SQL

Autor: Tomas Caporalettiti

En este archivo encontrarás información útil para manejar SQL.

Creacion de tablas

Para crear tablas en SQL, se utiliza la sentencia CREATE TABLE. A continuación, se describen algunos de los conceptos clave y opciones que puedes utilizar al crear una tabla:

Estructura básica de CREATE TABLE

```
CREATE TABLE <Nombre_de_la_tabla> (
    <Columna1> <Tipo_de_dato> [OPCIONES],
    <Columna2> <Tipo_de_dato> [OPCIONES],
    <Columna3> <Tipo_de_dato> [OPCIONES]
);
```

Definición de columnas

Cada columna en una tabla debe tener un nombre y un tipo de dato asociado. A continuación, algunos de los tipos de datos más comunes:

- INT: Almacena números enteros.
- VARCHAR(n): Almacena cadenas de texto de longitud variable, donde n especifica la longitud máxima.
- TEXT: Almacena texto largo.
- DATE: Almacena fechas en el formato YYYY-MM-DD.
- FLOAT: Almacena números con punto decimal.

Opciones de las columnas

Al definir las columnas, puedes utilizar diferentes opciones para especificar restricciones o comportamientos especiales:

- PRIMARY KEY: Define una columna (o un conjunto de columnas) que identificará de manera única cada fila en la tabla. Solo puede haber una clave primaria por tabla.

```
id INT PRIMARY KEY
```

- AUTO_INCREMENT: Se utiliza con columnas numéricas para que su valor se incremente automáticamente cada vez que se inserta una nueva fila. Es común utilizarlo junto con PRIMARY KEY para crear identificadores únicos.

```
id INT PRIMARY KEY AUTO_INCREMENT
```

- NOT NULL: Impone que la columna no puede contener valores nulos. Esto asegura que siempre haya un valor en esa columna para cada fila.

```
nombre VARCHAR(100) NOT NULL
```

- FOREIGN KEY: Define una clave foránea, que es una columna o un conjunto de columnas que hacen referencia a una clave primaria en otra tabla. Esto se utiliza para establecer relaciones entre tablas.

```
autor_id INT,
FOREIGN KEY (autor_id) REFERENCES Autores(id)
```

Ejemplo completo de CREATE TABLE

```
CREATE TABLE Libros (
    id INT PRIMARY KEY AUTO_INCREMENT,
    titulo VARCHAR(255) NOT NULL,
    autor_id INT,
    fecha_publicacion DATE,
    FOREIGN KEY (autor_id) REFERENCES Autores(id)
);
```

Clave primaria compuesta

Cuando una tabla requiere que más de una columna se utilice para formar una clave primaria, puedes definir una clave primaria compuesta. Esto asegura que la combinación de valores en esas

columnas sea única para cada fila.

Ejemplo de clave primaria compuesta: autor-libro

Imagina que tienes una tabla que registra los autores de libros. Es posible que un autor escriba varios libros, y que un libro tenga varios autores. Para asegurarte de que la combinación de autor y libro sea única en la tabla, defines ambas columnas como una clave primaria compuesta.

```
CREATE TABLE AutorLibro (
    autor_id INT,
    libro_id INT,
    PRIMARY KEY (autor_id, libro_id),
    FOREIGN KEY (autor_id) REFERENCES Autores(id),
    FOREIGN KEY (libro_id) REFERENCES Libros(id)
);
```

- Repetición permitida: Un mismo autor_id puede aparecer múltiples veces en la tabla si está asociado con diferentes libro_id. De la misma manera, un libro_id puede aparecer varias veces si está asociado con diferentes autor_id.
- Unicidad de la combinación: Lo que no está permitido es que la misma combinación de autor_id y libro_id se repita, garantizando que cada autor y libro aparezcan juntos solo una vez en la tabla.

Eliminacion de tablas

Eliminar una tabla es una tarea muy sencilla y se realiza mediante la sentencia DROP TABLE. Esta operación eliminará por completo la tabla especificada, junto con todos los datos que contiene.

```
DROP TABLE <Nombre_tabla>;
```

Insertar campos en tablas

Para insertar datos en una tabla en SQL, se utiliza la sentencia INSERT INTO. Dependiendo de tus necesidades, puedes optar por insertar todos los valores de un registro o solo ciertos datos.

