para especificar una clave primaria formada por varias columnas (también conocida como *clave compuesta*):

```
CREATE TABLE mi_tabla (
   id INTEGER NOT NULL,
   país VARCHAR(2),
   nombre VARCHAR(15) NOT NULL,
   CONSTRAINT
   pk_id_nombre PRIMARY
   KEY (id, nombre)
);
```

Al crear una `PRIMARY KEY`, las restricciones que está poniendo en la(s) columna(s) son que no pueden incluir valores `NULL` (`NOT NULL`) y los valores deben ser únicos (`UNIQUE`).

Mejores prácticas para claves principales

Cada tabla debe tener una clave primaria. Esto garantiza que cada fila de pueda identificarse de forma única.

Se recomienda que las claves primarias consistan en columnas ID, como (`country_id, name_id`) en lugar de (`country, name`). Técnicamente, varias filas podrían tener la misma combinación de `país` y `nombre`. Añadiendo columnas que contengan ID únicos (`101`, `102`, etc.), se garantiza que la combinación de `país_id` y `nombre_id` sea única.

Las claves primarias deben ser inmutables, lo que significa que no pueden modificarse . Esto permite que una fila concreta de una tabla esté siempre identificada por la misma clave primaria.

Especificar una clave externa

Una *clave externa* de una tabla hace referencia a una clave primaria de otra tabla. Las dos tablas pueden vincularse mediante la columna común. Una tabla puede tener cero o más claves externas.

La Figura 5-2 muestra un modelo de datos de dos tablas: la tabla de `clientes`, que tiene una clave primaria de `id`, y la tabla de `pedidos`, que tiene una clave primaria de `o_id`. Desde el punto de vista de `los clientes`, su columna `order_id coincide` con los valores de la columna `o_id`, lo que convierte a `order_id` en una clave ajena porque hace referencia a una clave primaria de otra tabla.

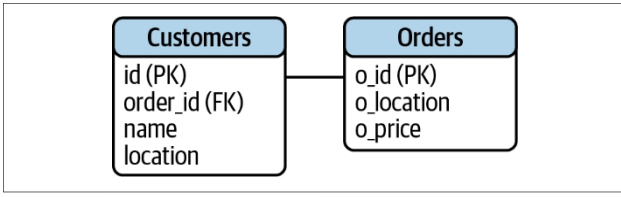

Figura 5-2. Dos tablas con claves primarias y una clave externa

Para especificar una clave externa, siga estos pasos:

1. Localice la tabla a la que desea hacer referencia e identifique la clave primaria.

   En este caso, haremos referencia a las órdenes, concretamente a la orden
   columna o_id:

   ```
   CREAR TABLA pedidos (
      o_id INTEGER PRIMARY KEY,
      o_localización
      VARCHAR(20), o_precio
      DECIMAL(6,2)
   );
   ```

2. Cree una tabla con una clave externa que haga referencia a la clave primaria de la otra tabla.

   En este caso, estamos creando la tabla clientes donde la columna order_id hace referencia a la clave primaria o_id de la tabla pedidos:

   ```
   CREAR TABLA clientes (
    id INTEGER PRIMARY KEY,
    order_id INTEGER,
    name VARCHAR(15),
    location VARCHAR(20),
    FOREIGN KEY (order_id)
    REFERENCES orders (o_id)
   );
   ```

Para especificar una clave externa formada por varias columnas, la clave primaria también debe estar formada por varias columnas:

```
CREAR TABLA pedidos (
   o_id INTEGER,
   o_localización VARCHAR(20),
   o_precio DECIMAL(6,2),
   PRIMARY KEY (o_id, o_location)
);

CREAR TABLA clientes ( id
   INTEGER PRIMARY KEY,
   order_id INTEGER,
   name VARCHAR(15),
   location VARCHAR(20),
   CONSTRAINT nombre_id_fk
   FOREIGN KEY (order_id, location)
   REFERENCIAS pedidos (o_id,
   o_location)
);
```

NOTA

La clave externa (`order_id`) y la clave primaria a la que hace referencia (`o_id`) deben ser ambas del mismo tipo de datos.

Crear una tabla con un campo generado automáticamente

Si planea cargar un conjunto de datos sin una columna de ID único, puede crear una columna que genere automáticamente un ID único . El código de la Tabla 5-8 genera automáticamente números secuenciales (1, 2, 3, etc.) en la `columna u_id`, en cada RDBMS.

Tabla 5-8. Código para generar automáticamente un ID único

RDBMSCode MySQL

```
            CREAR TABLA mi_tabla (
                   u_id INTEGER PRIMARY KEY AUTO_INCREMENT,
                   país VARCHAR(2),
                   nombre VARCHAR(15)
                );

Oracle      CREAR TABLA mi_tabla (
               u_id INTEGER GENERADO POR DEFAULT
                           EN NULL COMO IDENTIDAD,
              país VARCHAR2(2),
              nombre VARCHAR2(15)
            );

PostgreSQL  CREAR TABLA mi_tabla (
               u_id SERIAL, país
               VARCHAR(2), nombre
               VARCHAR(15)
            );

SQL Server--  u_id debe comenzar en 1 e incrementarse
            en 1 CREATE TABLE mi_tabla (
               u_id INTEGER IDENTITY(1,1),
               país VARCHAR(2),
               nombre VARCHAR(15)
            );

SQLite      CREAR TABLA mi_tabla (
               u_id INTEGER PRIMARY KEY AUTOINCREMENT,
               país VARCHAR(2),
               nombre VARCHAR(15)
            );
```

NOTA

En *Oracle*, se suele utilizar `VARCHAR2` en lugar de `VARCHAR`. Son idénticos en términos de funcionalidad, pero `VARCHAR` puede modificarse algún día, por lo que es más seguro utilizar `VARCHAR2`.

SQLite recomienda no utilizar `AUTOINCREMENTO` a menos que sea absolutamente necesario, ya que utiliza recursos informáticos adicionales. El código seguirá ejecutándose sin errores.

Insertar los resultados de una consulta en una tabla

En lugar de escribir manualmente los valores a insertar en una nueva tabla, es posible que desee cargar una nueva tabla con los datos de la(s) tabla(s) existente(s).

Aquí tienes una tabla:

```
SELECT * FROM mi_tabla_simple;

id país nombre
--- -------- -------
  1 EE.UU.   Sam
  2 EE.UU.   Selena
  3 CA       Shawn
  4 US       Sutton
```

Cree una nueva tabla con dos columnas:

```
            CREATE TABLE
    nueva_tabla_dos_columnas ( id
              INTEGER,
            nombre VARCHAR(15)
);
```

Inserta los resultados de una consulta en la nueva tabla:

```
           INSERT INTO
  nueva_tabla_dos_columnas (id,
             nombre)
SELECT id, nombre
```

```
DESDE  mi_tabla_simple
WHERE id < 3;
```

La nueva tabla quedaría así:

```
SELECT * FROM nueva_tabla_dos_columnas;

id nombre
--- -------
  1 Sam
  2 Selena
```

También puede insertar valores de una tabla existente y añadir o modificar otros valores por el camino.

Cree una nueva tabla con cuatro columnas:

```
CREATE TABLE
             nueva_tabla_cuatro_colum
             nas ( id INTEGER,
             nombre VARCHAR(15),
             nueva_columna_num INTEGER,
             nueva_columna_texto VARCHAR(30)
);
```

Inserta los resultados de una consulta en la nueva tabla y rellena los valores de las nuevas columnas:

```
INSERT INTO nueva_tabla_cuatro_columnas
       (id, name, new_num_column, new_text_column)
SELECT id, name, 2017, 'stargazing'
DESDE  mi_tabla_simple
WHERE id = 2;
```

Inserta los resultados de una consulta en la nueva tabla y cambia un valor de la fila (`id` en este caso):

```
INSERT INTO nueva_tabla_cuatro_columnas
       (id, nombre, nueva_columna_num,
nueva_columna_texto) SELECT 3, nombre, 2017, 'lobos'
DESDE  mi_tabla_simple
WHERE id = 2;
```

La nueva tabla quedaría así:

```
SELECT * FROM nueva_tabla_cuatro_columnas;

id          nombre_nuevo_num_columna nuevo_texto_columna
--- ------- ---------------- ---------------
  2 Selena 2017              observación de estrellas
  3 Selena 2017              lobos
```

Insertar datos de un archivo de texto en una tabla

Es posible que desee cargar datos de un *archivo de texto* (datos almacenados en texto plano sin formato especial) en una tabla. Un tipo común de archivo de texto es un archivo *.csv* (valores separados por comas). Los archivos de texto pueden abrirse en aplicaciones ajenas a un RDBMS, como Excel, Bloc de notas, TextEdit, etc.

El siguiente código muestra cómo cargar el archivo *mis_datos.csv* en una tabla.

Contenido del fichero *mis_datos.csv*:

```
unique_id,canada_us,artist_name
5, "CA", "Celine"
6, "CA", "Michael"
7, "US", "Stefani" 8,,
"Olivia"
...
```

Crea una tabla:

```
CREATE TABLE nueva_tabla
   ( id INTEGER,
   país VARCHAR(2),
   nombre VARCHAR(15)
);
```

El código de la Tabla 5-9 carga el archivo *mis_datos.csv* en la tabla `nueva_tabla` para cada RDBMS. Al cargar datos, puede especificar detalles adicionales sobre los datos, como:

- Los datos se separan por comas (`,`)

- Los valores de texto van entre comillas dobles ("")
- Cada nueva fila está en una nueva línea (\n)
- La primera fila del archivo de texto (que contiene la cabecera) debe ignorarse.

Tabla 5-9. Código para insertar datos de un archivo .csv

RDBMSCode MySQL

```
CARGAR DATOS LOCAL
INFILE
'<ruta_archivo>/mis_datos.csv'
INTO TABLE nueva_tabla
CAMPOS TERMINADOS POR ','
ENCERRADOS POR '"'
LÍNEAS TERMINADAS POR
'\n' IGNORAN 1 FILAS;
```

OracleSi bien esto se puede hacer en la línea de comandos utilizando `sqlldr`, el mejor enfoque es cargar los datos a través de una interfaz gráfica de usuario como SQL*Loader o SQL Developer en su lugar.

PostgreSQL
```
\copy nueva_tabla
      FROM
      '<ruta_archivo>/mis_datos.csv'
      DELIMITADOR ',' CSV HEADER
```

SQL Server
```
BULK INSERT nueva_tabla
FROM
'<ruta_archivo>/mis_datos.csv'
WITH
(
    FORMAT = 'CSV',
    FIELDTERMINATOR =
    ',', FIELDQUOTE =
    '"', ROWTERMINATOR =
    '\n', FIRSTROW = 2,
    TABLOCK
);
```

SQLite
```
.mode csv
```

```
.import <ruta_archivo>/mis_datos.csv
        nueva_tabla --skip 1
```

Después de insertar los datos, la tabla tendría el siguiente aspecto:

```
SELECT * FROM nueva_tabla;

id país nombre
--- -------- --------
  5 CA       Celine
  6 CA       Michael
  7 EE.UU.   Stefani
  8 NULL     Olivia
...
```

Ejemplo de ruta de archivo al escritorio

Si *mis_datos.csv* está en su Escritorio, así es como s e vería la ruta del archivo para cada sistema operativo:

- Linux: */home/nombre_de_usuario/escritorio/mis_datos.csv*
- MacOS: */Usuarios/mi_nombre_de_usuario/Escritorio/mis_datos.csv*
- Windows: *C:\NUsuarios\Nmi_nombre_de_usuario\Escritorio\Nmis_datos.csv*

NOTA

Si *MySQL* le da un error que dice que la carga de datos locales está deshabilitada, puede habilitarla actualizando la variable global `local_infile`, saliendo y reiniciando MySQL:

```
SET GLOBAL local_infile=1;
quit
```

Datos ausentes y valores nulos

Cada RDBMS interpreta los datos que faltan en un archivo *.csv de* un modo diferente . Cuando la siguiente línea en un archivo *.csv*:

```
8,, "Olivia"
```

se inserta en una tabla SQL, el valor que falta entre `8` y `Olivia` sería reemplazada por:

- Un valor `NULL` en *PostgreSQL* y *SQL Server*
- Una cadena vacía (`''`) en *MySQL* y *SQLite*

En *MySQL* y *SQLite*, puede utilizar `\N` en un archivo *.csv* para representar un
Valor `nulo` en una tabla SQL. Cuando la siguiente línea en un archivo *.csv*,

```
8,\N, "Olivia"
```

se inserta en una tabla *MySQL*, el `\N` sería sustituido por un Valor `nulo` en la tabla.

Cuando se inserta en una tabla *SQLite*, el código `\N` se introduce en la tabla. A continuación, podría ejecutar el código,

```
UPDATE nueva_tabla
SET país = NULL
WHERE país = '\N';
```

para sustituir los marcadores de posición `\NN` por valores `NULL` en la tabla.

Modificación de tablas

En esta sección se explica cómo cambiar el nombre de la tabla, las columnas, las restricciones de y los datos de una tabla.

> **NOTA**
>
> Necesitas privilegios ALTER para modificar una tabla. Si obtiene un error al ejecutar el código de esta sección, no tiene permiso para hacerlo y debe hablar con el administrador de su base de datos .

Cambiar el nombre de una tabla o columna

Después de crear una tabla, puede cambiarle el nombre y las

columnas de la tabla.

ADVERTENCIA

Si modifica una tabla, ésta cambiará permanentemente. *No se puede deshacer*, a menos que se haya creado una copia de seguridad. Compruebe dos veces sus sentencias antes de ejecutarlas.

Cambiar el nombre de una tabla

El código de la Tabla 5-10 muestra cómo renombrar una tabla en cada RDBMS.

Tabla 5-10. Código para renombrar una tabla

RDBMS	Código
MySQL, Oracle, PostgreSQL, SQLite	`ALTER TABLE nombre_tabla_antigua RENOMBRAR A nombre_tabla_nueva;`
SQL ServerEXEC	`sp_rename 'nombre_tabla_anti gua', 'nuevo_nombre_tabla';`

Renombrar una columna

El código de la Tabla 5-11 muestra cómo renombrar una columna en cada RDBMS.

Tabla 5-11. Código para renombrar una columna

RDBMS	Código
MySQL, Oracle, PostgreSQL, SQLite	`ALTER TABLE mi_tabla RENAME COLUMN old_column_name TO new_column_name;`
Servidor SQL	`EXEC sp_rename 'mi_tabla.antiguo_nombre_columna', 'nuevo_nombre_columna', 'COLUMNA';`

Visualizar, añadir y eliminar columnas

Después de crear una tabla, puede ver, añadir y eliminar columnas de la tabla.

Visualizar las columnas de una tabla

El código de la Tabla 5-12 muestra cómo mostrar las columnas de una tabla en cada RDBMS.

Tabla 5-12. Código para mostrar las columnas de una tabla

RDBMS	Código
MySQL, Oracle	`DESCRIBE mi_tabla;`
PostgreSQL	`\d mi_tabla`

SQL Server	`SELECT nombre_columna`
	`FROM esquema_informacion.columnas`
	`WHERE nombre_tabla = 'mi_tabla';`

SQLite	`pragma tabla_info(mi_tabla);`

Añadir una columna a una tabla

El código de la Tabla 5-13 muestra cómo añadir una columna a una tabla en cada RDBMS.

Tabla 5-13. Código para añadir una columna a una tabla

RDBMS	Código

MySQL, PostgreSQL	`ALTER TABLE mi_tabla`
	`ADD nueva_columna_num INTEGER,`
	`ADD nueva_columna_texto VARCHAR(30);`

Oracle	`ALTER TABLE mi_tabla ADD (`
	`nueva_columna_num INTEGER,`
	`nueva_columna_texto`
	`VARCHAR(30));`

SQL Server	`ALTER TABLE mi_tabla`
	`ADD nueva_columna_num INTEGER,`
	`nueva_columna_texto`
	`VARCHAR(30);`

RDBMSCode SQLite

```
ALTER TABLE mi_tabla
      ADD nueva_columna_num INTEGER;
ALTER TABLE mi_tabla
      ADD nueva_columna_texto VARCHAR(30);
```

Eliminar una columna de una tabla

El código de la Tabla 5-14 muestra cómo borrar una columna de una tabla en cada RDBMS.

NOTA

Si una columna tiene alguna restricción, primero debe eliminar las restricciones de antes de eliminar la columna.

Tabla 5-14. Código para eliminar una columna de una tabla

RDBMS	Código

MySQL, PostgreSQL	`ALTER TABLE mi_tabla`
	`DROP COLUMN nuevo_num_columna,`
	`DROP COLUMN`
	`nuevo_texto_columna;`

Oracle	`ALTER TABLE mi_tabla`
	`DROP COLUMN`
	`nueva_num_columna; ALTER TABLE`
	`mi_tabla`
	`DROP COLUMN nueva_columna_texto;`

SQL Server	`ALTER TABLE mi_tabla`
	`DROP COLUMN nueva_num_columna,`
	`nueva_columna_texto;`

SQLite	Referirse a los pasos de modificación manual de SQLite

Modificaciones manuales en SQLite

SQLite no admite algunas modificaciones de tablas, como la eliminación de columnas o la adición/modificación/eliminación de restricciones.

Como solución alternativa, puede utilizar una interfaz gráfica de usuario para generar código que modifique una tabla, o bien crear manualmente una nueva tabla en y copiar los datos (consulte los pasos siguientes).

1. Cree una nueva tabla con las columnas y restricciones que desee.

```
CREAR TABLA mi_tabla_2 (
   id INTEGER NOT NULL,
   país VARCHAR(2),
   nombre VARCHAR(30)
);
```

2. Copia los datos de la tabla antigua a la nueva.

```
INSERT INTO mi_tabla_2
SELECT id, país, nombre
FROM mi_tabla;
```

3. Confirme que los datos están en la nueva tabla.

```
SELECT * FROM mi_tabla_2;
```

4. Borra la tabla antigua.

```
DROP TABLE mi_tabla;
```

5. Cambia el nombre de la nueva tabla.

```
ALTER TABLE mi_tabla_2 RENAME A mi_tabla;
```

Visualizar, añadir y eliminar filas

Después de crear una tabla, puede ver, añadir y eliminar filas de la tabla.

Visualizar las filas de una tabla

Para mostrar las filas de una tabla, basta con escribir una sentencia SELECT:

```
SELECT * FROM mi_tabla;
```

Añadir filas a una tabla

Utilice `INSERT INTO` para añadir filas de datos a una tabla:

```
INSERT INTO mi_tabla
   (id, país, nombre)
VALUES (9, 'US', 'Charlie');
```

Eliminar filas de una tabla

Utilice `DELETE FROM` para eliminar filas de datos de una tabla:

```
DELETE FROM mi_tabla
WHERE id = 9;
```

Omita la cláusula `WHERE` para eliminar todas las filas de una tabla:

```
DELETE FROM mi_tabla;
```

La eliminación de filas de una tabla también se conoce como *truncamiento*, que elimina todos los datos de una tabla sin cambiar la definición de la tabla . Así, aunque los nombres de las columnas y las restricciones de la tabla siguen existiendo, ahora está vacía.

Para deshacerse por completo de una tabla, puede eliminarla.

Visualizar, añadir, modificar y eliminar restricciones

Una *restricción* es una regla que especifica qué datos se pueden insertar en una tabla. Encontrará más información sobre los distintos tipos de restricciones en la sección Crear una tabla con restricciones de este capítulo.

Visualizar las restricciones de una tabla

El código de la Tabla 5-15 muestra cómo mostrar las restricciones de una tabla en cada RDBMS.

Tabla 5-15. Código para mostrar las restricciones de una tabla

RDBMSCode	MySQL
	`SHOW CREAR TABLA mi_tabla;`

Oracle	`SELECT *`
	`FROM user_cons_columns`
	`WHERE nombre_tabla = 'MI_tabla';`

PostgreSQL	`\d mi_tabla`
SQL Server	`-- Lista de restricciones (excepto las predeterminadas) SELECT nombre_tabla, nombre_restricción, tipo_restricción FROM esquema_informacion.restricciones_tabla WHERE nombre_tabla = 'mi_tabla';` `-- Listar todas las restricciones por defecto SELECT OBJECT_NAME(parent_object_id), COL_NAME(parent_object_id, parent_column_id), definition FROM sys.default_constraints WHERE OBJECT_NAME(parent_object_id) = 'mi_tabla';`
SQLite	`.esquema mi_tabla`

NOTA

Oracle almacena los nombres de tablas y columnas en mayúsculas, a menos que rodee el nombre de la columna con comillas dobles. Al hacer referencia a un nombre de tabla o de columna en una sentencia SQL, debe escribir el nombre en mayúsculas (`MI_TABLE`).

Añadir una restricción

Empecemos con la siguiente sentencia `CREATE TABLE`:

```
CREATE TABLE mi_tabla (
   id INTEGER NOT NULL,
   country VARCHAR(2) DEFAULT 'CA',
   name VARCHAR(15),
   nombre_inferior VARCHAR(15)
);
```

El código de la Tabla 5-16 añade una restricción que asegura que la columna `nombre_minúscula` es una versión en minúsculas de la columna `nombre` en cada RDBMS.

Tabla 5-16. Código para añadir una restricción

RDBMS	Código
MySQL, PostgreSQL, SQL Server	`ALTER TABLE mi_tabla ADD CONSTRAINT chk_nombre_inferior CHECK (nombre_inferior = LOWER(nombre));`
Oracle	`ALTER TABLE mi_tabla ADD ( CONSTRAINT chk_nombre_inferior CHECK (nombre_inferior = LOWER(nombre)));`
SQLite	Referirse a los pasos de modificación manual de SQLite

Modificar una restricción

Empecemos con la siguiente sentencia `CREATE TABLE`:

```
CREATE TABLE mi_tabla (
   id INTEGER NOT NULL,
   country VARCHAR(2) DEFAULT 'CA',
   name VARCHAR(15),
   nombre_inferior VARCHAR(15)
);
```

El código de la Tabla 5-17 modifica las siguientes restricciones:

- Cambia la columna de `país` de predeterminada a `CA` a predeterminada a `NULL`
- Cambia la columna del `nombre` de 15 a 30 caracteres.

Tabla 5-17. Código para modificar las restricciones de una tabla

RDBMSCode MySQL

```
ALTER TABLE mi_tabla
      MODIFY país VARCHAR(2) NULL,
      MODIFY nombre VARCHAR(30);
```

Oracle

```
ALTER TABLE mi_tabla MODIFY (
      country DEFAULT NULL,
      name VARCHAR2(30)
);
```

PostgreSQL

```
ALTER TABLE mi_tabla
      ALTER country DROP DEFAULT,
      ALTER name TYPE
      VARCHAR(30);
```

SQL Server

```
ALTER TABLE mi_tabla
      ALTER COLUMN país VARCHAR(2)
      NULL;

ALTER TABLE mi_tabla
      ALTER COLUMN
      nombre
      VARCHAR(30) NULL;
```

SQLiteReferirse a los pasos de modificación manual de SQLite

Eliminar una restricción

El código de la Tabla 5-18 muestra cómo eliminar una restricción de una tabla en cada RDBMS.

Tabla 5-18. Código para eliminar una restricción de una tabla

RDBMS	Código
MySQL	`ALTER TABLE mi_tabla DROP CHECK chk_lower_name;`
Oracle, PostgreSQL, SQL Server	`ALTER TABLE mi_tabla DROP CONSTRAINT chk_nombre_inferior;`
	SQLiteReferirse a los pasos de modificación manual de SQLite

NOTA

En *MySQL*, `CHECK` puede sustituirse por `DEFAULT`, `INDEX` (para restricciones `UNIQUE`), `PRIMARY KEY` y `FOREIGN KEY`. Para eliminar una restricción `NOT NULL`, debe `MODIFICAR` la restricción .

Actualizar una columna de datos

Utilice `UPDATE .. SET ..` para actualizar los valores de una columna de datos. He aquí una tabla:

```
SELECCIONAR *
FROM mi_tabla;

id nombre del país     premios
--- -------- -------- -------
  2 CA       Celine          5
  3 CA       Michael         4
  4          Stefani         9
  EE.UU.
```

Previsualice el cambio que desea realizar:

```
SELECT LOWER(nombre)
FROM mi_tabla;

LOWER(nombre)
```

```
------------
celine
michael
stefani
```

Actualizar los valores de una columna de datos:

```
UPDATE mi_tabla
SET nombre = LOWER(nombre);

SELECT * FROM mi_tabla;



id nombre del país     premios
--- --------- -------- -------
  2 CA        celine          5
  3 CA        michael         4
  4           stefani         9
  EE.UU.
```

Actualizar filas de datos

Utilice `UPDATE .. SET .. WHERE ..` para actualizar los valores de una fila o varias filas de datos.

Aquí tienes una tabla:

```
SELECCIONAR *
FROM mi_tabla;

id nombre del país     premios
--- --------- -------- -------
  2 CA        Celine          5
  3 CA        Michael         4
  4           Stefani         9
  EE.UU.
```

Previsualice el cambio que desea realizar:

```
SELECCIONAR premios + 1
FROM mi_tabla
WHERE país = 'CA';

premios + 1
```

```
-----------
          6
          5
```

Actualizar los valores de varias filas de datos:

```
UPDATE mi_tabla
SET premios = premios + 1
WHERE país = 'CA';

SELECT * FROM mi_tabla;

id nombre del país     premios
--- --------- -------- -------
  2 CA        Celine          6
  3 CA        Michael         5
  4           Stefani         9
  EE.UU.
```

ADVERTENCIA

Es muy importante incluir una cláusula `WHERE` junto con la cláusula `SET` cuando se actualizan filas específicas de datos. Sin la cláusula `WHERE`, se actualizaría toda la tabla .

Actualizar filas de datos con los resultados de una consulta

En lugar de escribir manualmente los valores para actualizar una tabla, puede establecer en un nuevo valor basado en los resultados de una consulta.

Aquí tienes una tabla:

```
SELECT * FROM mi_tabla;

id nombre del país     premios
--- --------- -------- -------
  2 CA        Celine          5
  3 CA        Michael         4
  4           Stefani         9
```

```
EE.UU.
```

Previsualice el cambio que desea realizar:

```
SELECT MIN(premios) FROM mi_tabla;

MIN(premios)
  -----------
           4
```

Actualizar valores a partir de una consulta:

```
UPDATE mi_tabla
SET premios = (SELECT MIN(premios) FROM mi_tabla)
WHERE país = 'CA';

SELECT * FROM mi_tabla;

id nombre del país     premios
--- --------- -------- -------
  2 CA        Celine          4
  3 CA        Michael         4
  4           Stefani         9
  EE.UU.
```

NOTA

MySQL no permite actualizar una tabla con una consulta sobre la misma tabla. En el ejemplo anterior, no puede tener `UPDATE mi_tabla` y `FROM mi_tabla`. El estado se ejecutará si realiza una consulta `FROM otra_tabla`.

Los resultados de la consulta deben devolver siempre una columna y cero o una fila. Si se devuelven cero filas, el valor se establece en `NULL`.

Borrar una tabla

Cuando ya no necesite una tabla, puede eliminarla mediante una sentencia `DROP TABLE`:

```
DROP TABLE mi_tabla;
```

En *MySQL*, *PostgreSQL*, *SQL Server* y *SQLite*, también puede añadir `IF EXISTS` para evitar un mensaje de error si la tabla no existe:

```
DROP TABLE IF EXISTS mi_tabla;
```

ADVERTENCIA

Si elimina una tabla, perderá todos los datos que contenga. *No se puede deshacer*, a menos que se haya creado una copia de seguridad. Recomiendo no ejecutar este comando a menos que esté 100% seguro de que no necesita la tabla.

Eliminar una tabla con referencias de clave externa

Si otras tablas tienen claves foráneas que hacen referencia a la tabla que está eliminando, tendrá que eliminar las restricciones de clave foránea en las otras tablas junto con la tabla que está eliminando.

El código de la Tabla 5-19 muestra cómo borrar una tabla con referencias a claves foráneas en cada RDBMS.

Tabla 5-19. Código para eliminar una tabla con referencias de clave externa

RDBMS	Code
Oracle	`DROP TABLE mi_tabla CASCADE CONSTRAINTS;`
PostgreSQL	`DROP TABLE mi_tabla CASCADE;`
MySQL, SQL Server, SQLite	No existe la palabra clave CASCADE, por lo que debe eliminar manualmente cualquier restricción de clave externa que haga referencia a la tabla antes de eliminarla.

ADVERTENCIA

Es peligroso utilizar `CASCADE` sin saber exactamente lo que está borrando. Proceda con precaución. Recomiendo no ejecutar este comando a menos que esté 100% seguro de que no necesita las restricciones."

Índices

Imagine que tiene una tabla con 10 millones de filas. Escribes una consulta en la tabla para devolver los valores que se registraron el `2021-01-01`:

```
SELECCIONAR *
FROM mi_tabla
WHERE log_date = '2021-01-01';
```

Esta consulta tardaría mucho tiempo en ejecutarse. La razón es que, entre bastidores, se comprueba cada fila para ver si `log_date` coincide con `2021-01-01` o no. Son 10 millones de comprobaciones.

Para acelerar este proceso, puede crear un *índice* en la columna `log_date`. Esto es algo que harías una sola vez, y todas las consultas futuras podrían beneficiarse de ello.

Comparación entre el índice de libros y el índice SQL

Para entender mejor cómo funciona un índice SQL, es útil utilizar una analogía. La Tabla 5-20 compara el índice al final de este libro con un índice en una tabla SQL.

Tabla 5-20. Comparación entre el índice del libro y el índice SQL

	Tabla	BookSQL
TérminosUn	libro tiene muchas *páginas*. Cada página tiene *atributos que* incluyen el recuento de palabras, los conceptos tratados, etc.	Una tabla tiene muchas *filas*. Cada fila tiene *columnas*, entre ellas `customer_id`, `log_date`, etc.
libro y	EscenarioEstás leyendo este desea encontrar todas las páginas sobre el concepto *subconsultas*.	Está consultando una tabla y `d e s e a` encontrar todas las filas en las que `log_date` es `2021-01-01`.
El enfoque lento	Podrías empezar desde la página 1 y hojear cada página de este libro para ver si se mencionan o no *las subconsultas*. Esto llevaría mucho tiempo.	Podría empezar por la fila 1 y recorrer todas las filas para ver si la `fecha de registro` es `2021-01-01` o no. Esto llevaría mucho tiempo.
Crear un índice	Se ha creado un índice para todos los conceptos de este libro. Cada concepto se enumera en el índice junto con los números de página que hablan del concepto.	Se ha creado un índice en la columna `log_date` de la tabla. Cada `log_date` aparece en el índice junto con los números de fila que contienen la `log_date`.
El enfoque rápido	Para encontrar páginas sobre *subconsultas*, puedes ir al índice para encontrar rápidamente los números de página que hacen referencia a *las subconsultas* e ir a esas páginas.	Para encontrar filas con una `log_date` de `2021-01-01`, la consulta utiliza el índice para encontrar rápidamente los números de fila que contienen la fecha y devolver esas filas.

Cuando se ejecuta la misma consulta en `mi_tabla` (que ahora tiene el valor
columna `log_date` indexada):

```
SELECCIONAR *
```

```
FROM mi_tabla
WHERE log_date = '2021-01-01';
```

la consulta se ejecutará mucho más rápido porque en lugar de comprobar cada fila de la tabla, ve la `fecha_registro` de `2021-01-01`, va al índice y extrae rápidamente todas las filas que tienen esa `fecha_registro`.

CONSEJO

Es una buena idea crear un índice en algunas columnas sobre las que se filtra a menudo. Por ejemplo, la columna de clave primaria, la columna de fecha, etc.

Sin embargo, no conviene crear un índice para demasiadas columnas, porque ocupa espacio. Además, cada vez que se añadan o eliminen filas, habrá que reconstruir el índice, lo que lleva mucho tiempo.

Crear un índice para acelerar las consultas

El siguiente código crea un nuevo índice llamado `mi_índice` en el directorio
columna `log_date` en la tabla `mi_tabla`:

```
CREATE INDEX my_index ON my_table (log_date);
```

NOTA

Al crear un índice en *Oracle*, debe escribir el nombre de la columna en mayúsculas y entre comillas:

```
CREATE INDEX my_index ON my_table
  ('LOG_DATE');
```

Oracle crea automáticamente un índice para `PRIMARY KEY` y Columnas `UNIQUE` cuando se crea una tabla.

La creación de índices puede llevar mucho tiempo. Sin embargo, es una tarea que se realiza una sola vez y que merece la pena a largo plazo para poder realizar muchas consultas más rápidas en el futuro.

También puede crear un índice multicolumna o un *índice compuesto*. El siguiente código crea un índice en dos columnas: `log_date` y `team`:

```
CREATE INDEX my_index ON my_table (log_date, team);
```

El orden de las columnas es importante. Si escribe una consulta que filtra en:

- Ambas columnas: el índice agilizará la consulta
- La primera columna (`log_date`): el índice agilizará la consulta
- La segunda columna (`equipo`): el índice no ayudará porque primero organiza los datos por la `fecha_de_registro` y luego por la columna `equipo`.

NOTA

Necesita privilegios CREATE para crear un índice. Si obtiene un error al ejecutar el código anterior, no tiene permiso para hacerlo y necesita hablar con el administrador de su base de datos.

Borrar un índice

El código de la Tabla 5-21 muestra cómo borrar un índice en cada RDBMS .

Tabla 5-21. Código para borrar un índice

RDBMS	Código
MySQL, SQL Server	`DROP INDEX mi_índice ON mi_tabla;`
Oracle, PostgreSQL, SQLite	`DROP INDEX my_index;`

ADVERTENCIA

La eliminación de un índice no se puede deshacer. Asegúrate al 100% de que quieres borrar un índice antes de eliminarlo.

El lado positivo es que no hay pérdida de datos. Los datos de la tabla no se ven afectados y siempre se puede volver a crear el índice.

Vistas

Imagina que tienes una consulta SQL larga y compleja que incluye muchas uniones, filtros, agregaciones, etc. Los resultados de la consulta te resultan útiles y quieres volver a consultarlos más adelante.

Esta es una situación ideal para crear una *vista*, o dar un nombre a la salida de una consulta. Recuerde que el resultado de una consulta es una única tabla, por lo que una vista tiene el mismo aspecto que una tabla. La diferencia es que la vista no contiene datos como una tabla, sino que hace referencia a los datos.

NOTA

A veces, los administradores de bases de datos (DBA) crean vistas en para restringir el acceso a las tablas. Imaginemos que existe una tabla de `clientes`. La mayoría de la gente sólo debería poder leer los datos de la tabla, pero no modificarlos.

El DBA puede crear una vista de `cliente` que incluya datos idénticos a los de la tabla de `clientes`. Ahora, todo el mundo puede consultar la vista *de* `cliente`, y sólo el DBA podría editar los datos dentro de la tabla *de* `clientes`.

El siguiente código es una consulta compleja que no queremos escribir una y otra vez:

```
-- Número de cascadas propiedad de cada
propietario SELECT o.id, o.name,
       COUNT(w.id) AS num_waterfalls
FROM owner o LEFT JOIN waterfall w
     ON o.id = w.owner_id
GROUP BY o.id, o.name;

identificador           nombre   número_cascadas
----- ---------------- --------------
    1 Pictured Rocks                 3
```

```
2 Naturaleza de Michigan          3
3 AF LLC                          1
4 MI DNR                          1
5 Cataratas Horseshoe             0
```

Supongamos que queremos averiguar el número medio de cascadas que posee un propietario. Podríamos hacerlo utilizando una subconsulta o una vista:

```
-- Enfoque de subconsulta
SELECT AVG(número_cascadas) FROM
(SELECT o.id, o.name,
        COUNT(w.id) AS num_waterfalls
FROM owner o LEFT JOIN waterfall w
     ON o.id = w.owner_id
GROUP BY o.id, o.name) my_subquery;

AVG(número_cascadas)
   --------------------
                  1.6


-- Ver enfoque
CREAR VISTA owner_waterfalls_vw COMO
SELECT o.id, o.name,
       COUNT(w.id) AS num_waterfalls
FROM owner o LEFT JOIN waterfall w
     ON o.id = w.owner_id
GROUP BY o.id, o.name;

SELECT AVG(num_waterfalls)
  FROM owner_waterfalls_vw;

AVG(número_cascadas)
   --------------------
                  1.6
```

NOTA

Necesita privilegios CREATE para crear una vista. Si obtiene un error al ejecutar el código anterior, no tiene permiso para hacerlo y necesita hablar con el administrador de su base de datos.

Subconsultas frente a vistas

Tanto las subconsultas como las vistas representan los resultados de una consulta, , que luego pueden consultarse a su vez.

- Una *subconsulta* es temporal. Sólo existe mientras dura la consulta y es ideal para utilizarla una sola vez.
- Se guarda una *vista*. Una vez creada una vista, puede seguir escribiendo consultas que hagan referencia a la vista.

Crear una vista para guardar los resultados de una consulta

Utilice `CREATE VIEW` para guardar los resultados de una consulta como una vista. A continuación, la vista puede consultarse como una tabla.

Usando esta consulta:

```
SELECCIONAR *
FROM mi_tabla
WHERE country = 'US';

id país nombre
--- -------- ------
  1 EE.UU.   Anna
  2 EE.UU.   Emily
  3 EE.UU.   Molly
```

Crea una vista:

```
CREAR VISTA mi_vista COMO
SELECT *
FROM mi_tabla
WHERE country = 'US';
```

Consulta la vista:

```
SELECT * FROM mi_vista;

id país nombre
--- -------- ------
  1 EE.UU.   Anna
  2 EE.UU.   Emily
  3 EE.UU.   Molly
```

Visualizar las vistas existentes

El código de la Tabla 5-22 muestra cómo mostrar todas las vistas existentes en cada RDBMS.

Tabla 5-22. Código para mostrar vistas existentes

RDBMSCode	Code
MySQL	`MOSTRAR TABLAS COMPLETAS` `WHERE tipo_tabla = 'VIEW';`

Oracle	`SELECT`
	`nombre_vista`
	`FROM`
	`vistas_usuario;`

PostgreSQL	`SELECT nombre_tabla`
	`FROM esquema_informacion.vistas`
	`WHERE esquema_tabla NOT IN`
	`('information_schema', 'pg_catalog');`

SQL Server	`SELECT nombre_tabla` `FROM esquema_informacion.vistas;`

SQLite	`SELECT nombre`
	`FROM sqlite_master`
	`WHERE type = 'view';`

Actualizar una vista

Actualizar una vista es otra forma de decir sobrescribir una vista. El código de la Tabla 5-23 muestra cómo actualizar una vista en cada RDBMS.

Tabla 5-23. Código para actualizar una vista

RDBMS	Código
MySQL, Oracle, PostgreSQL	`CREATE OR REPLACE VIEW mi_vista AS SELECT * FROM mi_tabla WHERE país = 'CA';`
Servidor SQL	`CREATE OR ALTER VIEW my_view AS SELECT * FROM mi_tabla WHERE país = 'CA';`
SQLite	`DROP VIEW IF EXISTS mi_vista; CREATE VIEW mi_vista AS SELECT * FROM mi_tabla WHERE país = 'CA';`

Borrar una vista

Cuando ya no necesite una vista, puede eliminarla mediante una sentencia `DROP VIEW`:

```
DROP VIEW mi_vista;
```

ADVERTENCIA

No se puede deshacer la eliminación de una vista. Asegúrese al 100% de que desea eliminar una vista antes de eliminarla.

El lado positivo es que no hay pérdida de datos. Los datos siguen estando en la tabla original, y siempre se puede volver a crear la vista.

Gestión de transacciones

Una *transacción* permite actualizar una base de datos de forma más segura. Consiste en una secuencia de operaciones que se ejecutan como una sola unidad. O se ejecutan todas las operaciones o no se ejecuta ninguna, lo que también se conoce como *atomicidad*.

El siguiente código inicia una transacción antes de realizar cualquier cambio en las tablas de . Una vez ejecutadas las sentencias, no se realizan actualizaciones permanentes en la base de datos hasta que se confirman los cambios:

```
INICIAR TRANSACCIÓN;

INSERT INTO page_views (user_id, page)
   VALUES (525, 'home');
INSERT INTO page_views (user_id, page)
   VALUES (525, 'contact us');
DELETE FROM usuarios_nuevos WHERE user_id = 525;
UPDATE page_views SET page = 'solicitar
   información' WHERE page = 'contacta con
   nosotros';

COMPROMETERSE;
```

¿Por qué es más seguro utilizar una transacción?

Después de iniciar una transacción:

Las cuatro declaraciones se tratan como una unidad.
Imagina que ejecutas las tres primeras sentencias y, mientras lo haces, otra persona edita la base de datos de forma que tu cuarta sentencia no se ejecuta. Esto es problemático porque para actualizar la base de datos correctamente, las cuatro sentencias deben ejecutarse juntas. La transacción hace precisamente eso: requiere que las cuatro sentencias actúen como una unidad, de modo que o se ejecutan todas o no se ejecuta ninguna.

Puedes deshacer los cambios si es necesario.
Después de iniciar la transacción, puede ejecutar cada una de las sentencias y ver cómo afectarían a las tablas. Si todo parece correcto, puede finalizar la transacción y

asegure sus cambios con un COMMIT. Si algo parece mal y quieres volver a dejar las cosas como estaban antes de la transacción, puedes hacerlo con un ROLL BACK.

En general, si está actualizando una base de datos, es una buena práctica utilizar una transacción.

Las siguientes secciones cubren dos escenarios en los que el uso de una transacción es útil-uno que termina en un COMMIT para confirmar los cambios y el otro que termina en un ROLLBACK para deshacer los cambios.

Doble comprobación de los cambios antes de un COMMIT

Imagina que quieres borrar algunas filas de datos, pero quieres comprobar que se borran las filas correctas antes de eliminarlas definitivamente de la tabla.

El siguiente código muestra el proceso paso a paso de cómo utilizar una transacción en SQL para hacerlo.

1. Inicie una transacción.

```
-- MySQL y PostgreSQL
INICIAR TRANSACCIÓN;
o
COMIENZA;

-- SQL Server y SQLite
INICIAR TRANSACCIÓN;
```

   En *Oracle*, usted está esencialmente siempre en una transacción. Una transacción comienza cuando se ejecuta la primera sentencia SQL. Una vez finalizada una transacción (con un COMMIT o ROLL BACK), comienza otra cuando se ejecuta la siguiente sentencia SQL.

2. Visualiza la tabla que piensas modificar.

   En este momento se encuentra en modo transacción, lo que significa que no se realizará ningún cambio en la base de datos.

```
SELECT * FROM libros;

+------+----------------+
| id   | Título         |
+------+----------------+
|    1                  |
|    2 | Born a Crime |
|    3. Bossypants      |
+------+----------------+
```

3. Pruebe el cambio y vea cómo afecta a la tabla.

   Desea eliminar todos los títulos de libros con varias palabras. La siguiente sentencia `SELECT` le permite ver todos los títulos de libros multipalabra de la tabla.

```
SELECT * FROM libros WHERE título LIKE '% %';

+------+----------------+
| id   | Título         |
+------+----------------+
|    2 | Born a Crime |
+------+----------------+
```

   La siguiente sentencia `DELETE` utiliza la misma cláusula `WHERE` para eliminar ahora los títulos de libros con varias palabras de la tabla.

```
DELETE FROM libros WHERE título LIKE '% %';

SELECT * FROM libros;

+------+----------------+
| id   | Título         |
+------+----------------+
|    1                  |
|    3. Bossypants      |
+------+----------------+
```

   En este m o m e n t o todavía está en modo transacción, por lo que el cambio no se ha hecho permanente.

4. Confirme el cambio con `COMMIT`.

   Utilice `COMMIT` para bloquear los cambios. Después de este paso, ya no estarás en modo transacción.

```
COMPROMETERSE;
```

ADVERTENCIA

No se puede deshacer una transacción una vez que ha sido confirmada.

Deshacer cambios con ROLLBACK

Las transacciones son especialmente útiles para probar los cambios y deshacerlos si es necesario.

1. Inicie una transacción.

```
-- MySQL y PostgreSQL
INICIAR TRANSACCIÓN;
o
COMIENZA;

-- SQL Server y SQLite
INICIAR TRANSACCIÓN;
```

   En *Oracle*, usted está esencialmente siempre en una transacción. Una transacción comienza cuando se ejecuta la primera sentencia SQL. Una vez finalizada una transacción (con un `COMMIT` o `ROLL BACK`), comienza otra cuando se ejecuta la siguiente sentencia SQL.

2. Visualiza la tabla que piensas modificar.

   En este momento se encuentra en modo transacción, lo que significa que no se realizará ningún cambio en la base de datos.

```
SELECT * FROM libros;

+------+----------------+
| id   | Título         |
+------+----------------+
|    1                  |
|    2 | Born a Crime |
|    3. Bossypants      |
+------+----------------+
```

3. Pruebe el cambio y vea cómo afecta a la tabla.

Desea eliminar todos los títulos de libros con varias palabras. La siguiente sentencia DELETE borra accidentalmente toda la tabla (ha olvidado un espacio en '%%'). ¡Usted no quería que esto ocurriera!

```
DELETE FROM libros WHERE título LIKE '%%';

SELECT * FROM libros;

+------+----------------+
| id   | Título         |
+------+----------------+
```

Menos mal que en este momento todavía estás en modo transacción, para que el cambio no se haya hecho permanente.

4. Deshaga el cambio con ROLLBACK.

   En lugar de COMMIT, ROLLBACK los cambios. La tabla no se borrará. Después de este paso, ya no se encuentra en el modo de transacción y puede continuar con sus otras sentencias.

```
ROLLBACK;
```

CAPÍTULO 6

Tipos de datos

En una tabla SQL, cada columna sólo puede incluir valores de un único tipo de datos . En este capítulo se tratan los tipos de datos más utilizados, así como el modo y el momento de utilizarlos.

La siguiente sentencia especifica tres columnas junto con el tipo de datos de cada columna: `id` contiene valores enteros, `name` contiene valores de hasta 30 caracteres y `dt` contiene valores de fecha:

```
CREAR TABLA mi_tabla (
   id INT,
   nombre VARCHAR(30),
   dt FECHA
);
```

`INT`, `VARCHAR` y `DATE` son sólo tres de los muchos tipos de datos de SQL. La Tabla 6-1 lista cuatro categorías de tipos de datos, junto con subcategorías comunes. La sintaxis de los tipos de datos varía ampliamente según el RDBMS, y las diferencias se detallan en cada sección de este capítulo.

Tabla 6-1. Tipos de datos en SQL Tipos de datos en SQL

Numérico	Cadena	Fecha y hora	Otros
Entero (123) Decimal (1.23) Punto flotante (1.23e10)	Carácter ('hola') Unicode ('西瓜')	Fecha ('2021-12-01') Hora ('2:21:00') Fecha ('2021-12-01 2:21:00')	Boolean o (TRUE) Binario (imágenes, documentos, etc.)

La Tabla 6-2 enumera valores de ejemplo de cada tipo de datos para mostrar cómo se representan en SQL. Estos valores se denominan a menudo *literales* o *constantes*.

Tabla 6-2. Literales en SQL

Categoría	Subcategoría	Ejemplo Valores
Numérico	Entero	123 +123 -123
	Decimal	123.45 +123.45 -123.45
	Punto flotante	123.45E+23 123.45e-23
Cadena	Carácter	Gracias 'El combo es 39-6-27.'
	Unicode	N'Amélie' N'♥♥♥'
Fecha	Fecha	'2022-10-15' 15-OCT-2022" *(Oracle)*
	Hora	'10:30:00' '10:30:00.123456' '10:30:00 -6:00'
	Fecha y hora	'2022-10-15 10:30:00' 15-OCT-2022 10:30:00' *(Oracle)*

Categoría		Subcategoría	Ejemplo Valores
Otros	Booleano		TRUE FALSE
	Binario (los valores de ejemplo se muestran en hexadecimal)		`X'AB12'` *(MySQL, PostgreSQL)* `x'AB12'` *(MySQL, PostgreSQL)* `0xAB12` *(MySQL, SQL Server, SQLite)*

El literal NULL

Las celdas sin valor se representan mediante la palabra clave `NULL` (también conocida como el literal `NULL`), que no distingue entre mayúsculas y minúsculas (`NULL` = `Null` = `null`).

A menudo verá valores nulos en una tabla, pero nulo en sí no es un tipo de dato. Cualquier columna numérica, de cadena, de fecha y hora o de otro tipo puede incluir valores nulos dentro de la columna.

Cómo elegir un tipo de datos

A la hora de decidir el tipo de datos de una columna, es importante equilibrar el tamaño de almacenamiento y la flexibilidad en .

La Tabla 6-3 muestra algunos ejemplos de tipos de datos enteros. Tenga en cuenta que cada tipo de datos permite un rango diferente de valores y requiere una cantidad diferente de espacio de almacenamiento.

Tabla 6-3. Muestra de tipos de datos enteros

Tipo de datos permitidos	Gama	de valores	Tamaño de almacenamiento
`INT`	-2.147.483.648 a 2.147.483.647		4 bytes
`SMALLINT`	-32.768 a 32.767		2 bytes
`TINYINT`	0 a	255	1 byte

Imagine que tiene una columna de datos que contiene el número de alumnos de un aula:

```
15
25
50
70
100
```

Esta columna contiene datos numéricos, más concretamente, enteros. Puede elegir cualquiera de los tres tipos de datos enteros de la Tabla 6-3 para asignarlos a esta columna.

El caso de `INT`
: Si el espacio de almacenamiento no es un problema, entonces `INT` es una opción sencilla y sólida que funciona en todos los RDBMS.

El caso de `TINYINT`
: Dado que todos los valores están comprendidos entre 0 y 255, la elección de `TINYINT`
ahorraría espacio de almacenamiento.

El caso de `SMALLINT`
: Si más adelante se insertan en la columna recuentos de alumnos más elevados, `SMALLINT` permite una mayor flexibili- dad al tiempo que utiliza menos espacio que `INT`.

No existe una única respuesta correcta. El mejor tipo de datos para una columna depende tanto del espacio de almacenamiento como de la flexibilidad necesaria.

CONSEJO

Si ya ha creado una tabla pero desea cambiar el tipo de datos de una columna, puede hacerlo modificando la restricción de la columna con una sentencia `ALTER TABLE`. Encontrará más información en Modificación de una restricción, en el Capítulo 5.

Datos numéricos

En esta sección se presentan los valores numéricos para que tenga una idea de cómo se representan en SQL y, a continuación, se entra en detalle en los tipos de datos enteros, decimales y de coma flotante de .

Las columnas con datos numéricos pueden introducirse en funciones numéricas como `SUM()` y `ROUND()`, que se tratan en la sección Funciones numéricas del capítulo 7.

Valores numéricos

Los valores numéricos incluyen números enteros, d e c i m a l e s y de coma flotante.

Valores enteros

Los números sin decimal se tratan como enteros. El `signo +` es opcional.

```
123    +123-123
```

Valores decimales

Los decimales incluyen un punto decimal y se almacenan como valores exactos . El `signo +` es opcional.

```
123.45    +123.       45-123.45
```

Valores en coma flotante

Los valores en coma flotante utilizan la notación científica.

```
123.45E+23    123.45e-23
```

Estos valores se interpretan como $123{,}45 \times {}^{1023}$ y $123{,}45 \times {}^{10\text{-}23}$, respectivamente.

NOTA

Oracle permite una `F`, `f`, `D` o `d` al final para indicar `FLOAT` o `DOUBLE` (valor `FLOAT` más preciso):

```
123F            +123f-123,45D          123.45d
```

Tipos de datos enteros

El siguiente código crea una columna entera:

```
CREATE TABLE mi_tabla (
   mi_columna_integros
   INT
);

INSERT INTO mi_tabla VALUES
   (25),
   (-525),
   (2500252);

SELECT * FROM mi_tabla;

+-------------------+
| mi_columna_integral |
+-------------------+
                 |25 |
               |-525 |
             |2500252 |
+-------------------+
```

La Tabla 6-4 lista las opciones de tipos de datos enteros para cada RDBMS.

Tabla 6-4. Tipos de datos enteros Tipos de datos enteros

RDBMS	Tipo de datos	Gama de valores permitidos	Tamaño de almacenamiento
MySQL	TINYINT	de -128 a 127 0 a 255 (sin signo)	1 byte
	SMALLINT	de -32.768 a 32.767 0 a 65.535 (sin signo)	2 bytes
	MEDIUMINT	-8.388.608 a 8.388.607 0 a 16.777.215 (sin signo)	3 bytes
	INT o INTEGRO	-2.147.483.648 a 2.147.483.647 0 a 4.294.967.295 (sin signo)	4 bytes
	BIGINT	$^{-263}$ a ^{263} - 1 0 a ^{264} - 1 (sin signo)	8 bytes
Oracle	NÚMERO	$^{-10125}$ a ^{10125} - 1	1 a 22 bytes
PostgreSQL	SMALLINT	-32.768 a 32.7672 bytes	
	INT o INTEGRO	-2.147.483.648 a 2.147.483.647	4 bytes
	BIGINT	$^{-263}$ a ^{263} - 1	8 bytes
SQL Server	TINYINT	0 a	2551 byte
	SMALLINT	-32.768 a 32.7672 bytes	
	INT o INTEGRO	-2.147.483.648 a 2.147.483.647	4 bytes
	BIGINT	$^{-263}$ a ^{263} - 1	8 bytes
SQLite	INTEGRO	$^{-263}$ a ^{263} - 1 (si es más grande, cambiará a una REAL tipo de datos)	1, 2, 3, 4, 6 o 8 bytes

NOTA

MySQL permite tanto rangos con signo (enteros positivos y negativos) como rangos sin signo (sólo enteros positivos). El valor por defecto es el rango con signo. Para especificar un rango sin signo:

```
CREATE TABLE mi_tabla (
   mi_columna_integros INT UNSIGNED
);
```

PostgreSQL tiene un tipo de datos `SERIAL` que crea un entero auto incremental (1, 2, 3, etc.) en una columna. La Tabla 6-5 lista las opciones de `SERIAL`, cada una con un rango diferente.

Tabla 6-5. Opciones de serie en PostgreSQL

Tipo de datos	Gama de valores generados	Tamaño de almacenamiento
`SMALLSERIAL`	1 a 32.767	2 bytes
`SERIE`	1 a 2.147.483.647	4 bytes
`BIGSERIAL`	1 a 9.223.372.036.854.775.807	8 bytes

Tipos de datos decimales

Los números decimales también se conocen como números de *coma fija*. Incluyen un punto decimal y se almacenan como un valor exacto. Los datos monetarios (como `799,95`) suelen almacenarse como números decimales.

El siguiente código crea una columna decimal:

```
CREATE TABLE mi_tabla (
   mi_columna_decimal DECIMAL(5,2)
);

INSERT INTO mi_tabla VALUES
   (123.45),
   (-123),
   (12.3);
```

```
SELECT * FROM mi_tabla;
```

```
+------------------+
| mi_columna_decimal |
+------------------+
|123          .45 |
|-123        .00 |
|12            .30 |
+------------------+
```

Al definir el tipo de datos `DECIMAL(5,2)`:

- *5* es el número máximo de *dígitos totales* que se almacenan. Esto se denomina *precisión*.
- *2* es el número de dígitos a la *derecha del punto decimal*. Esto se denomina *escala*.

La Tabla 6-6 lista las opciones de tipos de datos decimales para cada RDBMS.

Tabla 6-6. Tipos de datos decimales Tipos de datos decimales

	RDBMSData TypeDígitos Por defecto		máximos permitidos
MySQL	DECIMAL o NÚMERO	Total: 65 Después del punto decimal: 30	`DECIMAL(10,0)`
Oracle	NÚMERO	Total: 38 Después del punto decimal: -84 a 127 (negativo significa antes del punto decimal)	0 dígitos después del punto decimal
PostgreSQL	DECIMAL o NÚMERO	Antes del punto decimal: 131,072 Después del punto decimal: 16,383	`DECIMAL(30,6)`
SQL Server	DECIMAL o NÚMERO	Total: 38 Después del punto decimal: 38	`DECIMAL(18,0)`
SQLite	NUMÉRICO	Sin	entradasNo por defecto

Tipos de datos en coma flotante

Los números de coma flotante son un concepto informático. Cuando un número tiene muchos dígitos, ya sea antes o después de un decimal

point, en lugar de almacenar todos los dígitos, los números de coma flotante sólo almacenan un número limitado de ellos para ahorrar espacio.

- Número: `1234.56789`
- Notación en coma flotante: `1.23 x 10^3`

Observará que el punto decimal "flotó" unos espacios a la izquierda y que se almacenó un valor *aproximado* (`1,23`), en lugar del valor original completo (`1234,56789`).

Existen dos tipos de datos de coma flotante:

- *Precisión única*: el número está representado por al menos 6 dígitos, con un rango completo de alrededor de 1E-38 a 1E+38
- *Doble precisión*: el número está representado por al menos 15 dígitos, con un rango completo de 1E-308 a 1E+308 El código siguiente crea una columna de coma flotante de precisión simple (`FLOAT`) y otra de doble precisión (`DOUBLE`) en :

```
CREATE TABLE mi_tabla (
   mi_columna_float FLOAT,
   mi_columna_doble DOUBLE
);

INSERT INTO mi_tabla VALUES
   (123.45, 123.45),
   (-12345.6789, -12345.6789),
   (1234567.890123456789, 1234567.890123456789);

SELECT * FROM mi_tabla;

+----------------+---------------------+
| mi_columna_flotante mi_columna_doble  |
+----------------+---------------------+
|123        .45 |123              .45 |
|-12345.  7 |-12345          .6789 |
|1234570    | 1234567.8901234567 |
```

```
+-----------------+----------------------+
```

ADVERTENCIA

Dado que los datos en coma flotante almacenan valores aproximados, las comparaciones y los cálculos pueden desviarse ligeramente de lo que cabría esperar.

Si sus datos siempre tendrán el mismo número de dígitos decimales, es mejor utilizar un tipo de datos de coma fija como `DECIMAL` para almacenar valores exactos en lugar de un tipo de datos de coma flotante.

La Tabla 6-7 enumera las opciones de tipos de datos de coma flotante para cada RDBMS.

Tabla 6-7. Tipos de datos de coma flotante

RDBMS	Tipo de datos	Rango de entrada	Tamaño de almacenamiento
MySQL	`FLOAT`	0 a 23 bits	4 bytes
	`FLOAT`	24 a 53 bits	8 bytes
	`DOBLE`	0 a 53 bits	bytes
Oracle	`BINARY_FLOAT`	Sin entradas	bytes
	`BINARY_DOUBLE`	Sin entradas	bytes
PostgreSQL	`REAL`	Sin entradas	4 bytes
	`DOBLE PRECISIÓN`	bytes	SQL Server
`REAL`		Sin entradas	4 bytes
	`FLOAT`	1 a 24 bits	4 bytes
	`FLOAT`	25 a 53 bits	8 bytes
SQLite	`REAL`	Sin entradas	8 bytes

NOTA

El tipo de datos `FLOAT` *de Oracle* NO es un número de coma flotante. En su lugar, `FLOAT` es equivalente a `NUMERIC`, que es un número decimal. Para un tipo de datos de coma flotante, debe utilizar `BINARY_FLOAT` o `BINARY_DOUBLE`.

Bits frente a Bytes frente a Dígitos

1 *bit es* la unidad más pequeña de almacenamiento. Puede tener un valor de `0` o 1. *1 byte* consta de 8 bits. Ejemplo de byte: 10101010.

Cada carácter está representado por un byte. La *cifra* `7` = `00000`111 en forma de bytes.

Cadena de datos

Esta sección presenta los valores de cadena para darle una idea de cómo se representan en SQL y, a continuación, entra en detalle en los tipos de datos char- acter y unicode.

Las columnas con datos de cadena pueden introducirse en funciones de cadena como `LENGTH()` y `REGEXP()` (expresión regular), que se tratan en la sección Funciones de cadena del capítulo 7.

Valores de cadena

Los valores de cadena son secuencias de caracteres que incluyen letras, números y caracteres especiales.

Conceptos básicos

La norma es encerrar los valores de cadena entre comillas simples:

```
'Esto es una cadena'.
```

Utilice dos comillas simples adyacentes cuando necesite incrustar una sola comilla en una cadena:

De nada.

SQL tratará las dos comillas simples adyacentes como una sola comilla dentro de la cadena y devolverá:

```
De nada.
```

CONSEJO

Como práctica recomendada, las comillas simples (`''`) deben utilizarse para encerrar valores de cadena, mientras que las comillas dobles (`""`) deben utilizarse para los identificadores (nombres de tablas, columnas, etc.).

Alternativas a las comillas simples

Si su texto contiene muchas comillas simples y desea utilizar un carácter diferente para denotar una cadena, Oracle y PostgreSQL le permiten hacerlo.

Oracle permite anteponer a una cadena una `Q` o `q`, seguida de cualquier carácter, luego la cadena y finalmente el carácter de nuevo:

```
Q'[Esto     es     una
cadena.]'  q'[Esto  es
una  cadena.]'  Q'|Esto
es una cadena.|'
```

PostgreSQL le permite rodear el texto con dos signos de dólar y un nombre de etiqueta opcional:

```
$$Esto es una cadena.$$
$mytag$Esto es una cadena.$mytag$
```

Secuencias de escape

MySQL y PostgreSQL soportan *secuencias de escape*, o una secuencia especial de texto que tiene significado. La Tabla 6-8 lista las secuencias de e s c a p e más comunes.

Tabla 6-8. Secuencias de escape comunes

Secuencia de Escape	Descripción
`\'`	Cita simple
`\t`	Tab
`\n`	Nueva línea
`\r`	Retorno de carro
`\b`	Retroceso
`\\`	Barra diagonal inversa

MySQL le permite incluir secuencias de escape dentro de una cadena utilizando el carácter `\`:

```
SELECCIONE 'hola', 'he\'llo', '\thello';

+-------+--------+--------------+
| hola | he'llo          |hello |
+-------+--------+--------------+
```

PostgreSQL le permite incluir secuencias de escape en cadenas si la cadena en general está precedida por una `E` o `e`:

```
SELECT 'hola', E'he\'llo', e'\thello';

----------+----------+---------------
          Hola.      |         hola
```

Las secuencias de escape sólo se aplican a las cadenas encerradas entre comillas simples y no a las cadenas encerradas entre signos de dólar.

Tipos de datos de caracteres

La forma más habitual de almacenar valores de cadena es utilizar tipos de datos de caracteres . El siguiente código crea una columna de caracteres variable que permite un máximo de 50 caracteres:

```
CREATE TABLE mi_tabla (
   mi_columna_caracter VARCHAR(50)
```

```
);
```

```
INSERT INTO mi_tabla VALUES
   ('Aquí hay algo de texto'),
   ('Y algunos números - 1 2 3 4 5'),
   ('¡Y algunos signos de puntuación!
   :)');

SELECT * FROM mi_tabla;

+------------------------------+
| mi_columna_carácter          |
+------------------------------+
| Aquí hay algo de texto.      |
| Y algunos números - 1 2 3 4 5 |
| Y algunos signos de puntuación. :)    |
+------------------------------+
```

Existen tres tipos principales de datos de caracteres:

`VARCHAR` *(carácter variable)*
: Este es el tipo de datos de cadena más popular. Si el tipo de datos es `VARCHAR(50)`, entonces la columna permitirá hasta 50 caracteres . En otras palabras, la longitud de la cadena es variable. En otras palabras, la longitud de la cadena es variable.

`CHAR` *(carácter)*
: Si el tipo de datos es `CHAR(5)`, entonces cada valor de la columna tendrá exactamente 5 caracteres. En otras palabras, la longitud de la cadena es fija. Los datos se rellenarán con espacios a la derecha para que tengan exactamente la longitud especificada. Por ejemplo, `'hi'` se almacenaría como `'hi '`.

`TEXT`
: A diferencia de `VARCHAR` y `CHAR`, `TEXT` no requiere entradas, lo que significa que no tiene que especificar una longitud para el texto. Es útil para almacenar cadenas largas, como un párrafo o más de texto.

La Tabla 6-9 enumera las opciones de tipos de datos de caracteres para cada RDBMS.

Tabla 6-9. Tipos de datos de caracteres

RDBMS	Tipo de datos	Rango de entrada	Por defecto	Tamaño de almacenamiento
MySQL	CHAR	0 a 255 caracteres	CHAR(1)	Varía
	VARCHAR	0 a 65.535 caracteres	Entrada necesaria	Varía
	TINY TEXTO	Sin entradas	Sin entradas	255 bytes
	TEXTO	Sin entradas	Sin entradas	65.535 bytes
	MEDIO TEXTO	Sin entradas	Sin entradas	16.777.215 bytes
	GRANDE TEXTO	Sin entradas	Sin entradas	4,294,967,295 bytes
Oracle	CHAR	De 1 a 2.000 caracteres	CHAR(1)	Varía
	VAR CHAR2	De 1 a 4.000 caracteres	Entrada necesaria	Varía
	LARGO	Sin entradas	Sin entradas	2 GB
PostgreSQL	CHAR	1 a 10.485.760 caracteres	CHAR(1)	Varía
	VARCHAR	1 a 10.485.760 caracteres	Entrada necesaria	Varía
	TEXTO	Sin entradas	Sin entradas	Varía
Servidor SQL	CHAR	De 1 a 8.000 bytes	Entrada necesaria	Varía
	VARCHAR	1 a 8.000 bytes, o máx.	Entrada necesaria	Varía, o hasta 2 GB

	TEXTO	Sin entradas	Sin entradas	2,147,483,647 bytes
SQLite	TEXTO	Sin entradas	Sin entradas	Varía

NOTA

Normalmente se utiliza `VARCHAR2` *de Oracle* en lugar de `VARCHAR`. Son idénticos en cuanto a funcionalidad, pero `VARCHAR` puede modificarse algún día, por lo que es más seguro utilizar `VARCHAR2`.