1. Insertar un registro completo
   Cuando conoces el orden y el número de columnas en la tabla y deseas insertar un registro completo sin especificar nombres de columna, puedes utilizar la siguiente sintaxis:

```
INSERT INTO <Nombre_tabla> VALUES (Valor1, Valor2, Valor3, ..., ValorN);
```

2. Insertar ciertos datos de un registro
   Si solo deseas insertar datos en ciertas columnas o si el orden de las columnas no es conocido, puedes especificar los nombres de las columnas en la sentencia INSERT INTO. Esto también te permite omitir columnas que tienen valores predeterminados o que son auto-incrementales.

```
INSERT INTO <Nombre_tabla> (<Columna1>, ..., <ColumnaN>) VALUES (Valor1, ..., Valor
```

- Integridad referencial: Si la tabla tiene claves foráneas, asegúrate de que los valores insertados en las columnas correspondientes correspondan a valores existentes en las tablas relacionadas.

Consultas simples

Para consultar datos en una tabla, utilizamos la sentencia SELECT, que es extremadamente versátil y puede adaptarse a diversas necesidades.

1. Seleccionar todos los datos de una tabla
   Para recuperar todos los datos de una tabla, puedes usar el asterisco (*), que actúa como un comodín para todas las columnas:

```
SELECT *
FROM <Nombre_tabla>;
```

2. Seleccionar columnas específicas
   Si solo deseas ver datos de columnas específicas, puedes enumerar las columnas que te interesan:

```
SELECT <Columna1>, <Columna2>, ..., <ColumnaN>
FROM <Nombre_tabla>;
```

3. Renombrar columnas en el resultado
   Puedes usar la cláusula AS para renombrar columnas en el resultado, lo que puede mejorar la legibilidad. Aunque AS no es obligatorio, es una buena práctica:

```
SELECT
    <Columna1> AS <"Nuevo_nombre1">,
    <Columna2> AS <"Nuevo_nombre2">,
    ...,
```

```
    <ColumnaN> AS <"Nuevo_nombreN">
FROM <Nombre_tabla>;
```

4. Seleccionar valores únicos
   Para obtener valores únicos y eliminar duplicados en el resultado, utiliza DISTINCT:

```
SELECT DISTINCT <Columna1>
FROM <Nombre_tabla>;
```

5. Limitar la cantidad de registros
   Para limitar la cantidad de registros devueltos por una consulta, usa LIMIT. También puedes usar OFFSET para especificar desde qué registro empezar:

```
SELECT *
FROM <Nombre_tabla>
LIMIT <Numero_de_registros>
OFFSET <A_partir_de_que_registro>;
```

WHERE

La cláusula WHERE es extremadamente útil para filtrar registros de una tabla en función de condiciones específicas. A continuación, se describen varios ejemplos de cómo usar WHERE en SQL.

1. Sintaxis básica

```
SELECT *
FROM <Nombre_tabla>
WHERE <Columna> [Operador_lógico] <Filtro>;
```

Operadores lógicos permitidos:

- == o = : Igual a
- != o <> : Distinto de
- > : Mayor que
- < : Menor que
- >= : Mayor igual que
- <= : Menor igual que

El filtro puede ser un valor de texto, número entero, fecha, u otros tipos de datos.

2. Uso de operadores lógicos
   Puedes combinar varias condiciones usando los operadores lógicos AND y OR:

```
SELECT *
FROM <Nombre_tabla>
WHERE ((Condicion1) AND (Condicion2)) OR (Condicion3);
```

3. Filtrar entre un rango
   Puedes usar BETWEEN para filtrar valores que caen dentro de un rango específico:

```
SELECT *
FROM <Nombre_tabla>
WHERE <Columna1> BETWEEN <Inicio> AND <Fin>;
```

4. Coincidencia con múltiples valores
   La cláusula IN se utiliza para filtrar registros que coincidan con uno o más valores específicos:

```
SELECT *
FROM <Nombre_tabla>
WHERE <Columna1> IN (<Valor1>, <Valor2>, ..., <ValorN>);
```

5. Filtrar registros con campos vacíos
   Puedes usar IS NULL para filtrar registros donde un campo específico no tiene ningún valor (es NULL):

```
SELECT *
FROM <Nombre_tabla>
WHERE <Columna1> IS NULL;
```