Tipos de datos Unicode

Los tipos de datos de caracteres suelen almacenarse como datos *ASCII*, pero también pueden almacenarse como datos *Unicode* si se necesita una biblioteca de caracteres más amplia .

Codificación ASCII frente a Unicode

Hay muchas formas de *codificar* los datos o, en otras palabras, de convertir los datos de en 0 y 1 para que los entienda un ordenador. La codificación por defecto que utiliza SQL se denomina *ASCII (American Standard Code for Information Interchange)*.

Con ASCII, hay ^{28} = 128 caracteres que se convierten en una serie de ocho 0's y 1's. Por ejemplo, el carácter ! corresponde a 00100001. Estos ocho 0 y 1 se conocen como *bytes* de datos.

Existen otros tipos de codificación además de ASCII, como *UTF (Unicode Transformation Format)*. Con Unicode, hay ^{221} caracteres:

- Los primeros ^{28} caracteres son los mismos que ASCII (! = `100001`).
- Otros caracteres incluyen caracteres asiáticos, símbolos matemáticos, emojis, etc.
- Aún no se han asignado valores a todos los caracteres.

El siguiente código muestra la diferencia entre VARCHAR y NVARCHAR (Unicode):

```
CREAR TABLA mi_tabla (
   ascii_texto VARCHAR(10),
   unicode_texto NVARCHAR(10)
);

INSERT INTO mi_tabla VALUES
   ('abc', 'abc'),
    (N'赵欣婉', N'赵欣婉');

SELECT * FROM mi_tabla;

+------------+ ---------------+
| ascii_text | unicode_text |
+------------+ ---------------+
| abc| abc    |
        | 赵欣婉      |
+------------+ ---------------+
```

NOTA

Al insertar datos Unicode de un archivo de texto en una columna NVARCHAR, los valores Unicode del archivo de texto no necesitan el prefijo N.

La Tabla 6-10 enumera las opciones de tipos de datos Unicode para cada RDBMS.

Tabla 6-10. Tipos de datos Unicode

RDBMS	Tipo de	datosDescripción
MySQL	NCHAR	Como CHAR, pero para datos Unicode
	NVARCHAR	Como VARCHAR, pero para datos
Unicode Oracle		NCHAR Como CHAR, pero para
datos Unicode		
	NVARCHAR2	Como VARCHAR2, pero para datos Unicode

RDBMS	Tipo de datos	Descripción
PostgreSQL	CHAR	Soporta datos Unicode
	VARCHAR	VARCHAR soporta datos Unicode
SQL Server	NCHAR	Like CHAR, pero para datos Unicode
	NVARCHAR	Como VARCHAR, pero para datos Unicode
SQLite	TEXT	TEXT soporta datos Unicode

Datos de fecha y hora

Esta sección presenta los valores datetime para darle una idea de cómo se representan en SQL, y luego entra en detalle en los tipos de datos datetime en cada RDBMS.

Las columnas con datos de fecha y hora pueden introducirse en funciones de fecha y hora como `DATEDIFF()` y `EXTRACT()`, que se tratan en la sección Funciones de fecha y hora del capítulo 7.

Valores de fecha y hora

Los valores de fecha y hora pueden adoptar la forma de fechas, horas o fechas y horas.

Valores de fecha

Una columna de fecha debe tener valores de fecha en el formato *AAAA- MM-DD*. En *Oracle*, el formato por defecto es *DD-MON-AAAA*.

El 15 de octubre de 2022 está escrito como:

```
'2022-10-15'
```

En *Oracle*, el 15 de octubre de 2022 se escribe como:

```
15-OCT-2022
```

Cuando se hace referencia a un valor de fecha en una consulta, se debe anteponer a la cadena una palabra clave `DATE` o `CAST` para

indicar a SQL que se trata de una fecha, como se muestra en la Tabla 6-11.

Tabla 6-11. Referencia a una fecha en una consulta

RDBMSCode	MySQL

	`SELECT FECHA '2021-02-25';`
	`SELECT FECHA('2021-02-25');`
	`SELECT CAST('2021-02-25' AS DATE);`

Oracle	`SELECT FECHA '2021-02-25' FROM dual;` `SELECT CAST('25-FEB-2021' AS DATE) FROM dual;`

PostgreSQL	`SELECT DATE '2021-02-25';`
	`SELECT DATE('2021-02-25');`
	`SELECT CAST('2021-02-25' AS DATE);`

SQL Server	`SELECT CAST('2021-02-25' AS DATE);`
SQLite	`SELECT FECHA('2021-02-25');`

NOTA

En *Oracle*, el formato de fecha después de la palabra clave `DATE` es diferente del formato de fecha dentro de la función `CAST`.

Además, en *Oracle, cuando se* realiza un cálculo o se busca una variable del sistema que sólo contiene una cláusula `SELECT`, es necesario añadir `FROM dual` al final de la consulta. `dual` es una tabla ficticia que contiene un único valor.

```
SELECT DATE '2021-02-25' FROM dual;
SELECT CURRENT_DATE FROM dual;
```

Si una columna contiene fechas con un formato diferente, como *MM/DD/AA*, puede aplicar una función de cadena a fecha para que SQL la reconozca como fecha.

Valores temporales

Una columna de tiempo debe tener valores de tiempo en el formato *hh:mm:ss*. 10:30 a.m. se escribe como:

```
'10:30:00'
```

También puede incluir segundos más granulares, de hasta seis decimales:

```
'10:30:12.345678'
```

También puede añadir una zona horaria. La hora estándar central también se conoce como `UTC-06:00`:

```
'10:30:12.345678 -06:00'
```

Cuando se hace referencia a un valor de tiempo en una consulta, se debe anteponer a la cadena una palabra clave `TIME` o `CAST` para indicar a SQL que se trata de un tiempo, como se muestra en la Tabla 6-12.

Tabla 6-12. Referencia a una hora en una consulta

RDBMSCode	MySQL

	`SELECT HORA '10:30';`
	`SELECT HORA('10:30');`
	`SELECT CAST('10:30' AS TIME);`

Oracle	`SELECT TIME '10:30:00' FROM dual;` `SELECT CAST('10:30' AS TIME) FROM dual;`
PostgreSQL	`SELECT TIEMPO '10:30';` `SELECT CAST('10:30' AS TIME);`
SQL Server	`SELECT CAST('10:30' AS TIME);`
SQLite	`SELECT HORA('10:30');`

NOTA

En *Oracle*, el formato de tiempo después de la palabra clave `TIME` debe incluir también los segundos.

Si una columna contiene horas con un formato diferente, como *mmss*, puede aplicar una función de cadena a hora para que SQL la reconozca como hora.

Valores de fecha y hora

Una columna datetime debe tener valores datetime en el formato *AAAA-MM-DD hh:mm:ss*. En *Oracle, el* formato por defecto es *DD- MON-YYYY hh:mm:ss*.

15 de octubre de 2022 a las 10:30 a.m. se escribe como:

```
'2022-10-15 10:30'
```

En *Oracle*, el 15 de octubre de 2022 a las 10:30 a.m. se escribe como:

```
15-OCT-2022 10:30
```

Cuando se hace referencia a un valor datetime en una consulta, se debe prefijar la cadena con una palabra clave DATETIME, TIMESTAMP, o CAST- para indicar a SQL que es un datetime, como se muestra en la Tabla 6-13.

Tabla 6-13. Referencia a una fecha y hora en una consulta

RDBMSCode	MySQL
	SELECT TIMESTAMP '2021-02-25 10:30'; SELECT TIMESTAMP('2021-02-25 10:30'); SELECT CAST('2021-02-25 10:30' AS DATETIME);
Oracle	SELECT TIMESTAMP '2021-02-25 10:30:00' DE doble; SELECT CAST('25-FEB-2021 10:30' AS TIMESTAMP) FROM dual;
PostgreSQL	SELECT TIMESTAMP '2021-02-25 10:30'; SELECT CAST('2021-02-25 10:30' AS
TIMESTAMP); SQL Server	SELECT CAST('2021-02-25 10:30' AS
DATETIME); SQLite	SELECT DATETIME('2021-02-25
10:30');	

NOTA

En *MySQL*, la palabra clave es `TIMESTAMP`, pero el tipo de datos es `DATETIME` dentro de la función `CAST`.

En *Oracle*, el formato de fecha después de la palabra clave `TIMESTAMP` es diferente del formato de fecha dentro de la función `CAST`. Además, el formato de hora después de la palabra clave `TIMESTAMP` debe incluir segundos, pero no es necesario dentro de la función `CAST`.

Si una columna contiene fechas con un formato diferente, como *MM/DD/AA mm:ss*, puede aplicar una función string to date o string to time para que SQL la reconozca como fecha y hora.

Tipos de datos Datetime

Hay muchas formas de almacenar valores de fecha y hora. Como los tipos de datos varían tanto, en esta sección hay una subsección separada para cada RDBMS.

Tipos de datos datetime de MySQL

El siguiente código crea cinco columnas datetime diferentes:

```
CREATE TABLE mi_tabla (
   dt DATE,
   tm TIEMPO,
   dttm FECHA,
   ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   yr AÑO
);

INSERT INTO my_table (dt, tm, dttm, yr)
   VALUES ('21-7-4', '6:30',
           2021, '2021-12-25 7:00:01');

+-----------+----------+ ----------------------+
| dt| tm| dttm           |
+-----------+----------+ ----------------------+
```

```
| 2021-07-04 | 06:30:00 | 2021-12-25 07:00:01 |
+------------+----------+ ----------------------+

+--------------------+ --------+
| ts| yr               |
+--------------------+ --------+
| 2021-01-29 12:56:20 | 2021 |
+--------------------+ --------+
```

La Tabla 6-14 enumera las opciones comunes del tipo de datos datetime en MySQL.

Tabla 6-14. Tipos de datos datetime de MySQL

Tipo de datos	Formato	Rango
	DATEYYY-MM-DD1000-01-01 a 9999-12-31	
HORA	hh:mm:ss-838	:59:59 a 838:59:59
FECHA	AAAA-MM-DD hh:mm:ss	1000-01-01 00:00:00 a 9999-12-31 23:59:59
TIMESTAMP	AAAA-MM-DD hh:mm:ss	1970-01-01 00:00:01 UTC a 2038-01-19 03:14:07 UTC
AÑO		AAAA0000 a 9999

NOTA

Tanto DATETIME como `TIMESTAMP` almacenan fechas y horas. La diferencia es que `DATETIME no tiene` una zona horaria asociada, mientras que `TIMESTAMP` almacena valores Unix (un punto específico en el tiempo) y se utiliza a menudo para anotar cuándo se crea o actualiza un registro.

Tipos de datos datetime de Oracle

El siguiente código crea cuatro columnas datetime diferentes:

```
CREATE TABLE mi_tabla (
   dt DATE,
   ts TIMESTAMP,
```

```
    ts_tz TIMESTAMP CON HORA LOCAL,
    ts_lc TIMESTAMP CON HORA LOCAL
);

INSERT INTO mi_tabla VALUES (
    '4-Jul-21', '4-Jul-21 6:30',
    4-Jul-21 6:30:45AM CST', '4-Jul-21 6:30'
);

DT            TS
------------ ------------------------------
              04-JUL-2104-JUL-21 06.30.00.000000 AM

TS_TZ
-----------------------------------
04-JUL-21 06.30.45.000000 AM CST

TS_LC
------------------------------
04-JUL-21 06.30.00.000000 AM
```

La Tabla 6-15 lista las opciones comunes de tipos de datos datetime en Oracle.

Tabla 6-15. Tipos de datos datetime de Oracle

Tipo de datos	Descripción
DATECuede	almacenar sólo la fecha o la fecha y la hora si el NLS_DATE_FORMAT se actualiza
	TIMESTAMPLike DATE, but adds fractional seconds (the default is six digits, but can go up to nine digits after the decimal point)
MARCA HORARIA CON ZONA HORARIA	Como TIMESTAMP, pero añade la zona horaria
MARCA HORARIA CON ZONA HORARIA LOCAL	Como TIMESTAMP WITH TIME ZONE, pero se ajusta en función de la zona horaria local del usuario.

Compruebe los formatos de fecha y hora en Oracle

El siguiente código comprueba los formatos actuales de fecha y hora:

```
SELECCIONAR valor
FROM nls_session_parameters
WHERE parámetro en ('NLS_DATE_FORMAT',
                    'NLS_TIMESTAMP_FORMAT');

VALOR
--------------------------
DD-MON-RR
DD-MON-RR HH.MI.SSXFF AM
```

Para cambiar el formato de la fecha o de la hora, puede modificar la opción
Parámetro `NLS_DATE_FORMAT` o `NLS_TIMESTAMP_FORMAT`.

El siguiente código cambia el actual `NLS_DATE_FORMAT = DD-MON-RR` para incluir también la hora:

```
ALTERAR SESIÓN
SET NLS_DATE_FORMAT = 'AAAA-MM-DD HH:MI:SS';
```

En la Tabla 7-27 se pueden encontrar otros símbolos comunes para fecha y hora, como `AAAA` para año y `HH` para hora: Especificadores de fecha y hora.

Tipos de datos datetime de PostgreSQL

El siguiente código crea cinco columnas datetime diferentes:

```
CREATE TABLE mi_tabla (
   dt DATE,
   tm TIEMPO,
   tm_tz HORA CON HUSO
   HORARIO, ts TIMESTAMP,
   ts_tz HORA CON HUSO HORARIO
);

INSERT INTO mi_tabla VALUES (
   '2021-7-4', '6:30', '6:30 CST',
```

```
   '2021-12-25 7:00:01', '2021-12-25 7:00:01 CST'
);

     dt     |    tm    |    tm_tz     |
------------+----------+ --------------+
 2021-07-04 | 06:30:00 | 06:30:00-06 |

         ts          |         ts_tz
---------------------+-----------------------
 2021-12-25 07:00:01 | 2021-12-25 07:00:01-06
```

La Tabla 6-16 enumera las opciones comunes de tipo de datos datetime en PostgreSQL.

Tabla 6-16. Tipos de datos datetime de PostgreSQL

Tipo de datos	Formato	Rango
FECHAAAA-MM-DD4713	a.C. a 5874897 d.C.	
HORA	hh:mm:ss00	:00:00 a 24:00:00
HORA CON HUSO HORARIO	hh:mm:ss+tz00	:00:00+1459 a 24:00:00-1459
FECHA	AAAA-MM-DD hh:mm:ss	4713 a.C. a 294276 d.C.
MARCA HORARIA CON ZONA HORARIA	AAAA-MM-DD hh:mm:ss+tz	4713 a.C. a 294276 d.C.

Tipos de datos datetime de SQL Server

El siguiente código crea seis columnas datetime diferentes:

```
CREATE TABLE mi_tabla (
   dt DATE,
   tm TIEMPO,
   dttm_sm SMALLDATETIME,
   dttm DATETIME,
   dttm2 DATETIME2,
   dttm_off DATETIMEOFFSET
);

INSERT INTO mi_tabla VALUES (
   '2021-7-4', '6:30', '2021-12-25 7:00:01',
```

```
    '2021-12-25 7:00:01', '2021-12-25 7:00:01',
    '2021-12-25 7:00:01-06:00'
);

dt             tm
-------------- --------------------
2021-07-04     06:30:00.00000000

dttm_sm
----------------------
2021-12-25 07:00:00

dttm
--------------------------
2021-12-25 07:00:01.000

dttm2
-----------------------------
2021-12-25 07:00:01.0000000

dttm_off
-----------------------------------
2021-12-25 07:00:01.0000000 -06:00
```

La Tabla 6-17 enumera las opciones comunes de tipo de datos datetime en SQL Server.

Tabla 6-17. Tipos de datos datetime de SQL Server

Tipo de datos	Formato	Rango
DATE	YYY-MM-DD	0001-01-01 a 9999-12-31
HORA	hh:mm:ss00	:00:00.0000000 a 23:59:59.9999999
SMALLDATETIME	AAAA-MM-DD hh:mm:ss	*Fecha*: 1900-01-01 a 2079-06-06 *Hora*: de 0:00:00 a 23:59:59
FECHA	AAAA-MM-DD hh:mm:ss	*Fecha*: 1753-01-01 a 9999-12-31 *Hora*: de 00:00:00 a 23:59:59.999
FECHA2	AAAA-MM-DD hh:mm:ss	*Fecha*: 0001-01-01 a 9999-12-31 *Hora*: de 00:00:00 a 23:59:59.9999999

Tipo de datos	Formato	Rango
DATETIMEOFFSET	AAAA-MM-DD hh:mm:ss +hh:mm	El desfase horario oscila entre -12:00 y +14:00

Tipos de datos datetime de SQLite

SQLite no tiene un tipo de datos datetime. En su lugar se utilizan `TEXT`, `REAL` o `INTEGER` puede utilizarse para almacenar valores de fecha y hora.

> **NOTA**
>
> Aunque en SQLite no existen tipos de datos datetime específicos, las funciones datetime como `DATE()`, `TIME()` y `DATETIME()` permiten trabajar con fechas y horas en SQLite.
>
> Encontrará más información en la sección Funciones de fecha y hora del capítulo 7.

El siguiente código muestra tres formas de almacenar valores datetime en SQLite:

```
CREATE TABLE mi_tabla (
   dt_texto TEXTO,
   dt_real REAL,
   dt_integro INTEGRO
);

INSERT INTO mi_tabla VALUES (
   '2021-12-25 7:00:01',
   '2021-12-25 7:00:01',
   '2021-12-25 7:00:01'
);

dt_text|dt_real
2021-12-25 7:00:01|2021-12-25 7:00:01
```

```
dt_entero
2021-12-25 7:00:01
```

La Tabla 6-18 lista las opciones del tipo de datos datetime en SQLite.

Tabla 6-18. Tabla 6-18. Tipos de datos datetime de SQLite

Tipo de datos	Descripción
TEXTO	Almacenado como una cadena en el formato *AAAA-MM-DD HH:MM:SS.SSS*
REAL	Almacenado como un número de día juliano, que es el número de días transcurridos desde el mediodía de Greenwich del 24 de noviembre de 4714 a.C.
INTEGRO	Almacenado como tiempo Unix, que es el número de segundos desde 1970-01-01 00:00:00 UTC

Otros datos

Hay muchos otros tipos de datos en SQL, incluyendo algunos que son específicos de cada RDBMS.

Algunos de ellos pertenecen a una de las categorías existentes de tipos de datos, pero capturan datos más detallados, como el tipo numérico `MONEY` o el tipo datetime `INTERVAL`.

Otros capturan datos más complejos, como datos geoespaciales que señalan una ubicación concreta en la Tierra o datos web almacenados en formatos JSON/XML.

Esta sección cubre dos tipos de datos adicionales: Datos booleanos y datos de archivos externos.

Datos booleanos

Los dos valores booleanos son `TRUE` y `FALSE`. No distinguen entre mayúsculas y minúsculas y deben escribirse sin comillas:

```
SELECCIONE VERDADERO, Verdadero, FALSO, Falso;

+------+------+-------+ ---------+
|1 |1 |0 |0 |
+------+------+-------+ ---------+
```

Tipos de datos booleanos

MySQL, *PostgreSQL* y *SQLite* soportan tipos de datos booleanos. El siguiente código crea una columna booleana:

```
CREATE TABLE mi_tabla (
   mi_columna_booleana BOOLEAN
);

INSERT INTO mi_tabla VALUES
   (TRUE),
   (falso),
   (1);

SELECT * FROM mi_tabla;

+------------------+
| mi_columna_booleana |
+------------------+
|1 |
|0 |
|1 |
+------------------+
```

Oracle y *SQL Server* no tienen tipos de datos booleanos, pero existen soluciones:

- En *Oracle*, utilice el tipo de datos `CHAR(1)` para mantener los valores `'T'` y `F'` o el tipo de datos `NUMBER(1)` para mantener los valores `1` y `0`.
- En *SQL Server*, utilice el tipo de datos `BIT`, que contiene `1`, `0` y Valores `nulos`.

Archivos externos (imágenes, documentos, etc.)

Si planea incluir imágenes (.jpg, .png, etc.) o documentos (.doc, .pdf, etc.) en una columna de datos, existen dos enfoques para hacerlo: almacenar enlaces a los archivos (más común) o almacenar los archivos como valores binarios.

Método 1: Almacenar enlaces a los archivos

Este suele ser el método recomendado si tus archivos pesan más de 1 MB cada uno. Como referencia, la foto media de un iPhone pesa unos pocos MB.

Los archivos se almacenarían fuera de la base de datos, lo que supondría una menor carga para ésta y, a menudo, un mejor rendimiento.

Pasos para almacenar enlaces a archivos:

1. Tenga en cuenta los nombres de ruta de los archivos en el sistema de archivos (`/Users/images/img_001.jpg`).
2. Cree una columna que almacene cadenas, como `VARCHAR(100)`.
3. Inserte los nombres de las rutas en la columna.

Método 2: Almacenar los archivos como valores binarios

Este suele ser el método recomendado si sus archivos son de menor tamaño.

Los archivos se almacenan dentro de la base de datos, lo que facilita las copias de seguridad.

Pasos para almacenar valores binarios:

1. Convierte los archivos a binario (si abres un archivo binario, parecerá una secuencia aleatoria de caracteres, como `Z™≈jhJcE Ät, ÷mfPfõrà`).
2. Crear una columna que almacene valores binarios, como `BLOB`.
3. Inserta los valores binarios en la columna.

Valores binarios y hexadecimales

Los datos binarios representan los valores brutos que un ordenador inter- preta. A menudo se muestran en una forma más compacta y legible para el ser humano, denominada *hexadecimal*.

- Carácter: `a`
- Valor binario equivalente: `01100001`
- Valor hexadecimal equivalente: `61`

Los hexadecimales convierten 1 y 0 en un sistema numérico de 16 símbolos (0-9 y A-F). Los hexadecimales van precedidos de `X`, `x` o `0x`:

```
SELECT X'AF12', x'AF12', 0xAF12;

+----------+----------+ ----------+
| 0xAF12 0xAF12        0xAF12
+----------+----------+ ----------+
```

MySQL admite los tres formatos. *PostgreSQL* admite los dos primeros formatos. *SQL Server* y *SQLite soportan* el tercer formato.

En *Oracle*, aunque no se puede mostrar fácilmente un valor hexadecimal, se puede utilizar la función `TO_NUMBER` para mostrar un valor hexadecimal como un número: `SELECT TO_NUMBER('AF12', 'XXXX') FROM dual;` la `X` representa la notación hexadecimal.

Tipos de datos binarios

El siguiente código crea una columna de datos binarios:

```
CREAR TABLA mi_tabla (
   mi_columna_binaria
   BLOB
);

INSERT INTO mi_tabla VALUES
   ('a'),
   ('aaa'),
   ('ae$ iou');

SELECT * FROM mi_tabla;
```

```
+--------------------------------------+
| mi_columna_binaria                   |
+--------------------------------------+
| 0x61                                 |
| 0x616161                             |
| 0x61652420696F75                     |
+--------------------------------------+
```

En *MySQL*, *Oracle* y *SQLite, el* tipo de datos binarios más común es `BLOB`.

En *PostgreSQL*, utilice `bytea` en su lugar.

En *SQL Server*, utilice `VARBINARY` (como `VARBINARY(100)`) en su lugar.

NOTA

En *Oracle* y *SQL Server*, la cadena `ae$ iou` no se reconoce automáticamente como un valor binario y debe convertirse en uno antes de insertarse en una tabla.

```
-- Oracle
SELECT RAWTOHEX('ae$ iou') FROM dual;

-- SQL Server
SELECT CONVERT(VARBINARY, 'ae$ iou');
```

La Tabla 6-19 lista las opciones de tipos de datos binarios para cada RDBMS.

Tabla 6-19. Tipos de datos binarios

RDBMS	Tipo de datos	Descripción	Rango de entrada	Tamaño de almacenamiento
MySQL	BINARIO	Cadena binaria de longitud fija en la que los valores se rellenan con 0 a la derecha para obtener e l tamaño exacto.	0 a 255 bytes	Varía
	VARBINARY	Longitud variable cadena binaria	0 a 65,535 bytes	Varía
	TINY	Tiny Binary Grande	Sin entradas	255 bytes
	BLOB	OBJETO		
	BLOB	Objeto binario grande	Sin entradas	65.535 bytes
	BLOB MEDIO	Binario medio Objeto grande	Sin entradas	16.777.215 bytes
	GRANDE BLOB	Grande Binario Grande OBJETO	Sin entradas	4,294,967,295 bytes
Oracle	RAW	Cadena binaria de longitud variable	1 a 32,767 bytes	Varía
	LARGO RAW	RAW más grande	Sin entradas	2 GB
	BLOB	Más grande LONG RAW	Sin entradas	4 GB
PostgreSQL	BYTEA	Cadena binaria de longitud variable	Sin entradas	1 o 4 bytes más la cadena binaria real

RDBMS	Tipo de datos	Descripción	Rango de entrada	Tamaño de almacenamiento
Servidor SQL	`BINARIO`	Cadena binaria de longitud fija en la que los valores s e rellenan con 0 para obtener e l tamaño exacto.	De 1 a 8.000 bytes	`Varía`
	VARBINARY	Longitud variable cadena binaria	De 1 a 8.000 bytes, o max	`Varía`, o hasta 2 GB
SQLite	BLOB	Objeto binario grande	Sin entradas	Almacenado exactamente como se introdujo

CAPÍTULO 7

Operadores y funciones

Los operadores y las *funciones* se utilizan para realizar cálculos, com- paraciones y transformaciones dentro de una sentencia SQL. Este capítulo proporciona ejemplos de código para operadores y funciones de uso común.

La siguiente consulta destaca cinco operadores (`+`, `=`, `OR`, `BETWEEN`, `AND`) y dos funciones (`UPPER`, `YEAR`):

```
-- Aumentos salariales para los empleados
SELECT nombre, tarifa_paga + 5 COMO
nueva_tarifa_paga FROM empleados
WHERE UPPER(title) = 'ANALISTA'
      OR YEAR(fecha_inicio) ENTRE 2016 Y 2018;
```

Operadores frente a funciones

Los operadores son símbolos o palabras clave que realizan un cálculo o una comparación. Los operadores se encuentran en las cláusulas `SELECT`, `ON`, `WHERE` y `HAVING` de una consulta.

Las funciones reciben cero o más entradas, aplican un cálculo o una transformación y generan un valor. Las funciones pueden encontrarse en dentro de las cláusulas `SELECT`, `WHERE` y `HAVING` de una consulta.

Además de las sentencias SELECT, los operadores y funciones también pueden utilizarse en las sentencias INSERT, UPDATE y DELETE.

Este capítulo incluye una sección sobre Operadores y cinco secciones sobre funciones: Funciones Agregadas, Funciones Numéricas, Funciones de Cadena, Funciones Fecha Hora y Funciones Nulas.

La Tabla 7-1 enumera los operadores comunes y la Tabla 7-2 enumera las funciones comunes de .

Tabla 7-1. Operadores comunes

Operadores lógicos	Operadores de comparación (símbolos)	Operadores de comparación (palabras clave)	Operadores matemáticos
Y	=	ENTRE	+
O	!=, <>	EXISTE	-
NO	<	EN	*
	<=	IS NULL	/
	>	COMO	%
	v=		

Tabla 7-2. Funciones comunes

Agregado Funciones	Numérico Funciones	Cadena Funciones	Fecha y hora Funciones	Nulo Funciones
CONTAR()	ABS()	LENGTH()	CURRENT_	COA
SUMA()	SQRT()	TRIM()	FECHA	LESCE()
AVG()	LOG()	CONCAT()	CURRENT_	
MIN()	REDONDO()	SUBSTR()	TIEMPO	
MAX()	CAST()	REGEXP()	DATEDIFF()	
			EXTRACTA()	
			CONVERTIR()	

Operadores

Los operadores pueden ser símbolos o palabras clave. Pueden realizar cálculos (+) o comparaciones (BETWEEN). Esta sección

describe los operadores disponibles en SQL.

Operadores lógicos

Los operadores lógicos se utilizan para modificar condiciones, cuyo resultado en es TRUE, FALSE o NULL. Los operadores lógicos del bloque de código (NOT, AND, OR) aparecen en negrita:

```
SELECCIONAR *
DE empleados
WHERE fecha_inicio NO ES NULA
      AND (title = 'analista' OR pay_rate < 25);
```

CONSEJO

Cuando se utilizan AND y OR para combinar varias sentencias condicionales, conviene indicar claramente el orden de las operaciones con paréntesis: ().

La Tabla 7-3 enumera los operadores lógicos en SQL.

Tabla 7-3. Operadores lógicos Operadores lógicos

Operador	Descripción
AND	Devuelve TRUE si ambas condiciones son TRUE. Devuelve FALSE si cualquiera de ellas es FALSE. En caso contrario, devuelve NULL.
OR	Devuelve TRUE si cualquiera de las dos condiciones es TRUE. Devuelve FALSE si ambas son FALSE. En caso contrario, devuelve NULL.
NO	Devuelve TRUE si la condición es FALSE. Devuelve FALSE si es TRUE. Devuelve NULL en caso contrario.

Imagina que hay una columna llamada nombre. La Tabla 7-4 muestra cómo se evaluarían los valores de la columna en un estado condicional- ment sin NOT y con NOT.

Tabla 7-4. Ejemplo NOT

nombre	nombre IN ('Henry', Harper")	name NOT IN ('Henry', Harper")
Henry	TRUE	FALSO
Lily	FALSO	TRUE
NULO	NULL	NULL

Imagine que hay dos columnas llamadas nombre y edad. La Tabla 7-5 muestra cómo se evaluarían los valores de las columnas en una sentencia con- dicional con un AND y con un OR.

Tabla 7-5. Ejemplo AND y OR

nombre	edad	nombre = Henry	edad > 3	nombre = Henry AND edad > 3	nombre = Henry' O edad > 3 años
Henry	5	TRUE	TRUE	TRUE	TRUE
Henry	1	TRUE	FALSO	FALSO	TRUE
Lily	2	FALSO	FALSO	FALSO	FALSO
Henry	NULL	TRUE	NULL	NULL	TRUE
Lily	NULL	FALSO	NULL	FALSO	NULL

Operadores de comparación

Los operadores de comparación se utilizan en predicados.

Operadores frente a predicados

Los predicados son comparaciones que incluyen un *operador*:

- El predicado `edad = 35` incluye el operador `=`.
- El predicado `COUNT(id) < 20` incluye el operador <.

Los predicados también se conocen como sentencias condicionales. Estos
se evalúan en cada fila de una tabla y dan como resultado un valor de VERDADERO, FALSO o NULO.

Los operadores de comparación del bloque de código (`IS NULL`, `=`, `BETWEEN`) aparecen en negrita:

```
SELECCIONAR *
DE empleados
WHERE fecha_inicio NO ES NULA
      AND (title = 'analista')
      OR pay_rate BETWEEN 15 AND 25);
```

La Tabla 7-6 enumera los operadores de comparación que son símbolos y La Tabla 7-7 enumera los operadores de comparación que son palabras clave.

Tabla 7-6. Operadores de comparación (símbolos)

Descripción del operador	
`=`	Pruebas de igualdad
`!=, <>`	Pruebas de desigualdad
`<`	Pruebas para menos de
`<=`	Comprueba si es menor o igual que
`>`	Pruebas de mayor que
`>=`	Comprueba si es mayor o igual que

NOTA

MySQL también permite `<=>`, que es una prueba de igualdad a prueba de nulos.

Cuando se utiliza `=`, si se comparan dos valores y uno de ellos es `NULL`, el valor resultante es `NULL`.

Cuando se utiliza `<=>`, si se comparan dos valores y uno de ellos es `NULL`, el valor resultante es `0`. Si ambos son `NULL`, el valor resultante es `1`.

Tabla 7-7. Operadores de comparación (palabras clave)

Operator	Description
`BETWEEN`	Comprueba si un valor se encuentra dentro de un rango determinado
`EXISTS`	Comprueba si existen filas en una subconsulta
`EN`	Comprueba si un valor está contenido en una lista de valores
`IS NULL`	Comprueba si un valor es nulo o no
`LIKE`	Comprueba si un valor coincide con un patrón simple

NOTA

El operador `LIKE` se utiliza para encontrar patrones sencillos, como por ejemplo, texto que empiece por la letra `A`. Encontrará más información en la sección `LIKE`.