6. Negación de condiciones
   Puedes invertir las condiciones usando NOT, lo que permite filtrar registros que no coincidan con los criterios especificados:

```
WHERE <Columna1> IS NOT NULL;
```

```
WHERE <Columna1> NOT BETWEEN <Inicio> AND <Fin>;
```

```
WHERE <Columna1> NOT IN (<Valor1>, <Valor2>, ..., <ValorN>);
```

7. Uso de comodines
   Los comodines son herramientas poderosas para buscar patrones específicos en los datos, especialmente cuando se combinan con la cláusula LIKE.
8. El comodín %
   El comodín % representa cero, uno o varios caracteres, y se utiliza para buscar patrones dentro de cadenas de texto.

```
SELECT * FROM <Nombre_tabla> WHERE <Columna1> LIKE '%<Caracter>%';
```

Ejemplos:

- Buscar cualquier valor que contenga la letra "s" en cualquier posición:

```
SELECT * FROM tabla WHERE titulo LIKE '%s%';
```

- Buscar cualquier valor que termine con "ing":

```
SELECT * FROM tabla WHERE titulo LIKE '%ing';
```

- Buscar cualquier valor que empiece con "Bu%"

```
SELECT * FROM tabla WHERE titulo LIKE 'Bu%';
```

2. El comodín _
   El comodín _ representa un único carácter. Es útil cuando sabes cuántos caracteres deben coincidir, pero no sabes cuáles.
   Ejemplos:

- Buscar títulos que comiencen con "s" seguido de dos caracteres específicos:

```
SELECT * FROM tabla WHERE titulo LIKE 's__';
```

- Buscar títulos donde la cuarta letra sea "d":

```
SELECT * FROM tabla WHERE titulo LIKE '___d%';
```

3. El comodín [^]
   El comodín [^] representa cualquier carácter que no esté dentro de los corchetes. Es útil para excluir ciertos caracteres en una posición específica.
   Ejemplo:

- Buscar títulos que no comiencen con "A" o "B":

```
SELECT *
FROM Libros
WHERE titulo LIKE '[^AB]%';
```

4. El comodín []
   El comodín [] representa cualquier carácter que esté dentro de los corchetes. Es útil para buscar uno de varios caracteres posibles en una posición específica.
   Ejemplo:

- Buscar títulos que comiencen con "A" o "B":

```
SELECT *
FROM Libros
WHERE titulo LIKE '[AB]%';
```

5. Sensibilidad a mayúsculas y minúsculas (Case sensitivity)
   En algunas bases de datos, las búsquedas con LIKE son sensibles a mayúsculas y minúsculas (case-sensitive), lo que significa que "Libro" y "libro" se consideran diferentes. En otras bases de datos, LIKE no distingue entre mayúsculas y minúsculas. Puedes utilizar las funciones LOWER() o UPPER() para normalizar la búsqueda y evitar problemas de sensibilidad.

```
SELECT * FROM <Nomre_tabla> WHERE LOWER(<Columna>) LIKE '%<Texto>%';
```

Operaciones Básicas

En SQL, existen varias funciones agregadas que permiten realizar operaciones básicas como contar, sumar, promediar, y obtener el valor máximo o mínimo de una columna. A continuación, se describen estas funciones y su uso.

1. COUNT
   La función COUNT se utiliza para contar el número de registros en una tabla. Puedes contar todos los registros o solo aquellos que cumplen con ciertas condiciones.

```
SELECT COUNT(*)
FROM <Tabla>;

SELECT COUNT (DISTINCT <Columna>)
FROM <Tabla>;
```

2. SUM
   La función SUM suma los valores de una columna numérica.

```
SELECT SUM (<Columna>) AS <Nuevo_nombre>
FROM <Tabla>;
```

3. AVG
   La función AVG calcula el promedio de los valores de una columna numérica.

```
SELECT AVG (<Columna>) AS <Nuevo_nombre>
FROM <Tabla>;
```

4. MAX y MIN
   Las funciones MAX y MIN se utilizan para obtener el valor máximo y mínimo de una columna, respectivamente.

```
SELECT MAX (<Columna>)
FROM <Tabla>;
```

```
SELECT MIN (<Columna>)
FROM <Tabla>;
```

5. ORDER BY
   La cláusula ORDER BY se utiliza para ordenar los resultados de una consulta por una o más columnas. Puedes ordenar los resultados de manera ascendente (ASC) o descendente (DESC). Por defecto, el orden es ascendente.

```
SELECT <Columna1>, <Columna2>, ..., <ColumnaN>
FROM <Tabla>
ORDER BY <Columna1> [ASC | DESC];
```

GROUP BY

El uso de GROUP BY en SQL es fundamental cuando necesitas agrupar filas que tienen los mismos valores en columnas específicas y luego aplicar funciones de agregación como COUNT(), SUM(), AVG(), MAX(), o MIN() a cada grupo.

1. Agrupación de Datos
   La cláusula GROUP BY permite agrupar filas que comparten un valor común en una o más columnas. Una vez agrupadas, puedes aplicar funciones de agregación para realizar cálculos sobre cada grupo.
2. Aplicación en Consultas
   GROUP BY se coloca después de las tablas en la cláusula FROM y antes de HAVING o ORDER BY.

Ejemplos de Uso

- Contar cuántos libros tiene cada autor:
  Aquí, id_autor es la columna por la cual agrupamos, y para cada autor, se cuenta el número de libros asociados

```
SELECT id_autor, COUNT(*) AS cantidad_de_libros
FROM autor_libro
GROUP BY id_autor;
```

- Calcular el total de ventas para cada libro:
  Aquí, id_libro es la columna por la cual agrupamos, y para cada libro, se calcula la suma de cantidad * precio.

```
SELECT id_libro, SUM(cantidad * precio) AS total_ventas
FROM detalle_Factura
GROUP BY id_libro;
```

- Agrupar datos por departamento y puesto, calculando el salario promedio para cada combinación:

```
SELECT departamento, puesto, AVG(salario) AS salario_promedio
FROM empleados
GROUP BY departamento, puesto;
```

3. Filtrar Grupos con HAVING
   La cláusula HAVING se utiliza para filtrar los resultados de GROUP BY en función de una condición establecida sobre los grupos. Es similar a WHERE, pero se usa con datos agrupados.

- Filtrar grupos con más de 5 pedidos:
  Aquí, HAVING COUNT(*) > 5 filtra los grupos de id_cliente que tienen más de 5 pedidos, después de que se han contado los pedidos en cada grupo

```
SELECT id_cliente, COUNT(*) AS num_pedidos
FROM pedidos
GROUP BY id_cliente
HAVING COUNT(*) > 5;
```

INNER JOIN

El INNER JOIN se utiliza para combinar filas de dos o más tablas en función de una condición relacionada, generalmente una clave foránea (Foreign Key) que corresponde a la clave primaria (Primary Key) de otra tabla. Solo devuelve las filas que tienen coincidencias en ambas tablas.

```
SELECT
    T1.<Columna1>,
    T2.<Columna2>,
    T1.<Columna2>
FROM <Tabla1> AS T1
INNER JOIN <Tabla2> AS T2 ON T1.<[Primmary Key]> = T2.<[Foreign Key]>
```

Ejemplo:

- Supongamos que tienes dos tablas: Estudiantes y Inscripciones. Cada estudiante tiene un id_estudiante, y la tabla de Inscripciones tiene una clave foránea id_estudiante que se relaciona con la clave primaria de la tabla Estudiantes.

```
SELECT
    Estudiantes.nombre,
    Estudiantes.apellido,
    Inscripciones.fecha_inscripcion
FROM Estudiantes
INNER JOIN Inscripciones ON Estudiantes.id_estudiante = Inscripciones.id_estudiante
```

Este INNER JOIN devuelve los nombres, apellidos y fechas de inscripción de los estudiantes que tienen registros en ambas tablas.

Consideraciones adicionales

- Filtrado Adicional: Puedes combinar INNER JOIN con cláusulas WHERE para agregar más condiciones a la consulta, como filtrar por fechas o valores específicos.