Las expresiones regulares se utilizan para buscar patrones más complejos, como extraer cualquier texto situado entre dos signos de puntuación. Encontrará más información en la sección de expresiones regulares.

Cada operador de comparación de palabras clave se explica detalladamente en las secciones siguientes.

ENTRE

Utilice `ENTRE` para comprobar si un valor se encuentra dentro de un intervalo. `BETWEEN` es una combinación de `>=` y `<=`. El menor de los dos valores siempre debe escribirse primero, con el operador `AND` separando los dos.

Encontrar todas las filas en las que las edades son mayores o iguales que 35 y menores o iguales que 44:

```
SELECCIONAR *
FROM mi_tabla
cuando la edad esté comprendida entre 35 y 44 años;
```

Para encontrar todas las filas en las que las edades son inferiores a 35 o superiores a 44:

```
SELECCIONAR *
FROM mi_tabla
cuando la edad no esté comprendida entre 35 y 44 años;
```

EXISTE

Utilice EXISTS para comprobar si una subconsulta devuelve resultados o no. Normalmente, la subconsulta hace referencia a otra tabla.

La siguiente consulta devuelve los empleados que también son clientes:

```
SELECT e.id, e.name
FROM empleados e
WHERE EXISTS (SELECT *
              DE clientes c
              WHERE c.email = e.email);
```

EXISTS frente a JOIN

La consulta EXISTS también podría escribirse con un JOIN:

```
SELECCIONAR *
FROM empleados e INNER JOIN clientes c
     ON e.email = c.email;
```

Se prefiere un JOIN cuando se desea obtener valores de ambas tablas (SELECT *).

Es preferible utilizar EXISTS cuando se desea obtener valores de una única tabla (SELECT e.id, e.name). Este tipo de consulta se denomina a veces *semi-join*. EXISTS también es útil cuando la segunda tabla tiene filas duplicadas y sólo le interesa saber si una fila existe o no.

La siguiente consulta devuelve los clientes que nunca han realizado una compra:

```
SELECT c.id, c.name
FROM clientes c
WHERE NOT EXISTS (SELECT *
                  FROM pedidos o
                  WHERE o.email = c.email);
```

EN

Utilice `IN` para comprobar si un valor se encuentra dentro de una lista de valores. La siguiente consulta devuelve los valores de unos cuantos empleados:

```
SELECCIONAR *
DE empleados
WHERE e.id IN (10001, 10032, 10057);
```

La siguiente consulta devuelve los empleados que no han disfrutado de un día de vacaciones:

```
SELECT e.id
FROM empleados e
WHERE e.id NOT IN (SELECT v.emp_id
                   DESDE vacaciones v);
```

ADVERTENCIA

Cuando se utiliza `NOT IN`, si hay un solo valor `NULL` en la columna de la subconsulta (`v.emp_id` en este caso), la subconsulta nunca será `TRUE`, lo que significa que no se devolverá ninguna fila .

Si hay valores potencialmente `NULL` en la columna de la subconsulta, es mejor utilizar `NOT EXISTS`:

```
SELECT e.id
FROM empleados e
WHERE NOT EXISTS (SELECT *
                  DESDE vacaciones v
                  WHERE v.emp_id = e.id);
```

IS NULL

Utilice `IS NULL` o `IS NOT NULL` para comprobar si un valor es nulo o no.

La siguiente consulta devuelve los empleados que no tienen jefe:

```
SELECCIONAR *
DE empleados
WHERE manager IS NULL;
```

La siguiente consulta devuelve los empleados que tienen un responsable:

```
SELECCIONAR *
DE empleados
WHERE manager IS NOT NULL;
```

COMO

Utilice `LIKE` para hacer coincidir un patrón simple. El signo de porcentaje (`%`) es un comodín de que significa uno o más caracteres.

He aquí una tabla de muestra:

```
SELECT * FROM mi_tabla;

+------+ --------------------+
| id| txt                    |
+------+ --------------------+
     |Eres genial.           |
|Gracias.                    |
|3    | Pensando en ti.      |
     |Estoy 100% seguro. |
+------+ --------------------+
```

Busca todas las filas que *contengan* el término `tú`:

```
SELECCIONAR *
FROM mi_tabla
WHERE txt LIKE '%you%';
```

```
-- Resultados de MySQL, SQL Server y SQLite
+------+ ------------------+
| id| txt                 |
+------+ ------------------+
|Eres genial.             |
|Gracias.                 |
|3    | Pensando en ti. |
+------+ ------------------+

-- Resultados de Oracle y PostgreSQL
+------+ ------------------+
| id| txt                 |
+------+ ------------------+
|Gracias.                 |
|3    | Pensando en ti. |
+------+ ------------------+
```

En *MySQL*, *SQL Server* y *SQLite*, el patrón no distingue entre mayúsculas y minúsculas. tive. Tanto `You` como `you` son capturados por `'%you%'`.

En *Oracle* y *PostgreSQL*, el patrón distingue entre mayúsculas y minúsculas. `Sólo`
es capturado por `'%you%'`.

Busca todas las filas que *empiecen por* el término `Tú`:

```
SELECCIONAR *
FROM mi_tabla
WHERE txt LIKE 'You%';

+------+ ----------------+
| id| txt               |
+------+ ----------------+
     |Eres genial. |
+------+ ----------------+
```

Utilice `NOT LIKE` para devolver filas que no contengan los caracteres.

En lugar del signo de porcentaje (`%`) para que coincida con uno o más `caracteres`, puede utilizar el guión bajo (`_`) para que coincida exactamente con un carácter .

ADVERTENCIA

Dado que % y _ tienen un significado especial cuando se utilizan con LIKE, si desea buscar esos caracteres reales, tendrá que añadir la palabra clave ESCAPE.

El código siguiente busca todas las filas que contienen el %. símbolo:

```
SELECCIONAR *
FROM mi_tabla
WHERE txt LIKE '%!%%' ESCAPE '!';

+------+ --------------------+
| id| txt                    |
+------+ --------------------+
     |Estoy 100% seguro. |
+------+ --------------------+
```

Después de la palabra clave ESCAPE, hemos declarado ! como carácter de escape, de modo que cuando se pone ! delante del % medio en
%!%%, !% se interpreta como %.

LIKE es útil cuando se busca una cadena de caracteres en particular. Para búsquedas de patrones más avanzadas, puede utilizar expresiones regulares, que se tratan en la sección de expresiones regulares más adelante en este capítulo.

Operadores matemáticos

Los operadores matemáticos son símbolos matemáticos que pueden utilizarse en SQL. El operador matemático en el bloque de código (/) está en negrita:

```
SELECT salario / 52 AS
salario_semanal FROM mi_tabla;
```

La Tabla 7-8 lista los operadores matemáticos en SQL.

Tabla 7-8. Operadores matemáticos

Operador	Descripción
+	Adición
-	Resta
*	Multiplicación
/	División
% (MOD en *Oracle*)	Módulo (resto)

NOTA

En *PostgreSQL*, *SQL Server* y *SQLite*, dividir un entero por un entero da como resultado un número entero:

```
SELECCIONE
15/2; 7
```

Si desea que el resultado incluya decimales, puede dividir por un decimal o utilizar la función `CAST`:

```
SELECCIONAR
15/2.0; 7.5

 -- PostgreSQL y SQL Server
SELECT CAST(15 AS DECIMAL) /
       CAST(2 COMO
       DECIMAL); 7.5

-- SQLite
SELECT CAST(15 COMO REAL)
       / CAST(2 COMO
       REAL);
7.5
```

Otros operadores matemáticos son:

- *Operadores bit a bit* como `&` (AND), `|` (OR) y `^` (XOR) para trabajar con bits (valores 0 y 1).
- *Operadores de asignación* como `+=` (añadir iguales) y `-=` (sub- tracto iguales) para actualizar valores en una tabla.

Funciones agregadas

Una función de *agregado* realiza un cálculo sobre muchas filas de datos y da como resultado un único valor. En la Tabla 7-9 se enumeran las cinco funciones de agregación básicas de en SQL.

Tabla 7-9. Funciones básicas de los agregados

Descripción de la función	
COUNT()	Cuenta el número de valores
SUM()	Calcula la suma de una columna
AVG()	Calcula la media de una columna
MIN()	Calcula el mínimo de una columna
MAX()	Obtiene el máximo de una columna

Las funciones de agregación aplican cálculos a los valores no nulos de una columna. La única excepción es `COUNT(*)`, que cuenta *todas las* filas, incluidos los valores nulos.

También puede agregar varias filas en una sola lista utilizando funciones como `ARRAY_AGG`, `GROUP_CONCAT`, `LISTAGG` y `STRING_AGG`. Encontrará más información en la sección Agregar filas en un único valor o lista del capítulo 8.

NOTA

Oracle admite funciones de agregación adicionales como la mediana (`MEDIAN`), la moda (`STATS_MODE`) y la desviación estándar (`STDDEV`).

Las funciones agregadas (en negrita en el ejemplo) se encuentran en la sección
Cláusulas `SELECT` y `HAVING` de una consulta:

```
SELECT COUNT(*) AS total_rows,
       AVG(edad) AS edad_media
FROM mi_tabla;

SELECT región, MIN(edad), MAX(edad)
FROM mi_tabla
GROUP BY región
HAVING MIN(edad) < 18;
```

ADVERTENCIA

Si decide incluir columnas agregadas y no agregadas en la secuencia `SELECT`, *debe* incluir todas las columnas no agregadas en la cláusula `GROUP BY` (`región` en el ejemplo anterior).

Algunos RDBMS arrojarán un error si no lo hace. Otros RDBMS (como *SQLite*), no arrojarán un error y permitirán que la sentencia se ejecute, aunque los resultados devueltos sean *inexactos*. Es una buena práctica volver a comprobar los resultados para asegurarse de que tienen sentido.

MÍNIMO/MÁXIMO frente a MÍNIMO/MÁXIMO

Las funciones `MIN` y `MAX` encuentran los valores más pequeños y más grandes dentro de una columna.

Las funciones `MENOR` y `MAYOR` encuentran los valores más pequeños y más grandes de dentro de una fila. Los valores de entrada pueden ser numéricos, de cadena o de fecha y hora. Si un valor es `NULL`, la función devuelve `NULL`.

La siguiente tabla muestra el total de kilómetros recorridos cada trimestre, y la consulta encuentra los kilómetros recorridos en el mejor trimestre:

```
SELECT * FROM cabra;

+--------+------+------+------+ -----------+
| Nombre | q1   | q2   | q3   | q4   |
+--------+------+------+------+ -----------+
| Ali    | 100 | 200 | 150 | NULL |
| Perno  | 350 | 400 | 380 | 300 |
| Jordan | 200 | 250 | 300 | 320 |
+--------+------+------+------+ -----------+

SELECCIONE nombre, MAYOR(q1, q2, q3, q4)
              AS más_millas
DE cabra;

+--------+--------------+
| Nombre | most_miles |
+--------+--------------+
| Ali    |       NULL
| Perno  |        400 |
| Jordania |      320 |
+--------+--------------+
```

Funciones numéricas

Las funciones numéricas pueden aplicarse a columnas con tipos de datos numéricos. Esta sección cubre las funciones numéricas comunes en SQL.

Aplicar funciones matemáticas

Existen múltiples tipos de cálculos matemáticos en SQL:

Operadores matemáticos
: Cálculos con símbolos como `+`, `-`, `*`, `/` y `%`.

Funciones agregadas
: Cálculos que resumen toda una columna de datos en un único valor, como `COUNT`, `SUM`, `AVG`, `MIN` y `MAX`.

Funciones matemáticas
: Cálculos utilizando palabras clave que se aplican a cada fila de datos como `SQRT`, `LOG`, y más que se enumeran en la Tabla 7-10

NOTA

SQLite sólo admite la función `ABS`. Las demás funciones matemáticas deben activarse manualmente. Encontrará más información en la página de funciones matemáticas del sitio web de SQLite.

Tabla 7-10. Funciones matemáticas

Categoría	Función	Descripción	Código	Resultado
Positivos y negativos	ABS	Valor absoluto	`SELECT ABS(-5);`	5
Valores	SIGNO	Devuelve -1, 0, o 1 en función de si un número es negativo, cero o positivo	`SELECCIONE SIGNO(-5);`	1

Categoría	Función	Descripción	Código	Resultado
Exponentes y Logaritmos	POTENCIA	*x* elevado a la potencia de *y*	`SELECT POWER(5,2);`	25
	SQRT	Raíz cuadrada	`SELECCIONE SQRT(25);`	5
	EXP	*e* (=2.71828) elevado a la potencia de *x*	`SELECCIONE EXP(2);`	7.389
	REGISTRO (`LOG(y,x)` en *SQL Server*)	Log de *y* base *x*	`SELECT LOG(2,10); SELECT LOG(10,2);`	3.322
	LN (`LOG` en *SQL Servidor*)	Log natural (base *e*)	`SELECT LN(10); SELECT LOG(10);`	2.303
	LOG10 (`LOG(10,x)` en *Oracle*)	Log base 10	`SELECT LOG10(100); SELECT LOG(10,100) DE doble;`	2
Otros	MOD (`x%y` en *SQL Servidor*)	Resto de *x* / *y*	`SELECT MOD(12,5); SELECT 12%5;`	2
	PI (no disponible en *Oráculo*)	Valor de pi	`SELECCIONE PI();`	3.14159
	COS, SIN, etc.	Coseno, seno y otras funciones trigonométricas (entrada en radianes)	`SELECCIONE COS(.78);`	0.711

Generar números aleatorios

La Tabla 7-11 muestra cómo generar un número aleatorio en cada RDBMS de . En algunos casos, puede introducir una *semilla* para que el número a l e a t o r i o generado sea el mismo cada vez.

Tabla 7-11. Generador de números aleatorios

RDBMS resultados		CódigoRango de
MySQL, Servidor SQL	`SELECCIONE RAND();` `-- Semilla opcional SELECT RAND(22);`	0 a 1
Oracle	`SELECT DBMS_RANDOM.VALUE DE doble;` `SELECT DBMS_RANDOM.RANDOM DE doble;`	0 a 1 -2E31 a +2E31
PostgreSQL	`SELECT RANDOM();`	0 a 1
SQLite	`SELECT RANDOM();`	-9E18 a +9E18

La función de números aleatorios se utiliza a veces para devolver unas cuantas filas aleatorias de una tabla. Aunque no es la consulta más eficiente (ya que hay que ordenar la tabla), es un truco rápido:

```
-- Devuelve 5 filas
aleatorias SELECT *
FROM mi_tabla
ORDER BY RANDOM()
LIMIT 5;
```

Oracle y *SQL Server* permiten tomar muestras aleatorias de una tabla:

```
-- Devuelve aleatoriamente el 20% de las
filas de Oracle SELECT *
FROM mi_tabla
MUESTRA(20);

-- Devuelve 100 filas aleatorias en SQL
```

```
Server SELECT *
```

```
FROM mi_tabla
TABLASMUESTRA(100 FILAS);
```

Redondear y truncar números

La Tabla 7-12 muestra las distintas formas de redondear números en cada RDBMS de .

Tabla 7-12. Opciones de redondeo

Función	Descripción	Código	Salida
CEIL (CEILING en *SQL Servidor*)	Redondea al entero más próximo	SELECCIONE CEIL(98,7654); SELECT CEILING(98,7654);	99
SUELO	Redondea hacia abajo al entero más próximo	SELECCIONE FLOOR(98.7654);	98
REDONDEAR	Redondea a un número específico de decimales, por defecto a 0 decimales	SELECT ROUND(98.7654,2);	98.77
TRUNC (TRUNCATE en *MySQL*; ROUND(x,y,1) en *SQL Server*)	Corta el número en un número específico de decimales, por defecto 0 decimales	SELECT TRUNC(98,7654,2); SELECT TRUNCATE(98,7654,2); SELECT ROUND(98,7654,2,1);	98.76

NOTA

SQLite sólo admite la función ROUND. Las demás opciones de redondeo deben activarse manualmente. Puedes encontrar más detalles en la página de funciones matemáticas del sitio web de SQLite.

Convertir datos a un tipo de datos numérico

La función `CAST` se utiliza para convertir entre varios tipos de datos , y a menudo se utiliza para datos numéricos.

En el siguiente ejemplo, queremos comparar una columna de cadena con una columna numérica

He aquí una tabla con una columna de cadena:

```
+------+ ---------+
| id|  str_col |
+------+ ---------+
|1    | 1.33     |
|2    | 5.5      |
|3    | 7.8      |
+------+ ---------+
```

Intenta comparar la columna de cadena con el valor numérico:

```
SELECCIONAR *
FROM mi_tabla
WHERE str_col > 3;

-- Resultados de MySQL, Oracle y SQLite
+------+ ---------+
| id|  str_col |
+------+ ---------+
|2    | 5.5      |
|3    | 7.8      |
+------+ ---------+

-- Resultados de PostgreSQL y SQL
Server Error
```

NOTA

En *MySQL*, *Oracle* y *SQLite*, la consulta devuelve los resultados correctos porque la columna de cadena se reconoce como una columna numérica cuando se introduce el operador >.

En *PostgreSQL* y *SQL Server*, debe especificar explícitamente `CAST` la columna de cadena en una columna numérica.

Convierte la columna de cadena en una columna decimal para compararla con un número:

```
SELECCIONAR *
FROM mi_tabla
WHERE CAST(str_col AS DECIMAL) > 3;

 id | str_col
----+---------
  2 | 5.5
  3 | 7.8
```

NOTA

El uso de `CAST` no cambia permanentemente el tipo de datos de la columna; sólo lo hace mientras dura la consulta. Para cambiar permanentemente el tipo de datos de una columna, puede modificar la tabla.

Funciones de cadena

Las funciones de cadena pueden aplicarse a columnas con tipos de datos de cadena . Esta sección cubre las operaciones de cadena comunes en SQL.

Determinar la longitud de una cadena

Utiliza la función `LONGITUD`.

En la cláusula SELECT:

```
SELECT LENGTH(nombre)
FROM mi_tabla;
```

En la cláusula WHERE:

```
SELECCIONAR *
FROM mi_tabla
WHERE LENGTH(nombre) < 10;
```

En *SQL Server*, utilice LEN en lugar de LENGTH.

NOTA

La mayoría de los RDBMS excluyen los espacios finales al calcular la longitud de una cadena, mientras que *Oracle* los incluye.

Ejemplo de cadena: 'Al '

Longitud: 2

Longitud en Oracle: 5

Para excluir los espacios finales en Oracle, utilice la función TRIM:

```
SELECT LENGTH(TRIM(nombre))
FROM mi_tabla;
```

Cambiar el caso de una cadena

Utilice la función ARRIBA o ABAJO.

ARRIBA:

```
SELECCIONAR SUPERIOR(tipo)
FROM mi_tabla;
```

BAJA:

```
SELECCIONAR *
FROM mi_tabla
WHERE LOWER(type) = 'public';
```

Oracle y *PostgreSQL* también disponen de `INITCAP(`*`string`*`)` para poner en mayúsculas la primera letra de cada palabra de una cadena y en minúsculas el resto de letras.

Recortar caracteres no deseados alrededor de una cadena

Utilice la función `TRIM` para eliminar tanto los caracteres iniciales como los finales de una cadena. . La siguiente tabla contiene varios caracteres que nos gustaría eliminar:

```
SELECT * FROM mi_tabla;

+----------------+
| Color          |
+----------------+
| .              |
| ...¡naranja!   |
|    ..amarillo.. |
+----------------+
```

Eliminar los espacios alrededor de una cadena

Por defecto, `TRIM` elimina los espacios tanto del lado izquierdo como del derecho de una cadena:

```
SELECT TRIM(color) AS color_clean
FROM mi_tabla;

+-------------+
| color_clean |
+-------------+
| .           |
| ...¡naranja! |
| ..amarillo.. |
+-------------+
```

Eliminar otros caracteres alrededor de una cadena

Puede especificar otros caracteres para eliminar además de un solo espacio. El siguiente código elimina los signos de exclamación alrededor de una cadena:

```
SELECT TRIM('!' FROM color) AS color_clean
FROM mi_tabla;

+-----------------+
| Color_clean     |
+-----------------+
| Rojo            |
| .orange         |
|    ..amarillo.. |
+-----------------+
```

En *SQLite*, utilice `TRIM(color, '!')` en su lugar.

Eliminar caracteres del lado izquierdo o derecho de una cadena

Existen dos opciones para eliminar caracteres a ambos lados de una cadena.

Opción 1: TRIM`(LEADING .. )` *y* `TRIM(TRAILING .. )`
: En *MySQL*, *Oracle* y *PostgreSQL*, puede eliminar caracteres del lado izquierdo o derecho de una cadena con `TRIM(LEADING ..)` y `TRIM(TRAILING ..)`, respectivamente. El siguiente código elimina los signos de exclamación del principio de una cadena:

```
SELECT TRIM(LEADING '!' FROM color) AS
color_clean FROM mi_tabla;

+ ----------------+
| Color_clean     |
+ ----------------+
| Rojo            |
| ...¡naranja!    |
|    ..amarillo.. |
+ ----------------+
```

Opción 2: `LTRIM` *y* `RTRIM`
: Utilice las palabras clave `LTRIM` y `RTRIM` para eliminar caracteres del lado izquierdo o derecho de una cadena, respectivamente.

  En *Oracle*, *PostgreSQL* y *SQLite*, todos los caracteres no deseados pueden listarse dentro de una única cadena. El código siguiente

elimina puntos, signos de exclamación y espacios del principio de una cadena:

```
SELECT LTRIM(color, '.! ') AS color_clean
FROM mi_tabla;

+-------------+
| color_clean |
+-------------+
| Rojo        |
| ¡Naranja!   |
| amarillo..  |
+-------------+
```

En *MySQL* y *SQL Server*, sólo se pueden eliminar los caracteres de espacio en blanco utilizando `LTRIM(color)` o `RTRIM(color)`.

Concatenar cadenas

Utilice la función `CONCAT` o el operador de concatenación (`||`).

```
-- MySQL, PostgreSQL y SQL Server SELECT
CONCAT(id, '_', name) AS id_name FROM
my_table;

-- Oracle, PostgreSQL y SQLite SELECT
id || '_' || nombre COMO id_nombre
FROM mi_tabla;

+-----------+
| Nombre_id |
+-----------+
| 1_Botas   |
| 2_Pumpkin |
| 3_Tiger   |
+-----------+
```

Buscar texto en una cadena

Existen dos métodos para buscar texto en una cadena.

Enfoque 1: ¿Aparece o no el texto en la cadena?

Utilice el operador `LIKE` para determinar si el texto aparece en una cadena o no. Con la siguiente consulta, sólo se devolverán las filas que contengan el texto `some`:

```
SELECCIONAR *
FROM mi_tabla
WHERE mi_texto LIKE '%alguno%';
```

Encontrará más información en la sección `LIKE` de este capítulo.

Enfoque 2: ¿Dónde aparece el texto en la cadena?

Utilice la función `INSTR`/`POSITION`/`CHARINDEX` para determinar la ubicación del texto en una cadena.

La Tabla 7-13 enumera los parámetros requeridos por las funciones de localización en cada RDBMS.

Tabla 7-13. Funciones para encontrar la posición de un texto en una cadena

RDBMS	Formato de código
MySQL	`INSTR(`*`cadena`*`, `*`subcadena`*`)` `LOCATE(`*`subcadena`*`, `*`cadena`*`, `*`posición`*`)`
Oracle	`INSTR(`*`cadena`*`, `*`subcadena`*`, `*`posición`*`,` *`ocurrencia`*`)`
PostgreSQL	`POSITION(`*`subcadena`*` EN `*`cadena`*`)` `STRPOS(`*`cadena`*`, `*`subcadena`*`)`
SQL Server	`CHARINDEX(`*`subcadena`*`, `*`cadena`*`, `*`posición`*`)`
SQLite	`INSTR(`*`cadena`*`, `*`subcadena`*`)`

Las entradas son:

- *cadena* (*obligatorio*): la cadena en la que se busca (es decir, el nombre de una columna `VARCHAR`)
- *subcadena* (*obligatorio*): la cadena que se busca (es decir, un carácter, una palabra, etc.)

- *posición* (*opcional*): la posición inicial de la búsqueda. Por defecto, comienza en el primer carácter (1). Si la *posición* es negativa, la búsqueda comienza al final de la cadena.
- *ocurrencia* (*opcional*): la primera/segunda/tercera, etc. vez que la subcadena aparece en la cadena. Por defecto es la primera aparición (1).

He aquí una tabla de muestra:

```
+------------------------------+
| Mi_texto                     |
+------------------------------+
| Aquí hay algo de texto.      |
| Y algunos números - 1 2 3 4 5 |
| Y algunos signos de puntuación. :)    |
+------------------------------+
```

Buscar la posición de la subcadena `some` dentro de la cadena `mi_texto`:

```
SELECT INSTR(my_text, 'some') AS some_location
FROM my_table;

+---------------+
| alguna_ubicacion |
+---------------+
|9 |
|5 |
|5 |
+---------------+
```

El recuento en SQL empieza por 1

A diferencia de otros lenguajes de programación que tienen índice cero (la cuenta empieza en 0), en SQL la cuenta empieza en 1.

El 9 en la salida anterior significa el noveno carácter.

NOTA

En *Oracle*, las expresiones regulares también se pueden utilizar para buscar una subcadena utilizando `REGEXP_INSTR`. Más detalles en la sección de expresiones regulares en Oracle.

Extraer una parte de una cadena

Utilice la función `SUBSTR` o `SUBSTRING`. El nombre de la función y las entradas difieren para cada RDBMS:

```
-- MySQL, Oracle, PostgreSQL y SQLite
SUBSTR(cadena, inicio, longitud)

-- MySQL
SUBSTR(cadena FROM inicio POR longitud)

-- MySQL, PostgreSQL y SQL Server
SUBSTRING(cadena, inicio, longitud)

-- MySQL y PostgreSQL
SUBSTRING(cadena FROM inicio PARA
longitud)
```

Las entradas son:

- *cadena* (*obligatorio*): la cadena en la que se busca (es decir, el nombre de una columna `VARCHAR`)
- *start* (*obligatorio*): el lugar de inicio de la búsqueda. Si *start* es `1`, la búsqueda comenzará desde el primer carácter, `2` es el segundo carácter, y así sucesivamente. Si *start* es `0`, se tratará como un 1. Si *start* es negativo, la búsqueda comenzará desde el último carácter.
- *longitud* (*opcional*): la longitud de la cadena devuelta. Si se omite length, se devolverán todos los caracteres desde el *principio* hasta el final de la cadena. En *SQL Server*, la *longitud* es obligatoria. He aquí una tabla de ejemplo:

```
+------------------------------+
| Mi_texto                     |
+------------------------------+
| Aquí hay algo de texto.      |
| Y algunos números - 1 2 3 4 5 |
| Y algunos signos de puntuación. :)    |
+------------------------------+
```

Extraer una subcadena:

```
SELECT SUBSTR(mi_texto, 14, 8) AS
sub_str FROM mi_tabla;

+----------+
| sub_str |
+----------+
| texto.   |
| ers - 1 |
| ¡Una situación! |
+----------+
```

NOTA

En *Oracle*, las expresiones regulares también se pueden utilizar para extraer una subcadena de utilizando `REGEXP_SUBSTR`. Más detalles en la sección de expresiones regulares en Oracle.

Reemplazar texto en una cadena

Utilice la función `REEMPLAZAR`. Tenga en cuenta el orden de las entradas de la función:

```
REPLACE(cadena, cadena_antigua, cadena_nueva)
```

He aquí una tabla de muestra:

```
+------------------------------+
| Mi_texto                     |
+------------------------------+
| Aquí hay algo de texto.      |
```

```
| Y algunos números - 1 2 3 4 5 |
| Y algunos signos de puntuación. :)    |
+-----------------------------+
```

Sustituye la palabra `algunos` por la palabra `los`:

```
SELECT REPLACE(mi_texto, 'algunos', 'los')
       AS new_text
FROM mi_tabla;

+-----------------------------+
| nuevo_texto                 |
+-----------------------------+
| Este es el texto.           |
| Y los números - 1 2 3 4 5 |
| ¡Y la puntuación! :)        |
+-----------------------------+
```

NOTA

En *Oracle* y *PostgreSQL, las expresiones regulares* también pueden utilizarse para reemplazar una cadena utilizando `REGEXP_REPLACE`. Encontrará más detalles en las secciones Expresiones regulares en Oracle y Expresiones regulares en PostgreSQL.

Borrar texto de una cadena

Puede utilizar la función `REPLACE`, pero especificando una cadena vacía como valor de sustitución.

Sustituye la palabra `some` por una cadena vacía:

```
SELECT REPLACE(mi_texto, 'algunos ', '')
       AS new_text
FROM mi_tabla;

+-------------------------+
| nuevo_texto             |
+-------------------------+
| Aquí está el texto.     |
```

```
| Y números - 1 2 3 4 5 |
| ¡Y puntuación! :)       |
+-------------------------+
```

Utilizar expresiones regulares

Las expresiones regulares permiten encontrar patrones complejos. Por ejemplo, encontrar todas las palabras que tengan exactamente cinco letras o encontrar todas las palabras que empiecen por mayúscula.

Imagina que tienes la siguiente receta de condimento para tacos:

```
- 1 cucharada de chile en polvo
- .5 cucharada de comino molido
- .5 cucharadita de pimentón
- .25 cucharadita de ajo en polvo
- .25 cucharadita de cebolla en polvo
- 0,25 cucharadita de hojuelas de pimiento rojo
  triturado
- .25 cucharadita de orégano seco
```

Quiere excluir las cantidades y tener sólo una lista de ingredientes. Para ello, puede escribir una expresión regular que extraiga todo el texto que sigue al término `cuchara`.

La expresión regular tendría el siguiente aspecto:

```
(?<=cuchara ).*$
```

y los resultados serían los siguientes:

```
chile en
polvo comino
molido
pimentón ajo
en polvo
cebolla en
polvo
copos de pimienta roja
triturada orégano seco
```

La expresión regular recorrió todo el texto y extrajo cualquier texto que se encontrara entre el término `cuchara` y el final de la línea.

Hay que tener en cuenta un par de cosas sobre las expresiones regulares:

- La sintaxis de las expresiones regulares no es intuitiva. Es útil desglosar el significado de cada parte de una expresión regular utilizando una herramienta en línea, como Regex101.
- Las expresiones regulares no son específicas de SQL. Pueden utilizarse en muchos lenguajes de programación y editores de texto.
- RegexOne ofrece un rápido tutorial introductorio. También puede consultar el post de Thomas Nield en O'Reilly, "An Introduc- tion to Regular Expressions".

CONSEJO

En lugar de memorizar la sintaxis de las expresiones regulares, recomiendo encontrar expresiones regulares existentes y `modificarlas` para adaptarlas a sus necesidades.

Para la expresión regular anterior, busqué "expresión regular texto después de cadena".

El segundo resultado de la búsqueda en Google me llevó a `(?<=WORD).*$`. Utilicé Regex101 para entender cada parte de la expresión regular, y finalmente sustituí `WORD` por `cuchara`.

Las funciones de las expresiones regulares varían mucho según el RDBMS, por lo que hay una sección separada para cada uno. SQLite no soporta expresiones regulares por defecto, pero pueden implementarse. Puede encontrar más detalles en la documentación de SQLite.

Expresiones regulares en MySQL

Utilice `REGEXP` para buscar un patrón de expresión regular en cualquier lugar de una cadena.

He aquí una tabla de muestra:

```
+---------------------------+ ----------------+
| Ciudad                    |
+---------------------------+ ----------------+
```

```
| 10 cosas que odio de ti | Seattle        |
| 22 Jump Street |           Nueva Orleans |
| The Blues Brothers Chicago |
| "Ferris Bueller's Day Off  |
+--------------------------+ ----------------+
```

Encuentra todas las variantes ortográficas de Chicago:

```
SELECCIONAR *
DE películas
WHERE ciudad REGEXP '(Chicago|CHI|Chitown)';

+-------------------------+------------+
| Ciudad                  |
+-------------------------+------------+
| The Blues Brothers       Chicago
| Ferris Bueller's Day Off | Chi     |
+-------------------------+------------+
```

Las expresiones regulares de MySQL no distinguen entre mayúsculas y minúsculas para las cadenas de caracteres ; `CHI` y `Chi` se consideran equivalentes.

Encuentra todas las películas con números en el título:

```
SELECCIONAR *
DE películas
WHERE title REGEXP '\\d';

+--------------------------+ ----------------+
| Ciudad                   |
+--------------------------+ ----------------+
| 10 cosas que odio de ti | Seattle        |
| 22 Jump Street |           Nueva Orleans |
+--------------------------+ ----------------+
```

En MySQL, cualquier barra invertida simple en una expresión regular (`\d` = cualquier dígito) debe cambiarse por una barra invertida doble.

Expresiones regulares en Oracle

Oracle admite muchas funciones de expresiones regulares, entre ellas

- `REGEXP_LIKE` coincide con un patrón de expresión regular dentro del texto.
- `REGEXP_COUNT` cuenta el número de veces que aparece un patrón en el texto.
- `REGEXP_INSTR` localiza las posiciones en las que aparece un patrón en el texto.
- `REGEXP_SUBSTR` devuelve las subcadenas del texto que coinciden con un patrón.
- `REGEXP_REPLACE` sustituye las subcadenas que coinciden con un patrón por otro texto.

He aquí una tabla de muestra:

```
 TÍTULO                         CIUDAD
---------------------------- -------------
 10 cosas que odio de ti      Seattle
 22 Jump                      StreetNueva Orleans
 The Blues
                              Brother
 sChicago Ferris Bueller's Day
                              OffChi
```

Encuentra todas las películas con números en el título:

```
SELECCIONAR *
DE películas
WHERE REGEXP_LIKE(title, '\d');

 TÍTULO                         CIUDAD
---------------------------- -------------
 10 cosas que odio de ti      Seattle
 22 Jump                      StreetNueva Orleans
```

NOTA

Las siguientes expresiones son equivalentes:

```
REGEXP_LIKE(título, \d)
REGEXP_LIKE(título, [0-9])
REGEXP_LIKE(título, [[:dígito:]])
```

La tercera opción utiliza la sintaxis de expresiones regulares POSIX, , que es compatible con Oracle.

Cuente el número de mayúsculas del título:

```
SELECT título, REGEXP_COUNT(título, '[A-Z]')
       AS num_caps
FROM movies;

 TÍTULO                        NUM_CAPS
---------------------------- ----------
 10 cosas que odio de ti              5
 22 Jump Street                       2
 Los Blues Brothers                   3
 Ferris Bueller's Day Off             4
```

Encuentra la ubicación de la primera vocal en el título:

```
SELECT título, REGEXP_INSTR(título, '[aeiou]')
       AS first_vowel
FROM películas;

 TÍTULO                        PRIMERA VOCAL
---------------------------- -------------
 10 cosas que odio de ti                 6
 22 Jump Street                          5
 Los Blues Brothers                      3
 Ferris Bueller's Day Off                2
```

Devuelve todos los números del título:

```
SELECT título, REGEXP_SUBSTR(título, '[0-9]+')
       AS nums
FROM películas
```

```
WHERE REGEXP_SUBSTR(title, '[0-9]+') IS NOT NULL;

 TÍTULO                       NÚMEROS
---------------------------- ------
 10 cosas que odio de ti        10
 22 Jump Street                 22
```

Sustituye todos los números del título por el número `100`:

```
SELECT REGEXP_REPLACE(título, '[0-9]+', '100')
       AS one_hundred_title
FROM películas;

 ONE_HUNDRED_TITLE
----------------------------
 100 cosas que odio de ti
 100 Jump Street
```

NOTA

Puede encontrar más detalles y ejemplos sobre expresiones regulares en Ora- cle en *Oracle Regular Expressions Pocket Reference* de Jonathan Gennick y Peter Linsley (O'Reilly).

Expresiones regulares en PostgreSQL

Utilice `SIMILAR A` o `~` para buscar un patrón de expresión regular en cualquier lugar de una cadena.

He aquí una tabla de muestra:

```
           título            |    ciudad
-----------------------------+-------------
 10 cosas que odio de ti | Seattle
 22 Jump Street|             Nueva Orleans
 The Blues Brothers          Chicago
 Ferris Bueller's Day Off    Chi
```

Encuentra todas las variantes ortográficas de Chicago:

```
SELECCIONAR *
DE películas
WHERE ciudad SIMILAR A '(Chicago|CHI|Chi|Chitown)';

                          título| ciudad
-------------------------+---------
 The Blues Brothers       Chicago
 Ferris Bueller's Day Off | Chi
```

Las expresiones regulares de PostgreSQL distinguen entre mayúsculas y minúsculas para las cadenas de caracteres ter; `CHI` y `Chi` se consideran valores diferentes.

SIMILAR A Versus ~

`SIMILAR A` ofrece capacidades limitadas de expresión regular, y se utiliza más a menudo para ofrecer múltiples alternativas (`Chicago|CHI| Chi`). Otros símbolos regex comunes para utilizar con SIMILAR A son
* (0 o más), + (1 o más) y {} (número exacto de veces).

La tilde (~) debe usarse para expresiones regulares más avanzadas junto con la sintaxis POSIX, que es otro tipo de expresión regular que soporta PostgreSQL.

La lista completa de símbolos compatibles se encuentra en la documentación de Post- greSQL.

En el siguiente ejemplo se utiliza ~ en lugar de

`SIMILAR A`. Encuentra todas las películas con

números en el título:

```
SELECCIONAR *
DE películas
WHERE title ~ '\d';

+--------------------------+ ----------------+
| Ciudad                     |
+--------------------------+ ----------------+
| 10 cosas que odio de ti | Seattle        |
| 22 Jump Street |           Nueva Orleans |
```

```
+---------------------------+ ----------------+
```

PostgreSQL también admite `REGEXP_REPLACE`, que permite reemplazar en los caracteres de una cadena que coincidan con un patrón determinado.

Sustituye todos los números del título por el número `100`:

```
SELECT REGEXP_REPLACE(title, '\d+', '100')
DE películas;

regexp_replace
----------------------------
100 cosas que odio de ti
100 Jump Street
The Blues Brothers
Ferris Bueller's Day Off
```

La expresión regular `\d` es equivalente a `[0-9]` y `[[:dígito::]]`.

Expresiones regulares en SQL Server

SQL Server admite una cantidad muy limitada de expresio- nes regulares a través de su palabra clave `LIKE`.

He aquí una tabla de muestra:

```
 título                      ciudad
--------------------------- -------------
 10 cosas que odio de ti     Seattle
 22 Jump                     StreetNueva Orleans
 The Blues
                             Brother
 sChicago Ferris Bueller's Day
                             OffChi
```

SQL Server utiliza una sintaxis de expresión regular ligeramente diferente, que se detalla en la documentación de Microsoft.

Encuentra todas las películas con números en el título:

```
SELECCIONAR *
DE películas
WHERE title LIKE '%[0-9]%';
```

```
 título                        ciudad
--------------------------- -------------
 10 cosas que odio de ti       Seattle
 22 Jump                       StreetNueva Orleans
```

Convertir datos a un tipo de cadena

Cuando las funciones de cadena se aplican a tipos de datos que no son cadenas, normalmente no se produce ningún problema aunque haya una discordancia de tipo de datos .

La siguiente tabla tiene una columna numérica llamada `números`:

```
+---------+
| números |
+---------+
| 1.33    |
| 2.5     |
| 3.777   |
+---------+
```

Cuando la función de cadena `LENGTH` (o `LEN` en *SQL Server*) se aplica a la columna numérica, la sentencia se ejecuta sin errores en la mayoría de los RDBMS:

```
SELECT LENGTH(numbers) AS len_num
FROM mi_tabla;

-- Resultados de MySQL, Oracle, SQL Server y SQLite
+---------+
| len_num |
+---------+
|4 |
|3 |
|5 |
+---------+

-- Resultados
PostgreSQL Error
```

En *PostgreSQL*, debe `convertir` explícitamente la columna numérica en una columna de cadena:

```
SELECT LENGTH(CAST(números COMO CHAR(5))) AS
len_num FROM mi_tabla;

 len_num
 ---------
       4
       3
       5
```

NOTA

El uso de CAST no cambia permanentemente el tipo de datos de la columna, sólo mientras dure la consulta. Para cambiar permanentemente el tipo de datos de una columna, puede modificar la tabla.

Funciones de fecha y hora

Las funciones datetime pueden aplicarse a columnas con tipos de datos datetime. Esta sección cubre las funciones datetime más comunes en SQL.

Devolver la fecha o la hora actual

Las siguientes sentencias devuelven la fecha actual, la hora actual y la fecha y hora actuales:

```
-- MySQL, PostgreSQL y SQLite SELECT
CURRENT_DATE;
SELECT HORA_ACTUAL;
SELECT HORA_ACTUAL;

-- Oracle
SELECT FECHA_ACtual FROM dual;
SELECT CAST(CURRENT_TIMESTAMP AS TIME) FROM dual;
SELECT CURRENT_TIMESTAMP FROM dual;

-- SQL Server
```

```
SELECT CAST(ACTUAL_TIMESTAMP AS DATE);
SELECT CAST(ACTUAL_TIMESTAMP AS TIME);
SELECT ACTUAL_TIMESTAMP;
```

Existen muchas otras funciones equivalentes a éstas, entre ellas `CURDATE()` en *MySQL*, `GETDATE()` en *SQL Server*, etc.

Las tres situaciones siguientes muestran cómo s e utilizan estas funciones en la práctica.

Muestra la hora actual:

```
SELECT HORA_ACTUAL;

+--------------+
| current_time |
+--------------+
| 20:53:35     |
+--------------+
```

Crear una tabla que marque la fecha y hora de creación:

```
CREATE TABLE mi_tabla
       (id INT,
       creation_datetime TIMESTAMP DEFAULT
                         HORA_ACTUAL);

INSERT INTO mi_tabla (id)
       VALUES (1), (2), (3);

 +------+ ---------------------+
 | id| creation_datetime       |
 +------+ ---------------------+
 |1    | 2021-02-15 20:57:12 |
 |2    | 2021-02-15 20:57:12 |
 |3    | 2021-02-15 20:57:12 |
 +------+ ---------------------+
```

Buscar todas las filas de datos anteriores a una fecha determinada:

```
SELECCIONAR *
DESDE  mi_tabla
WHERE creation_datetime < CURRENT_DATE;
```

```
+------+ --------------------+
| id| creation_datetime       |
+------+ --------------------+
|1    | 2021-01-15 10:47:02 |
|2 |  2021-01-15 10:47:02 |
|3    | 2021-01-15 10:47:02 |
+------+ --------------------+
```

Sumar o restar un intervalo de fecha u hora

Puede sumar o restar diversos intervalos de tiempo (años, meses, días, horas, minutos, segundos, etc.) a los valores de fecha y hora.

La Tabla 7-14 enumera las formas de restar un día.

Tabla 7-14. Devuelve la fecha de ayer

RDBMSCode	MySQL

	`SELECT FECHA_ACTUAL - INTERVALO 1 DIA;`
	`SELECT SUBFECHA(FECHA_ACTUAL, 1);`
	`SELECT FECHA_SUB(FECHA_ACTUAL,`
	`INTERVALO 1 DÍA);`

Oracle	`SELECT FECHA_ACTUAL - INTERVALO '1' DIA` `DE doble;`
PostgreSQL	`SELECT CAST(CURRENT_DATE -` `INTERVALO '1 día' AS DATE);`

SQL Server	`SELECT CAST(CURRENT_TIMESTAMP - 1 AS DATE);`
	`SELECT DATEADD(DAY, -1, CAST(`
	`CURRENT_TIMESTAMP AS DATE));`

SQLite	`SELECT FECHA(FECHA_ACtual, '-1 día');`

La Tabla 7-15 enumera las formas de añadir tres horas.

Tabla 7-15. Devuelve la fecha y la hora dentro de tres horas

RDBMSCode	MySQL

	`SELECT ACTUAL_TIMESTAMP + INTERVALO 3 HORA;`
	`SELECT ADDDATE(ACTUAL_TIMESTAMP,`
	`INTERVALO 3 HORA);`
	`SELECT DATE_ADD(ACTUAL_TIMESTAMP,`
	`INTERVALO 3 HORA);`

Oracle	`SELECT FECHA_ACTUAL + INTERVALO '3' HORA DE doble;`
PostgreSQL	`SELECT CURRENT_TIMESTAMP +` `INTERVALO '3 horas';`
SQL Server	`SELECT DATEADD(HOUR, 3, CURRENT_TIMESTAMP);`
SQLite	`SELECT DATETIME(FECHA-HORA_ACTUAL,` `+3 horas");`

Encontrar la diferencia entre dos fechas u horas

Puede hallar la diferencia entre dos fechas, horas o fechas-hora en términos de diversos intervalos de tiempo (años, meses, días, horas, minutos, segundos, etc.).

Encontrar una diferencia de fecha

Dadas una fecha de inicio y una fecha final, la Tabla 7-16 enumera las formas de hallar los días entre ambas fechas.

He aquí una tabla de muestra:

```
+------------+-------------+
| Fecha_inicio Fecha_final|
+------------+-------------+
| 2016-10-10 | 2020-11-11 |
| 2019-03-03 | 2021-04-04 |
+------------+-------------+
```

Tabla 7-16. Días entre dos fechas

RDBMSCode MySQL

```
SELECT DATEDIFF(fecha_fin, fecha_inicio)
       AS
day_diff FROM
       mi_tabla;
```

Oracle

```
SELECT (fecha_fin - fecha_inicio) AS
day_diff FROM     mi_tabla;
```

PostgreSQL

```
SELECT AGE(fecha_fin, fecha_inicio) AS
day_diff FROM     mi_tabla;
```

SQL Server

```
SELECT DATEDIFF(día, fecha_inicio, fecha_fin)
       AS
day_diff FROM
       mi_tabla;
```

SQLite

```
SELECT (julianday(fecha_fin) -
       julianday(fecha_inicio)) AS
day_diff FROM     mi_tabla;
```

Tras ejecutar el código de la tabla, estos son los resultados:

```
-- MySQL, Oracle, SQL Server y SQLite
+----------+
| day_diff |
+----------+
      |1493 |
       |763 |
+----------+

-- PostgreSQL

   diferencia_diaria
 --------------------
 4 años 1 mes 1 día
 2 años 1 mes 1 día
```

Encontrar una diferencia horaria

Dada una hora inicial y final, la Tabla 7-17 enumera las formas de hallar los segundos entre ambas horas.

He aquí una tabla de muestra:

```
+------------+ ------------+
| hora_inicio | hora_final |
+------------+ ------------+
| 10:30:00|  11:30:00 |
| 14:50:32|  15:22:45 |
+------------+ ------------+
```

Tabla 7-17. Segundos entre dos tiempos

RDBMSCode	MySQL

	`SELECT TIMEDIFF(hora_fin, hora_inicio)`
	`AS time_diff`
	`FROM mi_tabla;`

Oracle	No tipo de datos de tiempo

PostgreSQL	`SELECT EXTRACT(epoch from end_time -`
	`start_time) AS`
	`time_diff FROM mi_tabla;`

SQL Server	`SELECT DATEDIFF(second, start_time, end_time)`
	`AS time_diff`
	`FROM mi_tabla;`

SQLite	`SELECT (strftime('%s',hora_fin) -`
	`strftime('%s',hora_inicio))`
	`COMO diferencia_tiempo`
	`DESDE mi_tabla;`

Tras ejecutar el código de la tabla, estos son los resultados:

```
-- MySQL
+-----------+
| time_diff |
+-----------+
| 01:00:00 |
| 00:32:13 |
+-----------+
```

```
-- PostgreSQL, SQL Server y SQLite

 diferencia_tiempo
-----------
 3600
 1933
```

Encontrar una diferencia de fecha y hora

Dadas una fecha de inicio y una fecha de fin, la Tabla 7-18 enumera las formas de encontrar el número de horas entre las dos fechas.

He aquí una tabla de muestra:

```
+---------------------+ ----------------------+
|                      start_dt| end_dt       |
+---------------------+ ----------------------+
| 2016-10-10 10:30:00 | 2020-11-11 11:30:00 |
| 2019-03-03 14:50:32 | 2021-04-04 15:22:45 |
+---------------------+ ----------------------+
```

Tabla 7-18. Horas entre dos fechas

RDBMSCode MySQL

```
SELECT TIMESTAMPDIFF(hora, start_dt, end_dt)
       AS
diferencia_horas
FROM   mi_tabla;
```

Oracle

```
SELECT (fin_dt - inicio_dt) AS
hour_diff FROM    mi_tabla;
```

PostgreSQL

```
SELECT AGE(end_dt, start_dt) AS hour_diff
FROM   mi_tabla;
```

SQL Server

```
SELECT DATEDIFF(hora, start_dt, end_dt)
       AS
diferencia_horas
FROM   mi_tabla;
```

SQLite

```
SELECT ((julianday(end_dt) -
       julianday(start_dt))*24) AS hour_diff
FROM   mi_tabla;
```

Tras ejecutar el código de la tabla, estos son los resultados:

```
-- MySQL, SQL Server y SQLite
+-----------+
| hour_diff |
+-----------+
      |35833 |
      |18312 |
+-----------+

-- Oracle

 HOUR_DIFF
--------------------------
 +000001493 01:00:00.000000
 +000000763 00:32:13.000000

-- PostgreSQL

        diferencia_horas
  ----------------------------
 4 años 1 mes 1 día 01:00:00
 2 años 1 mes 1 día 00:32:13
```

NOTA

El resultado de *PostgreSQL* es largo:

```
SELECT AGE(fecha_fin,
fecha_inicio) FROM mi_tabla;

              edad
 ----------------------------
 4 años 1 mes 1 día 01:00:00
 2 años 1 mes 1 día 00:32:13
```

Utilice la función `EXTRACT` para extraer sólo el campo `del año`.

```
SELECT EXTRACT(año FROM
                AGE(fecha_final, fecha_inicial))
FROM mi_tabla;

fecha_parte
 -----------
          4
          2
```

Extraer una parte de una fecha u hora

Existen múltiples formas de extraer una unidad de tiempo (mes, hora, , etc.) de un valor de fecha u hora. La Tabla 7-19 muestra cómo hacerlo, específicamente para la unidad de tiempo mes.

Tabla 7-19. Extraer el mes de una fecha

RDBMSCode	MySQL
	`SELECT EXTRACT(mes FROM FECHA_ACtual);` `SELECT MES(FECHA_ACTUAL);`
Oracle	`SELECT EXTRACT(mes FROM FECHA_ACtual)` `DE doble;`
PostgreSQL	`SELECT EXTRACT(month FROM CURRENT_DATE);` `SELECT DATE_PART('month', CURRENT_DATE);`

RDBMS	Código
SQL Server	`SELECT DATEPART(month, CURRENT_TIMESTAMP);` `SELECT MONTH(CURRENT_TIMESTAMP);`
SQLite	`SELECT strftime('%m', CURRENT_DATE);`

Tanto *MySQL* como *SQL Server* soportan funciones específicas de unidad de tiempo como `MONTH()`, como se ve en la Tabla 7-19.

- *MySQL* admite `YEAR()`, `QUARTER()`, `MONTH()`, `WEEK()`, `DAY()`, `HOUR()`, `MINUTE()` y `SECOND()`.
- *SQL Server* admite `YEAR()`, `MONTH()` y `DAY()`.

Puede sustituir los valores `month` o `%m` de la Tabla 7-19 por otras unidades de tiempo. La Tabla 7-20 enumera las unidades de tiempo aceptadas por cada RDBMS.

Tabla 7-20. Opciones de unidad de tiempo

MySQL	Oracle	PostgreSQL	SQL Server	SQLite
microsegundo	segundo	microsegundo	nanosegundo	%f (fracción de segundo)
segundo	minuto	milisegundo	microsegundo	%S (segundo)
minuto	hora	segundo	milisegundo	%s (segundos desde 1970-01-01)
hora	día	minuto	segundo	%M (minuto)
día	mes	hora	minuto	%H (hora)
semana	año	día día	hora	%J (número del día juliano)
mes		semana	semana	%w (día de la semana)
trimestre		mes	día día	%d (día del mes)
año		trimestre	del año	%j (día del año)
		década	mes	%W (semana del año)
		siglo	trimestre	%m (mes)
			año	%Y (año)

NOTA

También puede extraer una unidad de tiempo de un valor de cadena. El código se encuentra en la Tabla 7-28: Extraer año de una cadena.

Determinar el día de la semana de una fecha

Dada una fecha, determinar el día de la semana:

- Fecha: `2020-03-16`
- Día numérico de la semana: `2` (el domingo es el primer día)
- Día de la semana `Lunes`

La tabla 7-21 devuelve el día numérico de la semana de una fecha dada. El domingo es el primer día, el lunes el segundo, y así sucesivamente.

Tabla 7-21. Devuelve el día numérico de la semana

RDBMS	CódigoRango	de valores
MySQL	`SELECT DAYOFWEEK('2020-03-16');`	1 a 7
Oracle	`SELECT TO_CHAR( date '2020-03-16', 'd') DE doble;`	1 a 7 0 a 6
PostgreSQL	`SELECT DATE_PART('dow', fecha '2020-03-16');`	1 a 7
SQL Server	`SELECT DATEPART(día de la semana, '2020-03-16');`	0 a 6
SQLite	`SELECT strftime('%w', '2020-03-16');`	

La Tabla 7-22 devuelve el día de la semana de una fecha dada.

Tabla 7-22. Devolver el día de la semana

RDBMSCode	MySQL
	`SELECT DAYNAME('2020-03-16');`
Oracle	`SELECT TO_CHAR(fecha '2020-03-16', 'día') FROM dual;`
PostgreSQL	`SELECT TO_CHAR(fecha '2020-03-16', 'día');`
SQL Server	`SELECT DATENAME(día de la semana, '2020-03-16');`
	SQLiteNo disponible

Redondear una fecha a la unidad de tiempo más próxima

Oracle y *PostgreSQL* admiten el redondeo y el truncamiento (también conocido como redondeo a la baja).

Redondeo en Oracle

Oracle permite redondear y truncar una fecha al año, mes o día más próximo (primer día de la semana).

Para redondear a primeros de mes:

```
SELECT TRUNC(fecha '2020-02-25', 'mes')
DE     doble;

01-FEB-20
```

Para redondear al mes más próximo:

```
SELECT ROUND(fecha '2020-02-25', 'mes')
DE     doble;

01-MAR-20
```

Redondeo en PostgreSQL

PostgreSQL permite truncar una fecha al año, trimestre, mes, semana (primer día de la semana), día, hora, minuto o segundo más cercano. Puede encontrar unidades de tiempo adicionales en la documentación de PostgreSQL.

Para redondear a primeros de mes:

```
SELECT DATE_TRUNC('mes', FECHA '2020-02-25');

2020-02-01 00:00:00-06
```

Para redondear al minuto:

```
SELECT DATE_TRUNC('minuto', TIEMPO '10:30:59.12345');

10:30:00
```

Convertir una cadena en un tipo de datos Datetime

Existen dos formas de convertir una cadena en un tipo de dato datetime:

- Utilice la función `CAST` para un caso sencillo.
- Utilice `STR_TO_DATE`/`TO_DATE`/`CONVERT` para un caso personalizado.

La función CAST

Si una columna de cadena contiene fechas en un formato estándar, puede utilizar la función `CAST` para convertirla en un tipo de datos de fecha.

La Tabla 7-23 muestra el código para convertir a un tipo de datos de fecha.

Tabla 7-23. Convertir una cadena en una fecha

RDBMSCódigo de	formato de fecha requerido
MySQL, PostgreSQL, SQL Server	YYYY-MM-DDSELECT CAST('2020-10-15' COMO FECHA);
Oracle	DD-MON-AAAA CAST('15-OCT-2020') COMO FECHA) DE doble;
SQLite	YYYY-MM-DDSELECT DATE('2020-10-15');

La Tabla 7-24 muestra el código para convertir a un tipo de datos de tiempo.

Tabla 7-24. Convertir una cadena en una hora

RDBMS	Código de formato de hora requerido	
MySQL, PostgreSQL, SQL Server	`hh:mm:ss`	`SELECT CAST('14:30' AS TIME);`
Oracle	`hh:mm:ss` `hh:mm:ss AM/PM`	`SELECT CAST('02:30:00 PM' AS TIME) DE doble;`
SQLite	`hh:mm:ss`	`SELECT TIME('14:30');`

La Tabla 7-25 muestra el código para convertir a un tipo de datos datetime.

Tabla 7-25. Convertir una cadena en una fecha y hora

RDBMS	Formato de fecha y hora requerido	Código
MySQL, Servidor SQL	`AAAA-MM-DD hh:mm:ss`	`SELECT CAST('2020-10-15 14:30' AS DATETIME);`
Oracle	`DD-MON-AAAA hh:mm:ss` `DD-MON-AAAA hh:mm:ss AM/PM`	`SELECT CAST('15-OCT-20 02:30:00 PM' AS TIMESTAMP) DE doble;`
PostgreSQL	`AAAA-MM-DD hh:mm:ss`	`SELECT CAST('2020-10-15 14:30' AS TIMESTAMP);`
SQLite	`AAAA-MM-DD hh:mm:ss`	`SELECT DATETIME('2020-10-15 14:30');`

La función `CAST` también puede utilizarse para convertir fechas a tipos de datos numéricos y de cadena.

Funciones STR_TO_DATE, TO_DATE y CONVERT

Para fechas y horas que no estén en los formatos estándar AAAA-MM-DD/DD-MON- AAAA/hh:mm:ss, utilice en su lugar una función de cadena a fecha o de cadena a hora .

La Tabla 7-26 lista las funciones string to date y string to time para cada RDBMS. Las cadenas de ejemplo en el código están en formatos no estándar `MM-DD-AA` y `hhmm`.

Tabla 7-26. Funciones de cadena a fecha y cadena a hora

RDBMS	String a fecha	String a hora
MySQL	`SELECT STR_TO_DATE('10-15-22', '%m-%d-%y');`	`SELECT STR_TO_DATE('1030', '%H%i');`
Oracle	`SELECCIONE TO_DATE('10-15-22', MM-DD-AA') DE doble;`	`SELECT TO_TIMESTAMP('1030', 'HH24MI') DE doble;`
PostgreSQL	`SELECT TO_DATE('10-15-22', MM-DD-AA');`	`SELECT TO_TIMESTAMP('1030', 'HH24MI');`
Servidor SQL	`SELECT CONVERT( VARCHAR, '10-15-22', 105);`	`SELECT CAST( CONCAT(10,':',30) AS TIME);`
SQLite	Sin función de fecha no estándar	Sin función de hora no estándar

NOTA

SQL Server utiliza la función `CONVERT` para cambiar una cadena a un tipo de dato datetime. `VARCHAR` es el tipo de datos original, `10-15-22` es la fecha, y `105` representa el formato `MM-DD- YYYY`.

Otros formatos de fecha son `MM/DD/AAAA` (101), `AAAA.MM.DD` (102), `DD/MM/AAAA` (103) y `DD.MM.AAAA` (104). Más para-en la documentación de Microsoft.

Los formatos de hora son `hh:mi:ss` (108) y `hh:mi:ss:mmm` (114), ninguno de los cuales coincide con el formato de la Tabla 7-26, razón por la cual la hora no puede ser leída por SQL Server utilizando `CON VERT`.

Puede sustituir los valores `%H%i` o `HH24MI` de la Tabla 7-26 por otras unidades de tiempo. La Tabla 7-27 enumera los especificadores de formato comunes para *MySQL*, *Oracle* y *PostgreSQL*.

Tabla 7-27. Especificadores de formato de fecha y hora

Descripción de MySQL Oracle y PostgreSQL		
%Y	AAAA	Año de 4 cifras
%y	YY	Año de 2 cifras
%m	MM	Mes numérico (1-12)
%b	LUN	Mes abreviado (enero-diciembre)
%M	MES	Nombre del mes (enero-diciembre)
%d	DD	Día (1-31)
%h	HH o HH12	12 horas (1-12)
%H	HH24	24 horas (0-23)
%i	MI	Minutos (0-59)
%s	SS	Segundos (0-59)

Aplicar una función de fecha a una columna de cadena

Imagina que tienes la siguiente columna de cadenas:

```
str_column
15/10/2022
10/16/2023
10/17/2024
```

Desea extraer el año de cada fecha:

```
año_columna
2022
2023
2024
```

Problema

No puede utilizar una función de fecha y hora (`EXTRACT`) en una columna de cadena (`str_column`).

Solución

Primero convierta la columna de cadena en una columna de fecha. A continuación, aplique la función datetime. La Tabla 7-28 enumera cómo hacerlo en cada RDBMS.

Tabla 7-28. Extraer año de una cadena

RDBMSCode	MySQL
	`SELECT AÑO(STR_AL_FECHA(str_columna, '%m/%d/%Y')) FROM mi_tabla;`
Oracle	`SELECT EXTRACT(YEAR FROM TO_DATE(str_column, 'MM/DD/AAAA')) DESDE mi_tabla;`
PostgreSQL	`SELECT EXTRACT(YEAR FROM TO_DATE(str_column, 'MM/DD/YYYY')) DESDE mi_tabla;`
SQL Server	`SELECT YEAR(CONVERT(CHAR, str_column, 101)) FROM mi_tabla;`
SQLite	`SELECT SUBSTR(str_columna, 7) FROMmi_tabla;`

NOTA

SQLite no dispone de funciones de fecha y hora, pero se puede utilizar la función SUBSTR (subcadena) para extraer los cuatro últimos dígitos de .

Funciones nulas

Las funciones nulas pueden aplicarse a cualquier tipo de columna y se activan en cuando se encuentra un valor nulo.

Devolver un valor alternativo si hay un valor nulo

Utilice la función `COALESCE`. He

aquí una tabla de ejemplo:

```
+------+ ----------+
| id|  saludo |
+------+ ----------+
|1    | hola |
|2    | ¡hola!     |
|3    | NULL       |
+------+ ----------+
```

Cuando no haya saludo, devuelve el `hola`:

```
SELECT COALESCE(saludo, 'hola') AS saludo
FROM mi_tabla;

+----------+
| saludo |
+----------+
| hola |
| ¡Hola!    |
| Hola      |
+----------+
```

MySQL y *SQLite* también aceptan `IFNULL(greeting, 'hi')`.

Oracle también acepta `NVL(greeting, 'hola')`.

SQL Server también acepta `ISNULL(greeting, 'hi')`.

Conceptos avanzados de consulta

Este capítulo cubre algunas formas avanzadas de manejar datos mediante consultas SQL, más allá de las seis cláusulas principales cubiertas en el Capítulo 4, *Fundamentos de las consultas*, y las palabras clave comunes cubiertas en el Capítulo 7, *Operadores y funciones*.

La Tabla 8-1 incluye descripciones y ejemplos de código de los cuatro conceptos de tratados en este capítulo.

Tabla 8-1. Conceptos avanzados de consulta

Concepto	DescripciónCódigo Ejemplo
Casos prácticos	Si se cumple una condición, devuelve un valor determinado. En caso contrario, devuelve otro valor.
Agrupar y resumir	Divide los datos en grupos, agrega los datos dentro de cada grupo y devuelve un valor para cada *grupo*.

```
SELECT
house_id,
  CASE WHEN
  flg = 1
  ENTONCES
  "en venta
  ELSE
  "vendido"
  END
DE las
casas;

SELECCIONE
zip, AVG(ft)
DE casas
GROUP BY
zip;
```

Concepto	Descripción	Código Ejemplo
Funciones de ventana	Divida los datos en grupos, agregue u ordene los datos dentro de cada grupo y devuelva un valor para cada *fila*.	`SELECT zip, ROW_NUMBER() OVER (PARTITION BY zip ORDER BY price) FROM casas;`
Pivotar y despivotar	Convierta los valores de una columna en varias columnas o consolide varias columnas en una sola. Compatible con *Oracle* y *SQL Server*.	`-- Sintaxis Oracle SELECT * FROM listing_info PIVOT (COUNT(*) FOR room IN ('bd','br'));`

Este capítulo describe detalladamente cada uno de los conceptos de la Tabla 8-1, junto con casos de uso comunes.

Casos prácticos

Una sentencia `CASE` se utiliza para aplicar la lógica if-else dentro de una consulta. Por ejemplo, puede utilizar una sentencia `CASE` para deletrear valores. Si se ve un `1`, muestra `vip`. Si no, muestra `admisión general`.

```
+--------+       +-------------------+
| ticket |       | Billete           |
+--------+       +-------------------+
    |1 |         | VIP               |
    |0 |    -->    | Entrada general
    |1 |         | VIP               |
+--------+       +-------------------+
```

En *Oracle*, también puede ver la función `DECODE`, que es una función más antigua que opera de forma similar a la sentencia `CASE`.

NOTA

El uso de una sentencia `CASE` actualiza temporalmente los valores mientras dura la consulta. Para guardar los valores actualizados, puede hacerlo con una `sentencia UPDATE`.

Las dos secciones siguientes abordan dos tipos de `CASE` declaraciones:

- Sentencia `CASE` *simple* para *una sola columna* de datos
- Sentencia `CASE` *buscada* para *varias columnas* de datos

Mostrar valores basados en la lógica Si-Entonces para una sola columna

Para comprobar la igualdad dentro de una única columna de datos, utilice la función
sintaxis *simple de* la sentencia

`CASE`. Nuestro objetivo:

En lugar de mostrar los valores `1`/`0`/`NULL`, muestre los valores `vip`/`plazas reservadas`/`admisión general`:

- Si `flag = 1`, `ticket = vip`
- Si `la bandera = 0`, entonces el `billete = asiento reservado`
- Si no, `billete = entrada general`

He aquí una tabla de muestra:

```
SELECT * FROM concierto;

+-------+-------+
| Nombre Bandera
+-------+-------+
| anton |1 |
| julia |0 |
```

```
| maren |1 |
| sarah | NULL |
+-------+-------+
```

Implementa la lógica if-else con una simple sentencia CASE:

```
SELECCIONAR nombre, bandera,
   CASE flag WHEN 1 THEN 'vip'
   WHEN 0 THEN 'asientos
   reservados'
   ELSE 'general admission' END AS ticket
DE concierto;

+-------+------+ --------------------+
| Nombre Bandera Entrada           |
+-------+------+ --------------------+
| anton |1     | vip                |
| julia |0     | asientos reservados |
| maren |1     | vip                |
| sarah | NULL | admisión general |
+-------+------+ --------------------+
```

Si no coincide ninguna cláusula WHEN y no se especifica ningún valor ELSE, se emitirá un
Se devolverá NULL.

Mostrar Valores Basados en la Lógica Si-Entonces para Múltiples Columnas

Para comprobar cualquier condición (=, <, IN, IS NULL, etc.) dentro de varias columnas de datos, utilice *la* sintaxis de la sentencia CASE.

Nuestro objetivo:

En lugar de mostrar los valores 1/0/NULL, muestre los valores vip/plazas reservadas/admisión general:

- Si name = anton, entonces ticket = vip
- Si bandera = 0 o bandera = 1, entonces billete = asiento reservado
- Si no, billete = entrada general

He aquí una tabla de muestra:

```
SELECT * FROM concierto;

+-------+-------+
| Nombre Bandera
+-------+-------+
| anton |1 |
| julia |0 |
| maren |1 |
| sarah | NULL |
+-------+-------+
```

Implemente la lógica if-else con una sentencia `CASE` buscada:

```
SELECCIONAR nombre, bandera,
   CASE WHEN name = 'anton' THEN 'vip'
   WHEN flag IN (0,1) THEN 'asientos
   reservados' ELSE 'admisión general' END AS
   ticket
DE concierto;

+-------+------+ --------------------+
| Nombre Bandera Entrada           |
+-------+------+ --------------------+
| anton |1     | vip                 |
| julia |0     | asientos reservados |
| maren |1     | asientos reservados |
| sarah | NULL | admisión general |
+-------+------+ --------------------+
```

Si se cumplen varias condiciones, prevalece la primera.

NOTA

Para sustituir todos los valores `NULL` de una columna por otro valor, podría utilizar una sentencia `CASE`, pero es más habitual utilizar la función `NULL COALESCE` en su lugar.

Agrupar y resumir

SQL permite separar las filas en grupos y resumir las filas dentro de cada grupo de alguna manera, devolviendo finalmente sólo una fila por grupo.

La Tabla 8-2 enumera los conceptos asociados a la agrupación y suma de datos.

Tabla 8-2. Agrupación y resumen de conceptos

Categoría	Palabra clave	Descripción
El concepto principal	GRUPO POR	Utilice la cláusula GROUP BY para separar las filas de datos en grupos.
Formas de resumir filas dentro de cada grupo	COUNT SUM MIN MAX AVG	Estas funciones de agregación resumen varias filas de datos en *un único valor*.
	ARRAY_AGG GROUP_CONCAT LISTAGG STRING_AGG	Estas funciones combinan varias filas de datos en *una sola lista*.
Ampliaciones de la Cláusula GROUP BY	ROLLUP	Incluye filas para subtotales y también el total general.
	CUBO	Incluye agregaciones para todas las combinaciones posibles de las columnas agrupadas por.
	CONJUNTOS DE AGRUPACIÓN	Permite especificar las agrupaciones concretas que se mostrarán.

Conceptos básicos de GROUP BY

La siguiente tabla muestra el número de calorías quemadas por

dos personas:

```
SELECT * FROM entrenamientos;
```

```
+------+ ----------+
| nombre | calorías |
+------+ ----------+
| ally          |80 |
| ...aliado...          75...
| ally          |90 |
| jess         |100 |
| jess          |92 |
+------+ ----------+
```

Para crear un cuadro resumen, tienes que decidir cómo hacerlo:

1. Agrupa los datos: separa todos los valores de `nombre` en dos grupos: `ally` y `jess`.
2. Agregue los datos dentro de los grupos: encuentre el total `calorías` dentro de cada grupo.

Utilice la cláusula `GROUP BY` para crear una tabla resumen:

```
SELECCIONAR nombre,
       SUM(calories) AS total_calories
FROM entrenamientos
GROUP BY nombre;
```

```
+------+ ----------------+
| name | total_calories |
+------+ ----------------+
| ally               |245 |
| ...jess...        192...
+------+ ----------------+
```

Encontrará más detalles sobre cómo funciona `GROUP BY` entre bastidores en la sección La cláusula GROUP BY del capítulo 4.

Agrupación por varias columnas

La siguiente tabla muestra el número de calorías quemadas por dos personas durante sus entrenamientos diarios:

```
SELECT * FROM entrenamientos_diarios;
```

```
+------+------+------------+-------------+
| id|  nombre | fecha|     calorías |
+------+------+------------+-------------+
|    1 | ally | 2021-03-03 |        80 |
|    1 | ally | 2021-03-04 |        75 |
|    1 | ally | 2021-03-05 |        90 |
|    2 | jess | 2021-03-03 |       100 |
|    2 | jess | 2021-03-05 |        92 |
+------+------+------------+-------------+
```

Cuando se escribe una consulta con una cláusula `GROUP BY` que agrupa por múltiples columnas y/o incluye múltiples agregaciones:

- La cláusula `SELECT` debe incluir todos los *nombres de columnas* y *agregaciones* que desea que aparezcan en la salida.
- La cláusula `GROUP BY` debe incluir los mismos *nombres de columna* que aparecen en la cláusula `SELECT`.

Utilice la cláusula `GROUP BY` para resumir las estadísticas de cada persona, devolviendo tanto el `id` como el `nombre` junto con dos agregaciones:

```
SELECCIONE id, nombre,
       COUNT(date) COMO
       entrenamientos,
       SUM(calories) COMO
       calorías
FROM entrenamientos_diarios
GROUP BY id, nombre;
```

```
+------+------+----------+-------------+
|      id| nombre | entrenamientos | calorías |
+------+------+----------+-------------+
|1    | aliado |3       |245 |
|2    | jess |2         |192 |
+------+------+----------+-------------+
```

Reducir la lista GROUP BY para mayor eficacia

Si sabe que cada `id` está vinculado a un único `nombre`, puede excluir la columna `name` de la cláusula `GROUP BY` y obtener los mismos resultados que en la consulta anterior:

```
SELECCIONE id,
       MAX(nombre) COMO nombre,
       COUNT(fecha) COMO
       entrenamientos,
       SUM(calorías) COMO
       calorías
FROM
entrenamientos_diar
ios GROUP BY id;
```

Esto se ejecuta de forma más eficiente entre bastidores, ya que la función `GROUP BY`
sólo tiene que producirse en una columna.

Para compensar la eliminación del `nombre` de la cláusula `GROUP BY`, observará que se ha aplicado una función agregada arbitraria (`MAX`) a la columna name dentro de la cláusula `SELECT`. Dado que sólo hay un valor de `nombre` dentro de cada grupo de id, `MAX(name)` simplemente devolverá el `nombre` asociado a cada `id`.

Agregar filas en un único valor o lista

Con la cláusula `GROUP BY`, debe especificar cómo deben resumirse las filas de datos de dentro de cada grupo utilizando cualquiera de las dos opciones:

- *Una función agregada para resumir filas en un único valor*: `COUNT`, `SUM`, `MIN`, `MAX` y `AVG`
- *Una función para resumir filas en una lista* (mostrada en la tabla de ejemplo): `GROUP_CONCAT` y otras enumeradas en la Tabla 8-3

He aquí una tabla de muestra:

```
SELECT * FROM entrenamientos;

+------+ ----------+
| nombre | calorías |
+------+ ----------+
| ally          |80 |
| ...aliado...          75...
| ally          |90 |
```

```
| jess         |100 |
| jess          |92 |
+------+ ----------+
```

Utilice `GROUP_CONCAT` en *MySQL* para crear una lista de calorías:

```
SELECCIONAR nombre,
       GROUP_CONCAT(calorías) AS calories_list
FROM entrenamientos
GROUP BY nombre;

+------+ ---------------+
| nombre | calories_list |
+------+ ---------------+
| ally | 80,75,90      |
| jess | 100,92        |
+------+ ---------------+
```

La función `GROUP_CONCAT` difiere para cada RDBMS. La Tabla 8-3 muestra la sintaxis soportada por cada RDBMS:

Tabla 8-3. Agregar filas en una lista en cada RDBMS

RDBMS	Separador	CodeDefault
MySQL	`GROUP_CONCAT(calorías)` `GROUP_CONCAT(calorías SEPARADOR ',')`	Coma
Oracle	`LISTAGG(calorías)` `LISTAGG(calorías, ',')`	Sin valor
PostgreSQL	`ARRAY_AGG(calorías)`	Coma
SQL Server	`STRING_AGG(calorías, ',')`	Separador obligatorio
SQLite	`GROUP_CONCAT(calorías)` `GROUP_CONCAT(calorías, ',')`	Coma

En *MySQL*, *Oracle* y *SQLite*, la parte del separador (`','`) es opcional. *PostgreSQL* no acepta separadores y *SQL Server* los requiere.

También puede devolver una lista ordenada o una lista única de valores. La Tabla 8-4 muestra la sintaxis soportada por cada RDBMS.

Tabla 8-4. Devolver una lista ordenada o única de valores en cada RDBMS

	RDBMSSorted ListLista	única
MySQL	`GROUP_CONCAT(calorías ORDER BY calories)`	`GROUP_CONCAT( DIS calorías TINCT)`
Oracle	`LISTAGG(calorías) WITHIN GROUP (ORDER BY calorías)`	`LISTAGG( DISTINCT calorías)`
PostgreSQL	`ARRAY_AGG(calorías ORDER BY calorías)`	`ARRAY_AGG( DIS calorías TINCT)`
SQL Server	`STRING_AGG(calorías, ',') WITHIN GROUP (ORDER BY calo ries)`	No se admite
	SQLiteNo soportadoGROUP_CONCAT	`( DIS TINCT calories)`

CONJUNTOS ROLLUP, CUBE y GROUPING

Además de `GROUP BY`, también puede añadir las palabras clave `ROLLUP`, `CUBE` o `GROUPING SETS` para incluir información resumida adicional .

En el siguiente cuadro se enumeran cinco compras en el transcurso de tres meses:

```
SELECT * FROM gastos;

 AÑO MES IMPORTE
----- ------ -------
 2019      1      20
 2019      1      30
 2020      1      42
 2020      2      37
 2020      2     100
```

Los ejemplos de esta sección se basan en los siguientes grupos `GROUP BY`
que devuelve el total de gastos mensuales:

```
SELECCIONE año, mes,
       SUM(importe) COMO total
```

```
DE gastos
GROUP BY año, mes
ORDER BY año, mes;

 AÑO MES TOTAL
----- ------ ------
 2019      1     50
 2020      1     42
 2020      2    137
```

ROLLUP

MySQL, *Oracle*, *PostgreSQL* y *SQL Server* soportan `ROLLUP`, que amplía el `GROUP BY` incluyendo filas adicionales para subtotales y el total general.

Utilice `ROLLUP` para visualizar también los gastos anuales y totales. Las líneas de gastos de 2019, 2020 y total se añaden con la adición de `ROLLUP`:

```
SELECCIONE año, mes,
       SUM(importe) AS total
FROM gastos
GROUP BY ROLLUP(año, mes)
ORDER BY año, mes;

 AÑO MES TOTAL
----- ------ ------
 2019      1     50
 2019            50 -- Gastos de 2019
 2020      1     42
 2020      2    137
 2020          179 -- Gastos en 2020
               229 -- Total de gastos
```

La sintaxis anterior funciona en *Oracle*, *PostgreSQL* y *SQL Server*. La sintaxis de *MySQL* es `GROUP BY year, month WITH ROLLUP`, que también funciona en *SQL Server*.

CUBO

Oracle, *PostgreSQL* y *SQL Server* soportan `CUBE`, que amplía el `ROLLUP` incluyendo filas adicionales para todas las combinaciones posibles de las columnas por las que se está agrupando, así como el total general.

Utilice `CUBE` para visualizar también los gastos mensuales (un solo mes a lo largo de varios años). Las líneas de gastos de enero y febrero se añaden con `CUBE`:

```
SELECCIONE año, mes,
       SUM(importe) AS total
FROM gastos
GROUP BY CUBE(año, mes)
ORDER BY año, mes;

 AÑO MES TOTAL
----- ------ ------
 2019      1     50
 2019            50
 2020      1     42
 2020      2    137
 2020           179
           1     92  -- Gastos de enero
           2    137  -- Gastos de febrero
                229
```

La sintaxis anterior funciona en *Oracle*, *PostgreSQL* y *SQL Server*. *SQL Server* también admite la sintaxis `GROUP BY year, month WITH CUBE`.

CONJUNTOS DE AGRUPACIÓN

Oracle, *PostgreSQL* y *SQL Server* admiten `CONJUNTOS DE AGRUPACIÓN`, que permiten especificar agrupaciones concretas que se desean mostrar en .

Estos datos son un subconjunto de los resultados generados por `CUBE`, que sólo incluyen agrupaciones de una columna cada vez. En este caso, sólo se muestran los gastos totales anuales y mensuales:

```
SELECCIONE año, mes,
       SUM(importe) AS total
FROM gastos
GROUP BY GROUPING SETS(año, mes)
ORDER BY año, mes;

 AÑO MES TOTAL
----- ------ ------
 2019             50
 2020            179
           1      92
           2     137
```

Funciones de ventana

Una función *ventana* (o *función analítica* en *Oracle*) es similar a una función agregada en el sentido de que ambas realizan un cálculo sobre filas de datos. La diferencia es que una función agregada devuelve un único valor, mientras que una función ventana devuelve un valor por cada fila de datos.

La siguiente tabla muestra una lista de empleados con sus ventas mensuales. Las siguientes consultas utilizan esta tabla para mostrar la diferencia entre una función agregada y una función de ventana.

```
SELECT * FROM ventas;

+-------+-------+ --------+
| Nombre Mes Ventas
+-------+-------+ --------+
| David |     3 |      2 |
| David |     4 |     11 |
| Laura |     3 |      3 |
| Laura |     4 |     14 |
| Laura |     5 |      7 |
| Laura |     6 |      1 |
+-------+-------+ --------+
```

Función agregada

`SUM()` es una función de suma. La siguiente consulta suma las ventas de cada persona y devuelve cada `nombre` junto con su valor `total_ventas`.

```
SELECCIONAR nombre,
       SUM(ventas) AS
total_ventas FROM ventas
GROUP BY nombre;

+-------+--------------+
| nombre  |   total_ventas |
+-------+--------------+
| David             |13 |
| Laura             |25 |
+-------+--------------+
```

Función de ventana

`ROW_NUMBER() OVER (PARTITION BY name ORDER BY month)` es un función de ventana. En la parte en negrita de la siguiente consulta, para cada persona se genera un número de fila que representa el primer mes, el segundo mes, etc. en que vendió algo. La consulta devuelve cada fila junto con su valor `sale_month`.

```
SELECCIONAR nombre,
       ROW_NUMBER() OVER (PARTITION BY name
       ORDER BY month) AS sale_month
FROM sales;

+-------+-------------+
| nombre  |   venta_mes |
+-------+-------------+
| David |1 |
| David |2 |
| Laura |1 |
| Laura |2 |
| Laura |3 |
| Laura |4 |
+-------+-------------+
```

Desglose de la función de ventana

```
ROW_NUMBER() OVER (PARTITION BY name ORDER BY month)
```

Una *ventana* es un grupo de filas. En el ejemplo anterior, había dos ventanas. El nombre `David` tenía una ventana de dos filas y el nombre `Laura` tenía una ventana de cuatro filas:

`ROW_NUMBER()`
: La función que desea aplicar a cada ventana. Otras funciones comunes son `RANK()`, `FIRST_VALUE()`, `LAG()`, etc. Esto es obligatorio.

`EN`
: Indica que está especificando una función de ventana. Esto es necesario.

`PARTICIÓN POR nombre`
: Indica cómo desea dividir los datos en ventanas. Puede dividirse según una o varias columnas. Es opcional. Si se excluye, la ventana es toda la tabla.

`ORDENAR POR mes`
: Indica cómo debe ordenarse cada ventana antes de aplicar la función . Es opcional en *MySQL*, *Post- greSQL* y *SQLite*. Es obligatorio en *Oracle* y *SQL Server*.

En las secciones siguientes se incluyen ejemplos de cómo se utilizan en la práctica las funciones de las ventanas.

Ordenar las filas de una tabla

Utilice la función `ROW_NUMBER()`, `RANK()` o `DENSE_RANK()` para añadir un número de fila a cada fila de una tabla.

La siguiente tabla muestra el número de bebés que han recibido nombres populares:

```
SELECT * FROM nombres_bebes;
```

```
+--------+--------+----------+
| Género Nombre   Bebés
+--------+--------+----------+
| F      | Emma   |     92 |
| F      | Mia    |     88 |
| F      | Olivia |    100 |
| M      | Liam   |    105 |
| M      | Mateo  |     95 |
| M      | Noah   |    110 |
+--------+--------+----------+
```

Las dos consultas siguientes:

- Ordenar los nombres por popularidad
- Clasificar los nombres por popularidad para cada sexo

Ordena los nombres por popularidad:

```
SELECCIONE género, nombre,
       ROW_NUMBER() SOBRE (
       ORDER BY babies DESC) AS popularity
FROM baby_names;
```

```
+--------+--------+--------------+
| género nombre   popularidad
+--------+--------+--------------+
| M      | Noah   |           1 |
| M      | Liam   |           2 |
| F      | Olivia |           3 |
| M      | Mateo  |           4 |
| F      | Emma   |           5 |
| F      | Mia    |           6 |
+--------+--------+--------------+
```

Ordena los nombres por popularidad para cada sexo:

```
SELECCIONAR sexo, nombre,
       ROW_NUMBER() OVER (PARTITION BY gender
       ORDER BY babies DESC) AS popularity
FROM baby_names;

+--------+--------+--------------+
```

```
| género          nombre popularidad
+--------+--------+--------------+
| F      | Olivia |            1 |
| F      | Emma   |            2 |
| F      | Mia    |            3 |
| M      | Noah   |            1 |
| M      | Liam   |            2 |
| M      | Mateo  |            3 |
+--------+--------+--------------+
```

ROW_NUMBER Versus RANK Versus DENSE_RANK

Existen tres métodos para sumar números de fila. Cada uno tiene una forma diferente de tratar los empates.

`ROW_NUMBER` rompe el empate:

```
NOMBRE   BEBÉS POPULARIDAD
------- ------- ------------
Olivia       99            1
Emma         80            2
Sophia       80            3
Mia          75            4
```

`RANK` mantiene el empate:

```
NOMBRE   BEBÉS POPULARIDAD
------- ------- ------------
Olivia       99            1
Emma         80            2
Sophia       80            2
Mia          75            4
```

`DENSE_RANK` mantiene el empate y no omite números:

```
NOMBRE   BEBÉS POPULARIDAD
Olivia ------- 99----------    1
Emma         80            2
Sophia       80            2
Mia          75            3
```

Devuelve el primer valor de cada grupo

Utilice `FIRST_VALUE` y `LAST_VALUE` para devolver la primera y la última fila de una ventana, respectivamente.

Las siguientes consultas desglosan el proceso en dos pasos para obtener el nombre más popular para cada sexo.

Paso 1: Mostrar el nombre más popular para cada sexo.

```
SELECCIONAR sexo, nombre, bebés,
       FIRST_VALUE(name) OVER (PARTITION BY gender
       ORDER BY babies DESC) AS top_name
FROM baby_names;

+--------+--------+--------+---------------+
| Género Nombre    | bebés top_name
+--------+--------+--------+---------------+
| F      | Olivia |    100 | Olivia    |
| F      | Emma   |     92 | Olivia    |
| F      | Mia    |     88 | Olivia    |
| M      | Noah   |    110 | Noah      |
| M      | Liam   |    105 | Noah      |
| M      | Mateo  |     95 | Noah      |
+--------+--------+--------+---------------+
```

Utilice la salida como subconsulta para el siguiente paso, que f i l m a la subconsulta.

Paso 2: Devolver sólo las dos filas que contienen los nombres más populares.

```
SELECT * FROM

(SELECCIONE sexo, nombre, bebés,
       FIRST_VALUE(name) OVER (PARTITION BY gender
       ORDER BY babies DESC) AS top_name
FROM baby_names) AS top_name_table

WHERE nombre = top_nombre;

+--------+--------+--------+---------------+
| Género Nombre    | bebés top_name
+--------+--------+--------+---------------+
| F      | Olivia      100 Olivia     |
```

```
| M       | Noah    |    110 | Noah      |
+--------+--------+--------+---------------+
```

En *Oracle*, excluya la parte `AS top_name_table`.

Devuelve el segundo valor de cada grupo

Utilice `NTH_VALUE` para devolver un número de rango específico dentro de cada ventana . *SQL Server* no soporta `NTH_VALUE`. En su lugar, consulte el código de la siguiente sección, Devuelva los dos primeros valores de cada grupo, pero sólo devuelva el segundo valor.

Las siguientes consultas desglosan el proceso en dos pasos para obtener el segundo nombre más popular para cada sexo.

Paso 1: Mostrar el segundo nombre más popular para cada sexo.

```
SELECCIONAR sexo, nombre, bebés,
      NTH_VALUE(name, 2) OVER (PARTITION BY gender
      ORDER BY babies DESC) AS second_name
FROM baby_names;
```

```
+--------+--------+--------+ ------------------+
| Género Nombre    | bebés segundo nombre
+--------+--------+--------+ ------------------+
| F       | Olivia |    100 | NULL          |
| F       | Emma   |     92 | Emma          |
| F       | Mia    |     88 | Emma          |
| M       | Noah   |    110 | NULL          |
| M       | Liam   |    105 | Liam          |
| M       | Mateo  |     95 | Liam          |
+--------+--------+--------+ ------------------+
```

El segundo parámetro de `NTH_VALUE(nombre, 2)` es el que especifica el segundo valor de la ventana. Puede ser cualquier número entero positivo.

Utilice la salida como subconsulta para el siguiente paso, que filma la subconsulta.

Paso 2: Devuelve sólo las dos filas que contienen los segundos nombres más populares.

```
SELECT * FROM

(SELECCIONE sexo, nombre, bebés,
```

```
        NTH_VALUE(name, 2) OVER (PARTITION BY gender
        ORDER BY babies DESC) AS second_name
FROM baby_names) AS second_name_table

WHERE nombre = segundo_nombre;

+--------+--------+--------+------------------+
| Género Nombre    | bebés segundo nombre
+--------+--------+--------+------------------+
| F      | Emma   |     92 Emma           |
| M      | Liam   |    105 Liam           |
+--------+--------+--------+------------------+
```

En *Oracle*, excluya la parte `AS segundo_nombre_tabla`.

Devolver los dos primeros valores de cada grupo

Utilice `ROW_NUMBER` dentro de una subconsulta para devolver varios números de rango dentro de cada grupo.

Las siguientes consultas desglosan el proceso en dos pasos para obtener el primer y el segundo nombre más popular de cada sexo.

Paso 1: Mostrar el rango de popularidad de cada género.

```
SELECCIONAR sexo, nombre, bebés,
       ROW_NUMBER() OVER (PARTITION BY gender
       ORDER BY babies DESC) AS
popularity FROM baby_names;

+--------+--------+--------+-----------------+
| Género y nombre | bebés popularidad
+--------+--------+--------+-----------------+
| F      | Olivia |    100 |           1 |
| F      | Emma   |     92 |           2 |
| F      | Mia    |     88 |           3 |
| M      | Noah   |    110 |           1 |
| M      | Liam   |    105 |           2 |
| M      | Mateo  |     95 |           3 |
+--------+--------+--------+-----------------+
```

Utilice la salida como subconsulta para el siguiente paso, que f i l m a la subconsulta.

Paso 2: Filtrar las filas que contienen los rangos 1 y 2.

```
SELECT * FROM

(SELECCIONE sexo, nombre, bebés,
       ROW_NUMBER() OVER (PARTITION BY gender
       ORDER BY babies DESC) AS popularity
FROM baby_names) AS popularity_table

WHERE popularidad IN (1,2);

+--------+--------+--------+-----------------+
| Género y nombre | bebés popularidad
+--------+--------+--------+-----------------+
| F      | Olivia      100 |           1 |
| F      | Emma   |     92 |           2 |
| M      | Noah   |    110 |           1 |
| M      | Liam   |    105 |           2 |
+--------+--------+--------+-----------------+
```

En *Oracle*, excluya la parte `AS popularity_table`.

Devuelve el valor de la fila anterior

Utilice `LAG` y `LEAD` para mirar un determinado número de filas por detrás y por delante, respectivamente.

Utilice `LAG` para volver a la fila anterior:

```
SELECCIONAR sexo, nombre, bebés,
       LAG(name) OVER (PARTITION BY gender
       ORDER BY babies DESC) AS prior_name
FROM nombres_bebé;

+--------+--------+--------+---------------+
| sexo nombre     bebés nombre anterior
+--------+--------+--------+---------------+
| F      | Olivia |    100 | NULL          |
| F      | Emma   |     92 | Olivia        |
| F      | Mia    |     88 | Emma          |
| M      | Noah   |    110 | NULL          |
| M      | Liam   |    105 | Noah          |
| M      | Mateo  |     95 | Liam          |
+--------+--------+--------+---------------+
```

Utilice `LAG(nombre, 2, 'Sin nombre')` para devolver los nombres de dos filas anteriores y sustituir los valores `NULL` por `Sin nombre`:

```
SELECT sexo, nombre, bebés,
       LAG(nombre, 2, 'Sin
       nombre') OVER (PARTITION
       BY sexo
       ORDER BY babies DESC) AS prior_name_2
FROM baby_names;
```

```
+--------+--------+--------+-----------------+
| género |        nombre| bebés | nombre_anterior_2 |
+--------+--------+--------+-----------------+
| F      | Olivia |    100 | Sin nombre    |
| F      | Emma   |     92 | Sin nombre    |
| F      | Mia    |     88 | Olivia        |
| M      | Noah   |    110 | Sin nombre    |
| M      | Liam   |    105 | Sin nombre    |
| M      | Mateo  |     95 | Noah          |
+--------+--------+--------+-----------------+
```

Las funciones `LAG` y `LEAD` reciben tres argumentos cada una: `LAG(nombre, 2, 'Ninguno')`

- `name` es el valor que desea devolver. Es obligatorio.
- `2` es el desplazamiento de fila. Es opcional y por defecto es 1.
- `No name'` es el valor que se devolverá cuando no haya ningún valor. Es opcional y por defecto es `NULL`.

Calcular la media móvil

Utilice una combinación de la función `AVG` y la función `FILAS ENTRE` para calcular la media móvil. He aquí

una tabla de ejemplo:

```
SELECT * FROM ventas;
```

```
+-------+-------+ --------+
| Nombre Mes Ventas
+-------+-------+ --------+
| David |     1 |      2 |
| David |     2 |     11 |
| David |     3 |      6 |
| David |     4 |      8 |
| Laura |     1 |      3 |
| Laura |     2 |     14 |
| Laura |     3 |      7 |
| Laura |     4 |      1 |
| Laura |     5 |     20 |
+-------+-------+ --------+
```

Para cada persona, halle la media móvil de tres meses de las ventas, desde dos meses antes hasta el mes en curso:

```
SELECCIONE nombre, mes, ventas,
       AVG(ventas) OVER (PARTITION BY
       nombre ORDER BY mes)
       FILAS ENTRE 2 PRECEDENTES Y
       FILA ACTUAL) tres_meses_ma
FROM ventas;
```

```
+-------+-------+-------+ ------------------+
| nombre  |   mes | ventas | tres_meses_ma |
+-------+-------+-------+ ------------------+
| David |     1 |      2 |          2.0000 |
| David |     2 |     11 |          6.5000 |
| David |     3 |      6 |          6.3333 |
| David |     4 |      8 |          8.3333 |
| Laura |     1 |      3 |          3.0000 |
| Laura |     2 |     14 |          8.5000 |
| Laura |     3 |      7 |          8.0000 |
| Laura |     4 |      1 |          7.3333 |
| Laura |     5 |     20 |          9.3333 |
+-------+-------+-------+ ------------------+
```

NOTA

El ejemplo anterior examina las dos filas anteriores a la fila actual:

```
FILAS ENTRE 2 PRECEDENTES Y LA FILA ACTUAL
```

También puede consultar las filas siguientes utilizando el `siguiente método` palabra clave:

```
FILAS ENTRE 2 PRECEDENTES Y 3 SIGUIENTES
```

Estas gamas se denominan a veces *ventanas correderas*.

Calcular el total

Utilice una combinación de la función `SUM` y la cláusula `ROWS BETWEEN UNBOUNDED` para calcular el total acumulado.

Para cada persona, calcula el total de ventas hasta el mes en curso:

```
SELECCIONE nombre, mes, ventas,
       SUM(ventas) OVER (PARTITION BY
       nombre ORDER BY mes)
       FILAS ENTRE PRECEDENTES NO LIMITADOS Y
       FILA ACTUAL) running_total
FROM ventas;
```

```
+-------+-------+-------+ -----------------+
| nombre mes ventas total actual
+-------+-------+-------+ -----------------+
| David |     1 |     2 |                2 |
| David |     2 |    11 |               13 |
| David |     3 |     6 |               19 |
| David |     4 |     8 |               27 |
| Laura |     1 |     3 |                3 |
| Laura |     2 |    14 |               17 |
| Laura |     3 |     7 |               24 |
| Laura |     4 |     1 |               25 |
| Laura |     5 |    20 |               45 |
+-------+-------+-------+ -----------------+
```

NOTA

Aquí calculamos el total acumulado de cada persona. Para calcular el total acumulado de toda la tabla, puede eliminar la parte del código relativa a `PARTITION BY name`.

FILAS frente a RANGO

Una alternativa a FILAS `ENTRE` es `RANGO ENTRE`. La siguiente consulta calcula el total de las ventas realizadas por todos los empleados, utilizando las palabras clave `FILAS` y `RANGO`:

```
SELECCIONE mes, nombre,
  SUM(sales) OVER (ORDER BY month ROWS BETWEEN
  UNBOUNDED PRECEDING AND CURRENT ROW) rt_rows,
  SUM(sales) OVER (ORDER BY month RANGE BETWEEN
  UNBOUNDED PRECEDING AND CURRENT ROW) rt_range
DE ventas;

+-------+-------+---------+-----------------+
| mes | nombre | rt_rows | rt_range |
+-------+-------+---------+-----------------+
|     1 | David |       2 |         5 |
|     1 | Laura |       5 |         5 |
|     2 | David |      16 |        30 |
|     2 | Laura |      30 |        30 |
|     3 | David |      36 |        43 |
|     3 | Laura |      43 |        43 |
|     4 | David |      51 |        52 |
|     4 | Laura |      52 |        52 |
|     5 | Laura |      72 |        72 |
+-------+-------+---------+-----------------+
```

La diferencia entre los dos es que `RANGE` devolverá el mismo valor total para cada `mes` (ya que los datos se ordenaron por `mes`), mientras que `ROWS` tendrá un valor total diferente para cada fila.

Pivotar y despivotar

Oracle y *SQL Server* admiten las operaciones `PIVOT` y `UNPIVOT`. `PIVOT` toma una sola columna y la divide en varias columnas. `UNPIVOT` toma varias columnas y las consolida en una sola.

Descomponer los valores de una columna en varias columnas

Imagina que tienes una tabla en la que cada fila es una persona, seguida de una fruta que comió ese día. Quieres tomar la columna de la `fruta` y crear una columna separada para cada fruta.

He aquí una tabla de muestra:

```
SELECT * FROM frutas;

+------+-------+ ---------------+
| id|  nombre  |   fruta       |
+------+-------+ ---------------+
|1    | Henry | fresas |
|2    | Henry | pomelo        |
|3    | Henry | sandía        |
|4    | Lirio  |   fresas |
|5    | Lirio  |   Sandía      |
|6    | Lirio  |   fresas |
|7    | Lirio  |   sandía      |
+------+-------+ ---------------+
```

Resultado esperado:

```
+-------+-------------+-----------+----------------+
| nombre  |   fresas | pomelo | sandía |
+-------+-------------+-----------+----------------+
| Henry |1 |1 |1 |
| Lily |2 |0 |2 |
+-------+-------------+-----------+----------------+
```

Utilice la operación `PIVOT` en *Oracle* y *SQL Server*:

```
-- Oracle
SELECT *
DE frutas
PIVOT
(COUNT(id) FOR fruta IN ('fresas',
                pomelo", "sandía"));

-- SQL Server
SELECT *
DE frutas
PIVOT
(COUNT(id) FOR fruta IN ([fresas],
                [pomelo], [sandía])
) AS frutas_pivot;
```

En la sección `PIVOT`, se hace referencia a las columnas `id` y `fruta`, pero no a la columna nombre. Por lo tanto, la columna de `nombre` permanecerá como su propia columna en el resultado final y cada fruta se convertirá en una nueva columna.

Los valores de la tabla son el recuento del número de filas de la tabla original que contenían cada combinación concreta de `nombre` y `fruta`.

Alternativa PIVOT: CASO

Una forma más manual de hacer un `PIVOT` es utilizar un estado CASE en su lugar en *MySQL*, *PostgreSQL* y *SQLite* ya que no soportan `PIVOT`.

```
SELECCIONAR nombre,
       SUM(CASE WHEN fruta = 'fresas'
            THEN 1 ELSE 0 END) AS fresas,
       SUM(CASE WHEN fruta = 'pomelo'
            THEN 1 ELSE 0 END) AS pomelo,
       SUM(CASE WHEN fruta = 'sandía'
            THEN 1 ELSE 0 END) AS sandía
FROM frutas
GROUP BY
nombre ORDER
BY nombre;
```

Listar los valores de varias columnas en una sola columna

Imagina que tienes una tabla en la que cada fila es una persona seguida de varias columnas que contienen sus frutas favoritas. Quieres reorganizar los datos para que todas las frutas estén en una columna.

He aquí una tabla de muestra:

```
SELECT * FROM frutas_favoritas;
```

```
+----+-------+-----------+-----------+ ---------------+
| id | name  |   fruit_one | fruit_two | fruit_thr |
+----+-------+-----------+-----------+ ---------------+
| 1 | Anna  |            manzana| plátano        |
                        |
| 2 | Barry | frambuesa |            |           |
| 3 | Liz| limón| lima| naranja       |
| 4 | Tom| melocotón|    pera| ciruela           |
+----+-------+-----------+-----------+ ---------------+
```

Resultado esperado:

```
+----+-------+-----------+---------+
| id | nombre  |   fruta|       rango |
+----+-------+-----------+---------+
|  1 | Anna  | manzana   |    1 |
|  1 | Anna  | plátano   |    2 |
|  2 | Barry | frambuesa |    1 |
|  3 | Liz   | limón     |    1 |
|  3 | Liz   | lima      |    2 |
|  3 | Liz   | naranja   |    3 |
|  4 | Tom   | melocotón |    1 |
|  4 | Tom   | pera      |    2 |
|  4 | Tom   | ciruela   |    3 |
+----+-------+-----------+---------+
```

Utilice la operación `UNPIVOT` en *Oracle* y *SQL Server*:

```
-- Oracle
SELECT *
FROM frutas_favoritas
UNPIVOT
```

```
(fruta FOR rango IN (fruta_uno AS 1,
```

```
    fruta_dos AS 2,
    fruit_thr AS 3));

-- SQL Server
SELECT *
FROM frutas_favoritas
UNPIVOT
(fruta FOR rango IN (fruta_uno,
                     fruta_dos,
                     fruta_thr)
) AS fruit_unpivot
WHERE fruit <> '';
```

La sección `UNPIVOT` toma las columnas `fruit_one`, `fruit_two` y `fruit_thr` y las consolida en una sola columna llamada `fruit`.

Una vez hecho esto, puede seguir adelante y utilizar una sentencia `SELECT` típica para extraer las columnas `id` y `name` originales junto con la columna `fruit` recién creada.

Alternativa UNPIVOT: UNION ALL

Una forma más manual de hacer un `UNPIVOT` es utilizar `UNION ALL` en *MySQL*, *PostgreSQL* y *SQLite* ya que no soportan `UNPIVOT`.

```
WITH all_fruits
AS (SELECT id,
name,
        fruta_una como fruta,
        1 AS rank
FROM
frutas_favoritas
UNION ALL
SELECCIONE id, nombre,
       fruta_dos como fruta,
       2 AS rank
FROM
frutas_favoritas
UNION ALL
SELECCIONE id, nombre,
       fruta_tres como fruta,
       3 Rango AS
FROM frutas_favoritas)
```

```
SELECCIONAR *
FROM all_fruits
WHERE fruta <> ''
ORDER BY id, name, fruit;
```

MySQL no soporta insertar una constante en una columna dentro de una consulta (`1 AS rank`, `2 AS rank`, y `3 AS rank`). Elimine esas líneas para que el código se ejecute.

CAPÍTULO 9

Trabajar con múltiples tablas y consultas

En este capítulo se explica cómo unir varias tablas mediante operadores de unión, y también cómo trabajar con varias consultas utilizando expresiones comunes de tabla.

La Tabla 9-1 incluye descripciones y ejemplos de código de los tres conceptos tratados en este capítulo.

Tabla 9-1. Trabajar con múltiples tablas y consultas

Concepto	Descripción	Código Ejemplo
Unir tablas	Combinar las columnas de dos tablas basándose en filas coincidentes.	`SELECT c.id, l.city FROM clientes c INNER JOIN loc l ON c.lid = l.id;`
Operarios sindicales	Combinar las filas de dos tablas basándose en columnas coincidentes.	`SELECT nombre, ciudad FROM empleados; UNION SELECT nombre, ciudad FROM clientes;`

Concepto	Descripción	Código Ejemplo

Expresiones comunes de tabla	Guarda temporalmente la salida de una consulta, para que otra consulta haga referencia a ella. También incluye consultas recursivas y jerárquicas.	`WITH mi_cte AS ( SELECT nombre, SUM(pedido_id ) COMO numero_pedido s FROM clientes GROUP BY nombre)`
		
		`SELECT MAX(num_orders) FROM mi_cte;`

Unir tablas

En SQL, *unir* significa combinar datos de varias tablas en una sola consulta. Las dos tablas siguientes muestran el estado en el que vive una persona y las mascotas que posee:

```
--                    estados-- mascotas
+------+-------++------+ ------------+
| nombre | estado ||    nombre | mascota |
+------+-------++------+ ------------+
| Ada,   AZ          || Deb, perro
| Deb  |   DE        || Deb, Pato.
+------+-------+|     Pat  |   cerdo |
                     +------+ ------+
```

Utilice la cláusula `JOIN` para unir las dos tablas en una sola:

```
SELECCIONAR *
FROM estados s INNER JOIN mascotas p
     ON s.name = p.name;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
                | Deb  |   DE|   Deb  |   dog |
                | Deb  |   DE|   Deb  |   pato |
+------+-------+------+ --------+
```

La tabla resultante sólo incluye filas para `Deb`, ya que está presente en ambas tablas.

Las dos columnas de la izquierda son de la `tabla de estados` y las dos de la derecha son de la tabla de `mascotas`. Las columnas de la salida se pueden consultar en con los alias `s.name`, `s.state`, `p.name` y `p.pet`.

Desglose de la cláusula JOIN

```
states s INNER JOIN pets p ON s.name = p.name
```

Tablas (`estados`, `mascotas`*)*
: Las mesas que nos gustaría combinar.

Alias (`s`, `p`*)*
: Son apodos para las mesas. Esto es opcional, pero se recomienda por simplicidad. Sin alias, la cláusula `ON` podría escribirse como `estados.nombre = mascotas.nombre`.

Tipo de unión (`INNER JOIN`*)*
: La parte `INNER` especifica que sólo se devolverán las filas coincidentes. Si sólo se escribe `JOIN`, entonces por defecto es un `INNER JOIN`. En la Tabla 9-2 se pueden encontrar otros tipos de join.

Join Condición (`ON s.name = p.name`*)*
: Condición que debe cumplirse para que dos filas se consideren coincidentes. Igual (=) es el operador más común, pero también se pueden utilizar otros como no igual (`!=` o <>), >, <, `ENTRE`, etc.

Además del `INNER JOIN`, la Tabla 9-2 enumera los distintos tipos de uniones en SQL. La siguiente consulta muestra el formato general para unir tablas:

```
SELECCIONAR *
FROM estados s [JOIN_TYPE]
     mascotas p ON s.name =
     p.name;
```

Sustituya la parte en negrita `[JOIN_TYPE]` por las palabras clave de la columna Keyword para obtener los resultados que se muestran en la columna Resulting Rows. Para el tipo de unión

`CROSS JOIN`, excluya la cláusula `ON` para obtener los resultados mostrados en la tabla.

Tabla 9-2. Formas de unir tablas

Palabra clave	Descripción	Filas resultantes
JOIN	Por defecto INNER JOIN.	nm | st | nm | pt -----+----+-----+------ Deb | DE | Deb | perro Deb | DE | Deb | pato
INNER JOIN	Devuelve las filas en común.	nm | st | nm | pt -----+----+-----+------ Deb | DE | Deb | perro Deb | DE | Deb | pato
LEFT JOIN	Devuelve las filas de la tabla izquierda y las filas coincidentes de la otra tabla.	nm | st | nm | pt -----+----+------+------ Ada | AZ | NULL | NULL Deb | DE | Deb | dog Deb | DE | Deb | duck
UNIÓN A LA DERECHA	Devuelve las filas de la tabla derecha y las filas coincidentes de la otra tabla.	nm | st | nm | pt ------+------+-----+------ DebDE Deb | perro DebDE Deb | pato NULL | NULL | Pat | cerdo
UNIÓN EXTERNA COMPLETA	Devuelve las filas de ambas tablas.	nm | st | nm | pt ------+------+------+------ AdaAZ NULL | NULL DebDE Deb | perro DebDE Deb | pato NULL | NULL | Pat | cerdo
CROSS JOIN	Devuelve todas las combinaciones de filas de las dos tablas.	nm | st | nm | pt -----+----+-----+------ Ada | AZ | Deb | perro Ada | AZ | Deb | pato Ada | AZ | Pat | cerdo Deb | DE | Deb | perro Deb | DE | Deb | pato Deb | DE | Pat | cerdo

Además de unir tablas usando la sintaxis estándar `JOIN ... ON ...`, la Tabla 9-3 enumera otras formas de unir tablas en SQL.

Tabla 9-3. Sintaxis para unir tablas

Tipo	Descripción	Código
JOIN ... ON ... Sintaxis	Sintaxis de unión más común que funciona con INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN y CROSS JOIN.	SELECCIONAR * DE los estados s **INNER JOIN** mascotas p **ON** s.name = p.name;
Atajo USING de la tecla	Utilice USING en lugar de la cláusula ON si los nombres de las columnas en las que se está uniendo coinciden.	SELECCIONAR * DE los estados INNER JOIN mascotas **USO DE (nombre)**;
UNIÓN NATURAL Atajo	Utilice NATURAL JOIN en lugar de INNER JOIN si los nombres de todas las columnas a las que se está uniendo coinciden.	SELECCIONAR * DE los estados **NATURAL ÚNETE a** las mascotas;
Sintaxis de unión antigua	Devuelve todas las combinaciones de las filas de dos tablas. Equivale a un CROSS JOIN.	SELECCIONAR * DE **estados s, mascotas p** WHERE s.name = p.name;

Tipo	Descripción	Código
Autounión	Utilice la sintaxis old join o new join para devolver todas l a s combinaciones de las filas de una tabla consigo misma.	>SELECT * FROM **estados s1, estados s2** WHERE s1.region = s2.region; SELECCIONAR * DE **estados s1** INNER JOIN **estados s2** WHERE s1.region = s2.region;

Las siguientes secciones describen en detalle los conceptos de las Tablas 9-2 y 9-3.

Join Basics y INNER JOIN

Esta sección explica cómo funciona conceptualmente una unión (join), en , así como la sintaxis básica de la unión utilizando un `INNER JOIN`.

Fundamentos de la adhesión

Puede pensar en unir tablas en dos pasos:

1. Mostrar todas las combinaciones de filas de las tablas.
2. Filtra las filas que tienen valores coincidentes.

Aquí hay dos mesas que nos gustaría unir:

```
--                    estados-- mascotas
+------+-------++------+ ------------+
| nombre | estado ||    nombre | mascota |
+------+-------++------+ ------------+
| Ada,   AZ          || Deb, perro
| Deb   |   DE       || Deb, Pato.
+------+-------+|     Pat   |   cerdo |
                     +------+ ------+
```

Paso 1: Visualizar todas las combinaciones de filas.
Al enumerar los nombres de las tablas en la cláusula `FROM`, se devuelven todas las combinaciones posibles de filas de las dos tablas.

```
SELECCIONAR *
DE estados, mascotas;

+------+-------+------+----------+
| nombre | estado | nombre | mascota |
+------+-------+------+----------+
| Ada, AZ      | Deb, perro
| Deb, DE      | Deb | perro |
| Ada, AZ      | Deb | pato |
| Deb, DE      | Deb | pato |
| Ada, AZ      | Pat | cerdo |
| Deb, DE      | Pat | cerdo |
+------+-------+------+----------+
```

La sintaxis `FROM states, pets` es una forma antigua de hacer una unión en SQL. Una forma más moderna de hacer lo mismo es usar un `CROSS JOIN`.

Paso 2: Filtre las filas que tengan nombres coincidentes.
Es probable que no desee mostrar todas las combinaciones de filas de las dos tablas, sino sólo las situaciones en las que coincida la columna `del nombre` de ambas tablas.

```
SELECCIONAR *
DE estados s, mascotas p
WHERE s.name = p.name;

+------+-------+------+----------+
| nombre | estado | nombre | mascota |
+------+-------+------+----------+
| Deb, DE      | Deb | perro |
| Deb, DE      | Deb | pato |
+------+-------+------+----------+
```

La línea `Deb/DE` aparece dos veces porque coincide con dos `Deb` en la tabla de `mascotas`.

El código anterior es equivalente a un `INNER JOIN`.

NOTA

El proceso de dos pasos descrito anteriormente es puramente conceptual. Las bases de datos rara vez realizan una unión cruzada al ejecutar una unión, sino que lo hacen de una forma más optimizada.

Sin embargo, pensar en estos términos conceptuales le ayudará a escribir correctamente las consultas join y a comprender sus resultados.

INNER JOIN

La forma más habitual de unir dos tablas es mediante una tabla `INNER JOIN`, que devuelve las filas que están en ambas tablas.

Utilice `INNER JOIN` *para devolver sólo las personas de ambas tablas*

```
SELECCIONAR *
FROM estados s INNER JOIN mascotas p
     ON s.name = p.name;
```

```
+------+-------+------+----------+
| nombre | estado | nombre | mascota |
+------+-------+------+----------+
| Deb, DE      | Deb | perro |
| Deb, DE      | Deb | pato |
+------+-------+------+----------+
```

Unir más de dos tablas

Esto puede hacerse mediante la inclusión de conjuntos adicionales del `JOIN .. ON ..` palabras clave:

```
SELECCIONAR *
DE los estados s
     INNER JOIN mascotas p
        ON s.name = p.name
     INNER JOIN almuerzo l
        ON s.name = l.name;
```

Unir en más de una columna

Para ello, incluya condiciones adicionales en la cláusula `ON`. Imagine que desea unir las siguientes tablas en `nombre` y `edad`:

```
-- states_ages             -- pets_ages
+------+-------+-----++------+-----+ ---------------+
| Nombre Estado Edad      | nombre mascota edad
+------+-------+-----++------+-----+ ---------------+
| Ada AK       | 25 |     | Ada Ant 30
| Ada AZ       | 30 |     | Pat, cerdo, 45.
+------+-------+-----++------+-----+ ---------------+

SELECCIONAR *
FROM states_ages s INNER JOIN pets_ages p
     ON s.name =
     p.name AND s.age
     = p.age;

+------+-------+------+------+------+ ------------+
| nombre | estado | edad | nombre | mascota | edad |
+------+-------+------+------+------+ ------------+
| Ada AZ       |   30 | Ada | ant |     30 |
+------+-------+------+------+------+ ------------+
```

LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN

Utilice `LEFT JOIN`, `RIGHT JOIN` y `FULL OUTER JOIN` para traer Juntar filas de dos tablas, incluidas las que no aparecen en ambas tablas.

LEFT JOIN

Utilice `LEFT JOIN` para devolver todas las personas de la tabla `states`. Las personas de la tabla de `estados` que no están en la tabla de `mascotas` se devuelven con valores `NULL`.

```
SELECCIONAR *
FROM estados s LEFT JOIN
     mascotas p ON s.nombre =
     p.nombre;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
```

```
| Ada   |   AZ| NULL | NULL |
                | Deb   |   DE|   Deb   |   dog |
| Deb   |   DE| Deb   |   pato |
+------+-------+------+ --------+
```

Una `LEFT JOIN` es equivalente a una `LEFT OUTER JOIN`.

UNIÓN A LA DERECHA

Utilice `RIGHT JOIN` para devolver todas las personas de la tabla `pets`. Las personas de la tabla de `mascotas` que no están en la tabla de `estados` se devuelven con valores `NULL`.

```
SELECCIONAR *
FROM estados s RIGHT JOIN
     mascotas p ON s.nombre =
     p.nombre;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
                | Deb   |   DE|   Deb   |   dog |
| Deb   |   DE| Deb   |   pato |
| NULL | NULL  |   Pat   |   cerdo |
+------+-------+------+ --------+
```

Una `RIGHT JOIN` es equivalente a una `RIGHT OUTER JOIN`.

SQLite no soporta `RIGHT JOIN`.

CONSEJO

El `LEFT JOIN` es mucho más común que el `RIGHT JOIN`. Si necesita una RIGHT `JOIN`, intercambie las dos tablas en la cláusula `FROM` y realice una `LEFT JOIN` en su lugar.

UNIÓN EXTERNA COMPLETA

Utilice `FULL OUTER JOIN` para obtener todas las personas de las tablas de `estados` y `mascotas`. Los valores que faltan en ambas tablas se devuelven con valores `NULL`.

```
SELECCIONAR *
FROM estados s FULL OUTER JOIN
     mascotas p ON s.nombre =
     p.nombre;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
| Ada  |   AZ| NULL | NULL |
               | Deb  |   DE|   Deb  |   dog |
| Deb  |   DE| Deb  |   pato |
| NULL | NULL  |   Pat  |   cerdo |
+------+-------+------+ --------+
```

Un `FULL OUTER JOIN` es equivalente a un `FULL JOIN`.

MySQL y *SQLite* no soportan `FULL OUTER JOIN`.

USO y UNIÓN NATURAL

Al unir tablas, para ahorrar tiempo, puede utilizar los métodos abreviados `USING` o `NATURAL JOIN` en lugar de la sintaxis estándar `JOIN ... ON . ON ..` estándar.

USO DE

MySQL, *Oracle*, *PostgreSQL* y *SQLite* admiten la función `USING` cláusula.

Puede utilizar el método abreviado `USING` en lugar de la cláusula `ON` para unir en dos columnas con el mismo nombre. La unión debe ser una unión equitativa (`=` en la cláusula `ON`) para utilizar `USING`.

```
-- Cláusula ON
SELECT *
FROM estados s INNER JOIN mascotas p
     ON s.name = p.name;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
               | Deb  |   DE|   Deb  |   dog |
```

```
| Deb  |  DE| Deb  |  pato |
```

```
+------+-------+------+ --------+

-- Equivalente USANDO el atajo
SELECT *
FROM estados INNER JOIN mascotas
     USANDO (nombre);

+------+-------+ -------+
| nombre | estado | mascota |
+------+-------+ -------+
| Deb  |   DE| dog |
| Deb  |   DE| pato |
+------+-------+ -------+
```

La diferencia entre las dos consultas es que la primera devuelve cuatro columnas, incluyendo `s.name` y `p.name`, mientras que la segunda devuelve tres columnas porque las dos columnas de `nombre` se fusionan en una sola y se llama simplemente `nombre`.

UNIÓN NATURAL

MySQL, *Oracle*, *PostgreSQL* y *SQLite* soportan `NATURAL JOIN`.

Puede utilizar el método abreviado `NATURAL JOIN` en lugar de la sintaxis `INNER JOIN .. ON ..` para unir dos tablas basándose en todas las columnas de que tengan exactamente el mismo nombre. La unión debe ser equi-join (`=` en la cláusula `ON`) para utilizar NATURAL JOIN.

```
-- INNER JOIN ... ON ... Y ... SELECT
*
FROM states_ages s INNER JOIN pets_ages p
     ON s.name = p.name
     AND s.age = p.age;

+------+-------+------+------+------+ ----------+
| nombre | estado | edad  |   nombre | mascota  |
edad |
+------+-------+------+------+------+ ----------+
| Ada  |   AZ      |30 | Ada  |   ant    |30 |
+------+-------+------+------+------+ ----------+
```

```
-- Acceso directo equivalente a NATURAL
JOIN SELECT *
FROM edades_estados NATURAL JOIN edades_mascotas;

+------+------+-------+ --------+
| Nombre Edad Estado Mascota
+------+------+-------+ --------+
| Ada |30   | AZ|     ant |
+------+------+-------+ --------+
```

La diferencia entre las dos consultas es que la primera devuelve seis columnas, incluyendo `s.nombre`, `s.edad`, `p.nombre` y `p.edad`, mientras que la segunda devuelve cuatro columnas porque las columnas duplicadas de `nombre` y `edad` se fusionan y se llaman simplemente `nombre` y `edad`.

ADVERTENCIA

Tenga cuidado al utilizar un `NATURAL JOIN`. Ahorra un poco de escritura, pero puede hacer una unión inesperada si se añade o elimina de una tabla una columna con el mismo nombre. Es mejor utilizarlo para consultas rápidas que para código de producción.

CROSS JOIN y Self Join

Otra forma de unir tablas consiste en mostrar todas las combinaciones de las filas de dos tablas. Esto se puede hacer con una `CROSS JOIN`. Si esto se hace en una tabla consigo misma, se llama *autounión*. Una autounión es útil cuando se desea comparar filas dentro de la misma tabla.

CROSS JOIN

Utilice `CROSS JOIN` para devolver todas las combinaciones de las filas de dos tablas. Equivale a enumerar las tablas en la cláusula `FROM` (lo que a veces se denomina "sintaxis join antigua").

```
-- CROSS JOIN
SELECT *
```

```
FROM estados CROSS JOIN mascotas;

-- Lista de tablas
equivalente SELECT *
DE estados, mascotas;

+------+-------+------+ --------+
| nombre | estado | nombre | mascota |
+------+-------+------+ --------+
| Ada  | AZ    | Deb  | perro |
| Deb  | DE    | Deb  | perro |
| Ada  | AZ    | Deb  | pato |
| Deb  | DE    | Deb  | pato |
| Ada  | AZ    | Pat  | cerdo |
| Deb  | DE    | Pat  | cerdo |
+------+-------+------+ --------+
```

Una vez listadas todas las combinaciones, puede optar por filtrar los resultados añadiendo una cláusula `WHERE` para devolver menos filas en función de lo que esté buscando.

Unirse uno mismo

Puede unir una tabla consigo misma mediante una autounión. Una autounión suele constar de dos pasos:

1. Mostrar todas las combinaciones de las filas de una tabla consigo misma.
2. Filtrar las filas resultantes en función de algunos criterios.

A continuación se presentan dos ejemplos de autouniones en la práctica. He aquí una tabla de empleados y sus jefes:

```
SELECT * FROM empleado;

+------+--------+----------+-----------+
| dept | emp_id | emp_name | mgr_id |
+------+--------+----------+-----------+
```

```
|           201 | lisa     |    101 |
tecnología
|
|           202 | monica   |    101 |
tecnología
|
| datos |   203 | nancy    |    201 |
| datos |   204 | olivia   |    201 |
| datos |   205 | penny    |    202 |
+------+--------+----------+-----------+
```

Ejemplo 1: Devolver una lista de empleados y sus jefes.

```
SELECT e1.emp_name, e2.emp_name as mgr_name
FROM empleado e1, empleado e2
WHERE e1.mgr_id = e2.emp_id;
```

```
+----------+ ------------+
| emp_name | mgr_name |
+----------+ ------------+
| Nancy    | Lisa     |
| Olivia   | Lisa     |
| Penny    | Monica   |
+----------+ ------------+
```

Ejemplo 2: Empareje a cada empleado con otro empleado de su departamento.

```
SELECT e.dept, e.emp_name, matching_emp.emp_name
FROM empleado e, empleado matching_emp
WHERE e.dept = matching_emp.dept
      AND e.emp_name <> matching_emp.emp_name;
```

```
+------+----------+ -------------+
| dept | emp_name | emp_name |
+------+----------+ -------------+
| tech | monica   | Lisa     |
| tech | lisa     | Monica   |
| data | penny    | nancy    |
| Olivia          | Nancy    |
| Datos           | Olivia   |
| Datos Nancy     | Olivia   |
| Datos Olivia    | penny    |
| data | nancy    | penny    |
+------+----------+ -------------+
```

NOTA

La consulta anterior tiene filas duplicadas (`monica/lisa` y `lisa/monica`). Para eliminar los duplicados y devolver sólo cuatro filas en lugar de ocho, puede añadir la línea

```
AND e.emp_name < matching_emp.emp_name
```

a la cláusula `WHERE` para que sólo devuelva filas en las que el primer nombre de esté antes que el segundo por orden alfabético. Este es el resultado sin duplicados:

```
+------+----------+------------+
| dept | emp_name | emp_name |
+------+----------+------------+
| tech |          lisa| monica      |
| Datos           Nancy Olivia      |
| Datos. Nancy. Penny.       |
| Datos Olivia Penny         |
+------+----------+------------+
```

Operarios sindicales

Utilice la palabra clave `UNION` para combinar los resultados de dos o más sentencias `SELECT` de . La diferencia entre `JOIN` y `UNION` es que `JOIN` une varias tablas en una sola consulta, mientras que `UNION` apila los resultados de varias consultas:

```
-- JOIN ejemplo
SELECT *
FROM cumpleaños b JOIN velas c
     ON b.nombre = c.nombre;

-- UNION ejemplo
SELECT * FROM
escritores UNION
SELECT * FROM artistas;
```

La figura 9-1 muestra la diferencia entre los resultados de un `JOIN` y una `UNION`, basada en el código anterior.

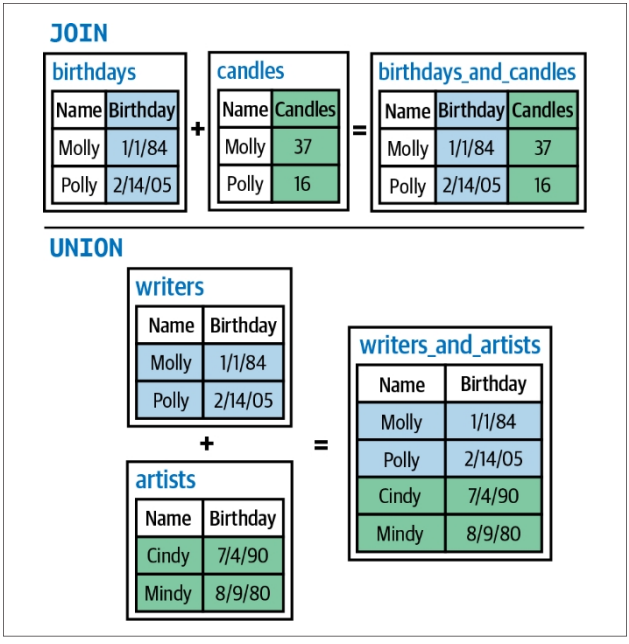

Figura 9-1. JOIN frente a UNION

Existen tres formas de combinar las filas de dos tablas . También se conocen como *operadores de unión*:

`UNIÓN`
: Combina los resultados de varias sentencias.

`EXCEPTO` *(*`MENOS` *en Oracle)*
: Devuelve los resultados menos otro conjunto de resultados.

`INTERSECT`
: Devuelve resultados superpuestos.

UNIÓN

La palabra clave `UNION` combina los resultados de dos o más `SELECT` en una sola salida.

Aquí hay dos tablas que nos gustaría combinar:

```
-- personal
+---------+----------+
| Nombre Origen
+---------+----------+
| michael | NULL     |
| NULL     |
| tahani   |   inglaterra |
+---------+----------+

-- residentes
+---------+---------+ --------------+
| Nombre  País Ocupación
+---------+---------+ --------------+
| eleanor usa temp   |
| chidi|  nigeria | profesor |
| tahani   |   inglaterra | modelo|
| jason usa dj       |
+---------+---------+ --------------+
```

Utilice `UNION` para combinar las dos tablas y eliminar las filas duplica d a s :

```
SELECT nombre, origen FROM personal
UNIÓN
SELECT nombre, país FROM residentes;

+---------+----------+
| Nombre Origen
+---------+----------+
| michael | NULL     |
| NULL     |
| tahani   |   inglaterra |
| eleanor | usa      |
| chidi   nigeria
| jason usa          |
+---------+----------+
```

Tenga en cuenta que `tahani/england` aparece tanto en la `plantilla` como en
`residentes`. Sin embargo, sólo aparece como una fila en la tabla

porque UNION elimina las filas duplicadas de la salida de .

¿Qué consultas puede unir?

Al realizar una UNION en dos consultas, algunas características de las consultas de deben coincidir y otras no.

Número de columnas: DEBE COINCIDIR
: Cuando se unen dos consultas, se debe especificar el mismo número de columnas en ambas consultas.

Nombres de columna: NO TIENEN QUE COINCIDIR
: No es necesario que los nombres de columna de las dos consultas coincidan para realizar una UNIÓN. Los nombres de columna utilizados en la primera sentencia SELECT de una consulta UNION se convierten en los nombres de las columnas de salida.

Tipos de datos: DEBEN COINCIDIR
: Los tipos de datos de las dos consultas deben coincidir para hacer una UNION. Si no coinciden, puede utilizar la función CAST para convertirlas al mismo tipo de datos antes de realizar la UNIÓN.

UNIÓN TODOS

Utilice UNION ALL para combinar las dos tablas y conservar las filas duplicadas:

```
SELECT nombre, origen FROM personal
UNIÓN TODOS
SELECT nombre, país FROM residentes;

+---------+----------+
| Nombre Origen
+---------+----------+
| michael | NULL     |
| NULL    |
| tahani  |  inglaterra |
| eleanor | usa      |
| chidi   nigeria
```

```
| tahani  |  inglaterra |
| jason usa          |
+---------+----------+
```

CONSEJO

Si sabe con certeza que no h a y filas duplicadas, utilice `UNION ALL` para mejorar el rendimiento. `UNION` realiza una ordenación adicional entre bastidores para identificar los duplicados.

UNIÓN con otras cláusulas

También puede incluir otras cláusulas cuando utilice `UNION`, como `WHERE`, `JOIN`, etc. Sin embargo, sólo se permite una cláusula `ORDER BY` para toda la consulta, y debe estar al final.

Filtra los valores nulos y ordena los resultados de una consulta `UNION`:

```
SELECT nombre, origen
FROM personal
WHERE origen IS NOT NULL

UNIÓN

SELECT nombre, país
FROM residentes

ORDER BY nombre;

+---------+----------+
| Nombre Origen
+---------+----------+
| chidi   nigeria
| eleanor | usa     |
| jason usa          |
| tahani  |  inglaterra |
+---------+----------+
```

UNION con más de dos tablas

Puede unir más de dos tablas incluyendo cláusulas `UNION` adicionales.

Combinar las filas de más de dos tablas:

```
SELECT nombre, origen
FROM personal

UNIÓN

SELECT nombre, país
FROM residentes

UNIÓN

SELECT nombre, país
FROM mascotas;
```

CONSEJO

`UNION` se utiliza normalmente para combinar resultados de varias tablas. Si va a combinar resultados de una sola tabla, es mejor que escriba una sola consulta y utilice la cláusula `WHERE`, la sentencia `CASE`, etc. adecuadas.

EXCEPTO e INTERSECCIÓN

Además de utilizar `UNION` para combinar las filas de varias tablas, puede utilizar `EXCEPT` e `INTERSECT` para combinar las filas de diferentes maneras.

EXCEPTO

Utilice `EXCEPT` para "sustraer" los resultados de una consulta de otra consulta.

Devolver a los miembros del personal que no sean residentes:

```
SELECT nombre FROM personal
EXCEPTO
SELECT nombre FROM residentes;

+---------+
| Nombre  |
+---------+
| michael |
| Janet   |
+---------+
```

MySQL no soporta `EXCEPT`. En su lugar, puede utilizar las palabras clave `NOT IN` como solución:

```
SELECT nombre
FROM personal
WHERE name NOT IN (SELECT name FROM residents);
```

Oracle utiliza `MINUS` en lugar de `EXCEPT`.

PostgreSQL también admite `EXCEPT ALL`, que no elimina duplicados. `EXCEPT` elimina todas las apariciones de un valor, mientras que EXCEPT `ALL` elimina instancias específicas.

INTERSECT

Utilice `INTERSECT` para encontrar las filas en común entre dos consultas .

Devuelve también a los miembros del personal que son residentes:

```
SELECT nombre, origen FROM personal
INTERSECT
SELECT nombre, país FROM residentes;

+---------+----------+
| Nombre Origen
+---------+----------+
| tahani  |  inglaterra |
+---------+----------+
```

MySQL no soporta `INTERSECT`. En su lugar, puede utilizar un `INNER JOIN` como solución:

```
SELECCIONE s.nombre, s.origen
FROM personal s INNER JOIN
     residentes r ON s.nombre =
     r.nombre;
```

PostgreSQL también admite `INTERSECT ALL`, que conserva los valores duplicados.

Operadores Sindicales: Orden de evaluación

Al escribir una sentencia con varios operadores de unión (`UNION`, `EXCEPT`, `INTERSECT`), utilice paréntesis para especificar el orden en que deben producirse las operaciones.

```
SELECT * FROM
personal EXCEPT
(SELECT * FROM residentes
UNION
SELECT * FROM mascotas);
```

A menos que se especifique lo contrario, los operadores de unión se realizan en orden descendente, excepto que `INTERSECT` tiene prioridad sobre `UNION` y `EXCEPT`.

Expresiones comunes de tabla

Una *expresión común de tabla (CTE) es* un conjunto de resultados temporal. En otras palabras, guarda temporalmente la salida de una consulta para que escriba otras consultas que hagan referencia a ella.

Puede reconocer una CTE cuando vea la palabra clave `WITH`. Existen dos tipos de CTE:

CTE no recursivo
: Una consulta para que otras consultas sirvan de referencia (véase "CTEs Versus Subqueries" en la página 293).

CTE recursivo
: Una consulta que hace referencia a sí misma (véase "CTEs recursivas" en la página 295).

NOTA

Las CTEs no recursivas son mucho más comunes que las recursivas. La mayoría de las veces, si alguien menciona una CTE, se está refiriendo a una CTE no recursiva.

He aquí un ejemplo de CTE no recursivo en la práctica:

```
-- Consultar los resultados de mi_cte
WITH mi_cte AS (
   SELECT name, AVG(grade) AS avg_grade
   FROM mi_tabla
   GROUP BY nombre)

SELECCIONAR *
FROM mi_cte
WHERE avg_grade < 70;
```

He aquí un ejemplo de CTE recursiva en la práctica:

```
-- Generar los números del 1 al 10
WITH RECURSIVE mi_cte(n) AS
(
  SELECT 1 -- Incluir FROM dual en Oracle
  UNION ALL
  SELECT n + 1 FROM mi_cte WHERE n < 10
)

SELECT * FROM mi_cte;
```

En *MySQL* y *PostgreSQL*, la palabra clave `RECURSIVE` es obligatoria. En *Oracle* y *SQL Server*, la palabra clave `RECURSIVE` debe omitirse en . *SQLite* funciona con ambas sintaxis.

En *Oracle*, es posible que veas código antiguo que utiliza la sintaxis `CONNECT BY` para consultas recursivas, pero las CTE son mucho más comunes hoy en día.

CTE frente a subconsultas

Tanto las CTE como las subconsultas permiten escribir una consulta y, a continuación, escribir otra consulta que haga referencia a la primera. Esta sección describe la diferencia entre ambos enfoques.

Imagine que su objetivo es encontrar el departamento con el salario medio más alto. Esto se puede hacer en dos pasos: escriba una consulta que devuelva el salario medio de cada departamento; utilice una CTE o subconsulta para escribir una consulta alrededor de la primera consulta que devuelva el departamento con el salario medio más alto.

Paso 1. Consulta del salario medio de cada departamento

```
SELECT dept, AVG(salary) AS avg_salary
FROM empleados
GROUP BY dept;

+-------+--------------+
| dept | avg_salary |
+-------+--------------+
| .            78000 |
| Ventas       61000 |
| Tecnología   83000 |
+-------+--------------+
```

Paso 2. CTE y subconsulta que buscan el departamento con el salario medio más alto utilizando la consulta anterior

```
-- Enfoque CTE
WITH avg_dept_salary AS (
   SELECT dept, AVG(salary) AS avg_salary
   FROM empleados
   GROUP BY dept)

SELECCIONAR *
FROM avg_dept_salary
ORDER BY avg_salary DESC
LIMIT 1;

-- Enfoque de subconsulta equivalente
SELECT *
FROM
```

```
(SELECT dept, AVG(salary) AS avg_salary
FROM empleados
GROUP BY dept) avg_dept_salary

ORDER BY avg_salary DESC
LIMIT 1;

+------+ -------------+
| dept | avg_salary |
+------+ -------------+
| Tecnología  83000 |
+------+ -------------+
```

La sintaxis de la cláusula `LIMIT` difiere según el software. Sustituya `LIMIT 1` por `ROWNUM = 1` en *Oracle* y `TOP 1` en *SQL Server*. Encontrará más detalles en la sección La cláusula LIMIT del Capítulo 4.

Ventajas de una CTE frente a una subconsulta

Utilizar una CTE en lugar de una subconsulta tiene algunas ventajas.

Referencias múltiples
: Una vez definida una CTE, puede hacer referencia a ella por su nombre varias veces en las consultas `SELECT` siguientes:

```
CON mi_cte COMO (. )

SELECT * FROM mi_cte WHERE id > 10
UNION
SELECT * FROM mi_cte WHERE puntuación > 90;
```

Con una subconsulta, tendría que escribir la subconsulta completa cada vez.

Mesas múltiples
: La sintaxis CTE es más legible cuando se trabaja con múltiples tablas porque se pueden listar todas las CTEs por adelantado:

```
CON mi_cte1 COMO (. ),
    mi_cte2 COMO (. )

SELECCIONAR *
```

```
    FROM mi_cte1 m1
        INNER JOIN mi_cte2 m2
        ON m1.id = m2.id;
```

Con las subconsultas, éstas se dispersarían por toda la consulta general.

Los CTEs *no* son *compatibles* con el software SQL más antiguo, por lo que todavía se utilizan habitualmente las subconsultas .

CTE recursivos

Esta sección recorre dos situaciones prácticas en las que sería útil un CTE recursivo .

Rellenar las filas que faltan en una secuencia de datos

La siguiente tabla incluye fechas y precios. Obsérvese que en la columna de fechas faltan datos para el segundo y el quinto día del mes.

```
SELECT * FROM precios_acciones;

+------------+ ---------+
| Fecha Precio
+------------+ ---------+
| 2021-03-01 | 668.27 |
| 2021-03-03 | 678.83 |
| 2021-03-04 | 635.40 |
| 2021-03-06 | 591.01 |
+------------+ ---------+
```

Rellene las fechas con un proceso de dos pasos:

1. Utilice un CTE recursivo para generar una secuencia de fechas.
2. Une la secuencia de fechas con la tabla original.

NOTA

El siguiente código se ejecuta en *MySQL*. La Tabla 9-4 tiene la sintaxis para cada RDBMS.

Paso 1: Utilizar un CTE recursivo para generar una secuencia de fechas llamada `mis_fechas`.

La tabla `mis_fechas` comienza con la fecha `2021-03-01`, y añade la siguiente fecha una y otra vez, hasta la fecha `2021-03-06`:

```
-- Sintaxis MySQL
WITH RECURSIVE mis_fechas(dt) AS (
   SELECT '2021-03-01'
   UNIÓN TODOS
   SELECT dt + INTERVALO 1 DÍA
   FROM mis_fechas
   WHERE dt < '2021-03-06')

SELECT * FROM mis_fechas;

+------------+
| dt         |
+------------+
| 2021-03-01 |
| 2021-03-02 |
| 2021-03-03 |
| 2021-03-04 |
| 2021-03-05 |
| 2021-03-06 |
+------------+
```

Paso 2: Unir por la izquierda la CTE recursiva con la tabla original.

```
-- Sintaxis MySQL
WITH RECURSIVE mis_fechas(dt) AS (
   SELECT '2021-03-01'
   UNIÓN TODOS
   SELECT dt + INTERVALO 1 DÍA
   FROM mis_fechas
   WHERE dt < '2021-03-06')
```

```
SELECT d.dt, s.price
FROM mis_fechas d
     LEFT JOIN stock_precios s
     ON d.dt = s.date;

+------------+ ----------+
| dt         | Precio
+------------+ ----------+
| 2021-03-01 | 668.27 |
| 2021-03-02 |   NULL
| 2021-03-03 | 678.83 |
| 2021-03-04 | 635.40 |
| 2021-03-05 |   NULL
| 2021-03-06 | 591.01 |
+------------+ ----------+
```

Paso 3 (Opcional): Rellene los valores nulos con el precio del día anterior.

Sustituya la cláusula `SELECT (SELECT d.dt, s.price)` por:

```
SELECT d.dt, COALESCE(s.precio,
       LAG(s.precio) OVER
       (ORDER BY d.dt)) COMO precio
...

+------------+ ----------+
| dt         | Precio
+------------+ ----------+
| 2021-03-01 | 668.27 |
| 2021-03-02 | 668.27 |
| 2021-03-03 | 678.83 |
| 2021-03-04 | 635.40 |
| 2021-03-05 | 635.40 |
| 2021-03-06 | 591.01 |
+------------+ ----------+
```

Existen diferencias sintácticas para cada RDBMS.

Esta es la sintaxis general para generar una columna de fecha. Las partes en negrita difieren según el RDBMS, y el código específico del software se indica en la Tabla 9-4.

```
[WITH] my_dates(dt) AS (
   SELECT [FECHA])
   UNIÓN TODOS
```

```
    SELECCIONE [FECHA MÁS UNO]
    FROM mis_fechas
    WHERE dt < [ÚLTIMA FECHA])

SELECT * FROM mis_fechas;
```

Tabla 9-4. Generación de una columna de fecha en cada RDBMS

RDBMS	CON	FECHA	FECHA MÁS UNO	ÚLTIMA FECHA
MySQL	CON RECURSIVE	'2021-03-01'	dt + INTERVALO 1 DÍA	'2021-03-06'
Oracle	CON	FECHA '2021-03-01'	dt + INTERVALO '1' DÍA	FECHA '2021-03-06'
PostgreSQL	CON RECURSIVE	CAST('2021-03-01' COMO FECHA)	CAST(dt + INTERVALO 1 día COMO FECHA)	'2021-03-06'
Servidor SQL	CON	CAST('2021-03-01' COMO FECHA)	DATEADD(DAY, 1, CAST(dt AS DATE))	'2021-03-06'
SQLite	CON RECURSIVE	FECHA('2021-03-01')	FECHA(dt, 1 día")	'2021-03-06'

Devuelve todos los padres de una fila hija

La siguiente tabla incluye las funciones de varios miembros de la familia . La columna de más a la derecha incluye el id del progenitor de una p e r s o n a .

```
SELECT * FROM árbol_familiar;

+------+---------+----------+--------------+
| id| name| role|            parent_id |
+------+---------+----------+--------------+
|1    | Lao Ye  |   Abuelo          |NULL |
|2 |  Lao Lao | Abuela               |NULL |
```

```
|    3 | Ollie   | Papá     |       NULL |
|    4 | Alice   | Mamá     |          1 |
|    4 | Alice   | Mamá     |          2 |
|    5 | Henry   | Hijo     |          3 |
|    5 | Henry   | Hijo     |          4 |
|    6 | Lily    | Hija     |          3 |
|    6 | Lily    | Hija     |          4 |
+------+---------+----------+--------------+
```

NOTA

El siguiente código se ejecuta en *MySQL*. La Tabla 9-5 contiene la sintaxis para cada RDBMS.

Puedes listar los padres y abuelos de cada p e r s o n a con un CTE recursivo:

```
-- Sintaxis MySQL
WITH RECURSIVE mi_cte (id, nombre, linaje) AS (
   SELECT id, nombre, nombre COMO linaje
   FROM árbol_familiar
   WHERE parent_id IS NULL
   UNIÓN TODOS
   SELECT ft.id, ft.name,
          CONCAT(mc.linaje, ' > ', ft.nombre)
   FROM árbol_familiar ft
        INNER JOIN mi_cte mc
        ON ft.parent_id = mc.id)

SELECT * FROM mi_cte ORDER BY id;
```

```
+------+---------+---------------------------+
| id| nombre| linaje                          |
+------+---------+---------------------------+
|    1 | Lao Ye  | Lao Ye                    |
|    2 | Lao Lao | Lao Lao                   |
|    3 | Ollie   | Ollie                     |
|    4 | Alice   | Lao Ye > Alice            |
|    4 | Alice   | Lao Lao > Alice           |
```

```
|    5 | Henry   | Ollie > Henry           |
|    5 | Henry   | Lao Ye > Alice > Henry  |
|    5 | Henry   | Lao Lao > Alice > Henry |
|    6 | Lily    | Ollie > Lily            |
|    6 | Lily    | Lao Ye > Alice > Lily   |
|    6 | Lily    | Lao Lao > Alice > Lily  |
+------+---------+-------------------------+
```

En el código anterior (también conocido como *consulta jerárquica*), `my_cte` contiene dos sentencias que se unen:

- La primera sentencia `SELECT` es el punto de partida. Las filas en las que `parent_id` es `NULL` se tratan como las raíces del árbol.
- La segunda sentencia `SELECT` define el vínculo recursivo entre las filas padre e hijo. Los hijos de cada raíz del árbol se devuelven y se añaden a la `columna de linaje` hasta que se obtiene el linaje completo.

Existen diferencias sintácticas para cada RDBMS.

Esta es la sintaxis general para listar todos los padres. Las partes en negrita difieren según el RDBMS, y el código específico del software se lista en la Tabla 9-5.

```
[WITH] my_cte (id, name, lineage) AS (
   SELECT id, name, [NAME] AS lineage
   FROM family_tree
   WHERE parent_id IS NULL
   UNION ALL
   SELECT ft.id, ft.name, [LINEA]
   FROM árbol_familiar ft
        INNER JOIN mi_cte mc
        ON ft.parent_id = mc.id)

SELECT * FROM mi_cte ORDER BY id;
```

Tabla 9-5. Listado de todos los padres en cada RDBMS

RDBMS	CON	NOMBRE	LÍNEA
MySQL	CON RECURSIVO	nombre	CONCAT(mc.linaje, ' > ', ft.nombre)

RDBMS	CON	NOMBRE	LÍNEA
Oracle	CON	namemc	.lineage || ' > ' || ft.name
PostgreSQL	CON RECURSIVO	CAST(nombre COMO VARCHAR(30))	CAST(CONCAT(mc.linaje, ' > ', ft.name) AS VARCHAR(30))
		SQL ServerCON CAST(nombreCOMO VARCHAR(30))	CAST(CONCAT(mc.linaje, ' > ', ft.name) AS VARCHAR(30))
SQLite	CON RECURSIVE	nombre	mc.linaje || ' > ' || ft.name

¿Cómo puedo...?

Este capítulo pretende ser una referencia rápida para las preguntas SQL más frecuentes que combinan varios conceptos:

- Encontrar las filas que contienen valores duplicados
- Seleccionar filas con el valor máximo de otra columna
- Concatenar texto de varios campos en uno solo
- Buscar todas las tablas que contengan un nombre de columna específico
- Actualizar una tabla cuyo ID coincide con el de otra tabla

Buscar las filas que contienen valores duplicados

En la siguiente tabla se enumeran siete tipos de té y las temperaturas a las que deben dejarse en infusión. Tenga en cuenta que hay dos grupos de valores de `té`/`temperatura` duplicados, que aparecen en negrita.

```
SELECT * FROM teas;

+----+--------+ --------------+
| id | té|     temperatura |
```

```
+----+--------+ --------------+
| 1 Verde               170
| 2 negro               200 |
```

```
|  3 | negro |          200 |
|  4 | herbal |         212 |
|  5 hierbas             212 |
|  6 | herbal |          210 |
|  7 | oolong |          185 |
+----+--------+ --------------+
```

Esta sección cubre dos escenarios diferentes:

- Devuelve todas las combinaciones únicas de `té` y `temperatura`
- Devuelve sólo las filas con `té`/`temperatura` duplicados valores

Devolver todas las combinaciones únicas

Para excluir los valores duplicados y devolver sólo las filas únicas de una tabla, utilice la palabra clave `DISTINCT`.

```
SELECT DISTINCT té, temperatura
FROM tés;
```

```
+--------+--------------+
         | Temperatura
+--------+--------------+
| verde             |170 |
| negro |           200 |
| hierbas           212 |
| ...a base de hierbas...       210...
| oolong            |185 |
+--------+--------------+
```

Posibles ampliaciones

Para obtener el número de filas únicas de una tabla, utilice las palabras clave `COUNT` y `DISTINCT` a la vez. Encontrará más información en la sección DISTINCT del capítulo 4.

Devolver sólo las filas con valores duplicados

La siguiente consulta identifica las filas de la tabla con valores duplicados.

```
WITH dup_rows AS (
     SELECT té, temperatura,
            COUNT(*) as num_rows
     DE tés
     GROUP BY té, temperatura
     TENIENDO COUNT(*) > 1)

SELECT t.id, d.tea, d.temperature
FROM teas t INNER JOIN dup_rows d
     ON t.té = d.té
     AND t.temperatura = d.temperatura;
```

```
+----+--------+ --------------+
| id | té|    temperatura |
+----+--------+ --------------+
| 2 | negro              |200 |
| 3 | negro              |200 |
| 4 | herbal             |212 |
| 5 | herbal             |212 |
+----+--------+ --------------+
```

Explicación

La mayor parte del trabajo se realiza en la consulta `dup_rows`. Se cuentan todas las combinaciones de `té` y `temperatura` y, a continuación, sólo se conservan las combinaciones que aparecen más de una vez con la cláusula `HAVING de` . Este es el aspecto de `dup_rows`:

```
+--------+------------+ ------------+
| té|    temperatura | num_rows |
+--------+------------+ ------------+
| negro             |200 |2 |
| herbal            |212 |2 |
+--------+------------+ ------------+
```

El propósito del `JOIN` en la segunda mitad de la consulta es volver a introducir la columna `id` en la salida final.

Palabras clave de la consulta

- **`WITH dup_rows`** es el inicio de una expresión de tabla común, que permite trabajar con varias sentencias `SELECT` dentro de una misma consulta.
- **`HAVING COUNT(*) > 1`** utiliza la cláusula `HAVING`, que permite filtrar sobre una agregación como `COUNT()`.
- **`teas t INNER JOIN dup_rows d`** utiliza un `INNER JOIN`, que permite reunir la tabla `teas` y la consulta dup_rows.

Posibles ampliaciones

Para eliminar determinadas filas duplicadas de una tabla, utilice un `DELETE`
declaración. Encontrará más información en el capítulo 5

Seleccionar filas con el valor máximo de otra columna

La siguiente tabla enumera los empleados y el número de ventas que han realizado. Desea devolver el número de ventas más reciente de cada empleado, que aparecen en negrita.

```
SELECT * FROM ventas;

+------+----------+------------+ ----------+
| id|  empleado | fecha|        ventas |
+------+----------+------------+ ----------+
|    1 | Emma     | 2021-08-01 |      6 |
|    2 | Emma     | 2021-08-02 |     17 |
|    3 | Jack     | 2021-08-02 |     14 |
|    4 | Emma     | 2021-08-04 |     20 |
|    5 | Jack     | 2021-08-05 |      5 |
|    6 | Emma     | 2021-08-07 |      1 |
+------+----------+------------+ ----------+
```

Solución

La siguiente consulta devuelve el número de ventas que cada empleado realizó en su fecha de venta más reciente (también conocida como el valor de fecha más grande de cada empleado).

```
SELECT s.id, r.employee, r.recent_date, s.sales
FROM (SELECT empleado, MAX(fecha) AS recent_date
      DE ventas
      GROUP BY empleado) r
INNER JOIN ventas s
           ON r.empleado = s.empleado
           AND r.fecha_reciente =
           s.fecha;
```

```
+------+----------+------------+ ----------+
| id|  empleado | recent_date | ventas |
+------+----------+------------+ ----------+
|5    | Jack|     2021-08-05 |5 |
|6    | Emma|     2021-08-07 |1 |
+------+----------+------------+ ----------+
```

Explicación

La clave de este problema es dividirlo en dos partes. El primer objetivo es identificar la fecha de venta más reciente de cada empleado. Este es el aspecto de la salida de la subconsulta `r`:

```
+----------+--------------+
| empleado | recent_date |
+----------+--------------+
| Emma |    2021-08-07 |
| Jack,    2021-08-05.
+----------+--------------+
```

El segundo objetivo es recuperar las columnas `id` y `sales` en la salida final, lo que se hace utilizando el `JOIN` en la segunda mitad de la consulta .

Palabras clave de la consulta

- **`GROUP BY empleado`** utiliza la cláusula `GROUP BY`, que divide la tabla por `empleado` y encuentra el **`MAX(fecha)`** para cada empleado .
- **`r INNER JOIN ventas s`** utiliza un `INNER JOIN`, que permite reunir la subconsulta `r` y la tabla `ventas`.

Posibles ampliaciones

Una alternativa a la solución `GROUP BY` es utilizar una función de ventana (`OVER ... PARTITION BY ...`) con una función FIRST_VALUE- tion, que devolvería los mismos resultados. Encontrará más detalles en la sección "Funciones de ventana" en la página 250 del capítulo 8.

Concatenar texto de varios campos en uno solo

Esta sección cubre dos escenarios diferentes:

- Concatenar el texto de los campos *de una fila* en un único valor
- Concatenar texto de campos *de varias filas* en un único valor

Concatenar texto de campos en una sola fila

La siguiente tabla tiene dos columnas y desea concatenarlas en una sola columna.

```
+----+ ---------+        +-----------+
| Nombre        |        | Nombre_id |
+----+ ---------+        +-----------+
| 1 Botas       |  --->  | 1_Botas   |
  | ... 2 ...              | 2_Pumpkin |
  Calabaza ...
| 3 | Tigre     |        | 3_Tiger   |
+----+ ---------+        +-----------+
```

Utilice la función `CONCAT` o el operador de concatenación (`||`) para reunir los valores:

```
-- MySQL, PostgreSQL y SQL Server SELECT
CONCAT(id, '_', name) AS id_name FROM
my_table;

-- Oracle, PostgreSQL y SQLite SELECT
id || '_' || nombre COMO id_nombre
FROM mi_tabla;

+-----------+
| Nombre_id |
+-----------+
| 1_Botas   |
| 2_Pumpkin |
| 3_Tiger   |
+-----------+
```

Posibles ampliaciones

El capítulo 7, *Operadores y funciones*, cubre otras formas de trabajar con valores de cadena además de `CONCAT`, incluyendo:

- Determinar la longitud de una cadena
- Encontrar palabras en una cadena
- Extraer texto de una cadena

Concatenar Texto de Campos en Varias Filas

La siguiente tabla muestra las calorías quemadas por cada persona. Quieres concatenar las calorías de cada persona en una sola fila.

```
+------+----------++------+ +
                          | nombre | calorías ||   nombre
| calorías |
+------+----------++------+ +
| ally          |80 | ---> | ally | 80,75,90 |
| ally          |75 ||   jess | 100,92     |
| ally          |90 |+------+---------------+
| jess         |100 |
| jess          |92 |
+------+ ----------+
```

Utilice una función como `GROUP_CONCAT`, `LISTAGG`, `ARRAY_AGG` o `STRING_AGG` para crear la lista.

```
SELECCIONAR nombre,
       GROUP_CONCAT(calorías) AS calories_list
FROM entrenamientos
GROUP BY nombre;
```

```
+------+ ---------------+
| nombre | calories_list |
+------+ ---------------+
| ally | 80,75,90       |
| jess | 100,92         |
+------+ ---------------+
```

Este código funciona en *MySQL* y *SQLite*. Sustituya `GROUP_CONCAT(calories)` por lo siguiente en otros RDBMS:

Oracle
```
LISTAGG(calorías, ',')
```

PostgreSQL
```
ARRAY_AGG(calorías)
```

Servidor SQL
```
STRING_AGG(calorías, ',')
```

Posibles ampliaciones

La sección Agregar filas en un único valor o lista del capítulo 8 incluye detalles sobre cómo utilizar otros separadores además de la coma (,), cómo ordenar los valores y cómo devolver valores únicos .

Buscar todas las tablas que contienen un nombre de columna específico

Imagine que tiene una base de datos con muchas tablas. Quiere encontrar rápidamente todas las tablas que contengan un nombre de columna con la palabra `ciudad` en .

Solución

En la mayoría de los SGBDR, existe una tabla especial que contiene todos los nombres de tablas y columnas. La Tabla 10-1 muestra cómo consultar esa tabla en cada RDBMS.

La última línea de cada bloque de código es opcional. Puede incluirla si desea limitar los resultados a una base de datos o un usuario concretos. Si se excluye, se devolverán todas las tablas.

Tabla 10-1. Buscar todas las tablas que contienen un nombre de columna específico

RDBMS	Código
MySQL	`SELECT nombre_tabla, nombre_columna FROM` **`esquema_informacion.columnas`** `WHERE nombre_columna LIKE` **`'%ciudad%'`** `AND esquema_tabla = 'mi_nombre_db';`
Oracle	`SELECT nombre_tabla, nombre_columna FROM` **`all_tab_columns`** `WHERE nombre_columna LIKE` **`'%CITY%'`** `AND owner = 'MI_NOMBRE_USUARIO';`
PostgreSQL, SQL Server	`SELECT nombre_tabla, nombre_columna FROM` **`esquema_informacion.columnas`**

```
W                   ERE nombre_columna LIKE '%ciudad%'
H                         AND tabla_catálogo = 'mi_nombre_db';
```

La salida mostrará todos los nombres de columna que contengan el término
`ciudad` junto con las mesas en las que se encuentran:

```
+------------+ --------------+
| NOMBRE_TABLA | NOMBRE_COLUMNA |
+------------+ --------------+
| Clientes Ciudad           |
| Empleados Ciudad          |
| Localidades Ciudades metropolitanas
+------------+ --------------+
```

NOTA

SQLite no tiene una tabla que contenga todos los nombres de las columnas. En su lugar, puede mostrar manualmente todas las tablas y luego ver los nombres de las columnas dentro de cada tabla:

```
.tablas
pragma información_tabla(mi_tabla);
```

Posibles ampliaciones

El capítulo 5, *Creación, actualización y eliminación*, cubre más formas de interactuar con bases de datos y tablas, incluyendo:

- Visualización de las bases de datos existentes
- Visualización de las tablas existentes
- Visualización de las columnas de una tabla

El capítulo 7, *Operadores y funciones*, cubre más formas de buscar texto además de `LIKE`, incluyendo:

- `=` para buscar una coincidencia exacta
- `IN` para buscar varios términos
- Expresiones regulares para buscar un patrón

Actualizar una tabla cuyo ID coincide con el de otra tabla

Imagine que tiene dos tablas: `productos` y `ofertas`. Desea actualizar los nombres de la tabla de `ofertas` con los nombres de los artículos de la tabla de `productos` que tengan un `id` coincidente.

```
SELECT * FROM productos;

+------+ --------------------+
| Nombre                     |
+------+ --------------------+
| 101 Macarrones con queso
| 102 | Teclado MIDI         |
| 103 | Tarjeta del día de la madre |
+------+ --------------------+

SELECT * FROM ofertas;

+------+ --------------+
| Nombre               |
+------+ --------------+
| 102 | Regalo técnico| --> Teclado      MIDI
| 103 | Tarjeta de vacaciones | --> Tarjeta del día de
la madre
+------+ --------------+
```

Solución

Utilice una sentencia `UPDATE` para modificar los valores de una tabla utilizando la función `ACTUALIZAR .SET. . . . .` sintaxis. La Tabla 10-2 muestra cómo hacerlo en cada RDBMS.

Tabla 10-2. Actualizar una tabla cuyo ID coincide con el de otra tabla

RDBMS	Código
MySQL	`UPDATE ofertas d, productos p`

```
SET    d.name =
p.name WHERE d.id =
p.id;
```

RDBMS	Código

Oracle	`UPDATE ofertas d`
	`SET nombre = (SELECT p.nombre`
	`FROM productos p`
	`WHERE d.id = p.id);`

PostgreSQL, SQLite	`Ofertas UPDATE`
	`SET nombre =`
	`p.nombre FROM`
	`operaciones`
	`d`
	`INNER JOIN productos`
	`p ON d.id = p.id`
	`WHERE ofertas.id = p.id;`

Servidor SQL	`UPDATE d`
	`SET d.name =`
	`p.name FROM`
	`operaciones d`
	`INNER JOIN productos`
	`p ON d.id = p.id;`

La tabla de `ofertas` se actualiza ahora con los nombres de la tabla tabla de `productos`:

```
SELECT * FROM ofertas;

+------+ -------------------+
| Nombre                    |
+------+ -------------------+
| 102 | Teclado MIDI        |
| 103 | Tarjeta del día de la madre |
+------+ -------------------+
```

ADVERTENCIA

Una vez ejecutada la sentencia UPDATE, los resultados no pueden deshacerse . La excepción es si se inicia una transacción antes de ejecutar la sentencia UPDATE.

Posibles ampliaciones

El capítulo 5, *Creación, actualización y eliminación*, cubre más formas de modificar tablas, incluyendo:

- Actualizar una columna de datos
- Actualización de filas de datos
- Actualización de filas de datos con los resultados de una consulta
- Añadir una columna a una tabla

Palabras finales

Este libro cubre los conceptos y palabras clave más populares de SQL, pero sólo hemos arañado la superficie. SQL se puede utilizar para realizar muchas tareas, utilizando una variedad de enfoques diferentes. Te animo a seguir aprendiendo y explorando.

Te habrás dado cuenta de que la sintaxis SQL varía mucho según RDBMS. Escribir código SQL requiere mucha práctica, paciencia y buscar la sintaxis. Espero que esta guía de bolsillo de te haya resultado útil para ello.

Índice

Símbolos

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Sobre el autor

Alice Zhao es una científica de datos a la que le apasiona enseñar y hacer que las cosas complejas sean fáciles de entender. Ha impartido numerosos cursos de SQL, Python y R como científica de datos sénior en Metis y como cofundadora de Best Fit Analytics. Sus tutoriales técnicos en YouTube son conocidos por ser prácticos, entretenidos y visualmente atractivos.

Escribe sobre análisis y cultura pop en su blog, *A Dash of Data*. Su trabajo ha aparecido en *Huffington Post*, *Thrill- ist* y *Working Mother*. Ha sido ponente en diversas conferencias, como Strata en Nueva York y ODSC en San Francisco, sobre temas que van desde el procesamiento del lenguaje natural a la visualización de datos. Tiene un máster en análisis y una licenciatura en ingeniería eléctrica, ambos por la Northwestern University.

Colofón

El animal que aparece en la portada de la *Guía de Bolsillo SQL es* un *salamandra alpino* (*salamandra atra*). Habitual de los barrancos de los Alpes (por encima de los 1.000 m), la salamandra alpina destaca por su inusual capacidad para soportar el frío. Estas criaturas negras y brillantes prefieren los lugares sombríos y húmedos y las grietas y huecos de los muros de piedra. Se alimenta de gusanos, arañas, caracoles y pequeñas larvas de insectos.

A diferencia de otras salamandras, la salamandra alpina da a luz a juveniles completamente formados. La gestación dura dos años, pero a mayor altitud (1.400-1.700 m) puede durar hasta tres. En general, la especie está protegida en todos los Alpes, pero el cambio climático ha afectado más recientemente a su hábitat preferido de paisajes rocosos y no demasiado secos.

Muchos de los animales de la portada de O'Reilly están en peligro de extinción; todos ellos son importantes para el mundo.

La ilustración de la portada es obra de Karen Montgomery, basada en un grabado en blanco y negro de la *Royal Natural History de Lydekker*. Los tipos de letra de la portada son Gilroy Semibold y Guardian Sans. La fuente del texto es Adobe Minion Pro; la del encabezamiento, Adobe Myriad Condensed; y la del código, Dalton Maag's Ubuntu Mono.