LEFT JOIN

El LEFT JOIN (o LEFT OUTER JOIN) devuelve todas las filas de la tabla de la izquierda (Tabla1) y las filas coincidentes de la tabla de la derecha (Tabla2). Si no hay coincidencias, las columnas de la tabla de la derecha tendrán valores NULL.

```
SELECT
    T1.<Columna1>,
    T2.<Columna2>,
    T1.<Columna3>
FROM <Tabla1> AS T1
LEFT JOIN <Tabla2> AS T2 ON T1.<Primary_Key> = T2.<Foreign_Key>;
```

Ejemplo

Supongamos que queremos obtener una lista de todos los estudiantes, independientemente de si tienen una inscripción o no.

```
SELECT
    Estudiantes.nombre,
    Estudiantes.apellido,
    Inscripciones.fecha_inscripcion
FROM Estudiantes
LEFT JOIN Inscripciones ON Estudiantes.id_estudiante = Inscripciones.id_estudiante;
```

Este LEFT JOIN devolverá todos los nombres y apellidos de los estudiantes, y la fecha de inscripción para aquellos que están inscritos. Para los estudiantes que no están inscritos, la columna fecha_inscripcion mostrará NULL.

UNION

La cláusula UNION se utiliza para combinar los resultados de dos o más consultas SELECT. UNION elimina automáticamente los duplicados, pero si deseas incluir duplicados, puedes usar UNION ALL.

```
SELECT <Columna1>, <Columna2>
FROM <Tabla1>
UNION
SELECT <Columna1>, <Columna2>
FROM <Tabla2>;
```

Subconsultas o consultas anidadas

Las subconsultas, también conocidas como consultas anidadas, son consultas dentro de otra consulta. Se utilizan para realizar operaciones más complejas que no se pueden lograr fácilmente con una sola consulta simple.

Sintaxis Básica

```
SELECT <Columna1>, <Columna2>, ..., <ColumnaN>
FROM <Tabla1>
WHERE <Columna1> [Operador logico] (SELECT AVG(<Columna1>) FROM <Tabla2>);
```

Ejemplo Práctico

Supongamos que tienes dos tablas: Ventas y Productos. La tabla Ventas tiene una columna precio, y deseas encontrar todos los productos cuyo precio esté por encima del precio promedio de todos los productos.

```
SELECT nombre_producto, precio
FROM Productos
WHERE precio > (SELECT AVG(precio) FROM Productos);
```

Funciones utiles

1. Funciones de fecha
   1.1 STRFTIME
   Formatea una fecha u hora (en tipo texto) a un formato específico.

```
SELECT STRFTIME("%[Modificador de formato]", <Columna_fechas_tipo_texto>
FROM <Tabla>;
```

Modificadores de formato:

- %d: Día del mes, con dos dígitos (01-31).
- %m: Mes del año, con dos dígitos (01-12).
- %Y: Año con cuatro dígitos (por ejemplo, 2024).
- %H: Hora en formato de 24 horas (00-23).
- %M: Minutos (00-59).
- %S: Segundos (00-59).
- %w: Día de la semana (0 = domingo, 1 = lunes, …, 6 = sábado).
- %W: Semana del año (00-53).
- %j: Día del año (001-366).
- %B: Nombre completo del mes (por ejemplo, "August").
- %A: Nombre completo del día de la semana (por ejemplo, "Monday").
- %p: AM o PM.

Ejemplo de combinación:

```
SELECT strftime('%I:%M %p', '2024-08-16 15:30:00');
```

Resultado: '03:30 PM'

1.2 NOW()
Devuelve la fecha y hora actual.

```
SELECT NOW();
```

1.3 DATE()
Extrae la parte de la fecha de una fecha y hora. Además, convierte una fecha en texto a tipo fechafecha.

```sql
SELECT DATE('2024-08-16 10:30:00');
```

1.4 DATEADD()
Suma un intervalo a una fecha.

```sql
SELECT DATEADD(day, 5, '2024-08-16');
```

1.5 DATEDIFF()
Calcula la diferencia entre dos fechas.

```sql
SELECT DATEDIFF('2024-08-16', '2024-08-01');
```

2. Funciones de Control de Flujo
   Estas funciones son útiles para realizar decisiones lógicas dentro de las consultas SQL.

2.1 CASE
Funciona como una declaración IF-THEN-ELSE.

```sql
SELECT nombre,
       CASE
           WHEN salario > 50000 THEN 'Alto'
           ELSE 'Bajo'
       END AS nivel_salarial
FROM empleados;
```

2.2 IFNULL()
Devuelve un valor alternativo si el valor original es NULL.

```sql
SELECT nombre, IFNULL(comision, 0) FROM empleados;
```

3. Funciones de Cadena de Texto
   Estas funciones son útiles para manipular y formatear cadenas de texto.

3.1 CONCAT()
Combina dos o más cadenas en una sola.

```sql
SELECT CONCAT(nombre, ' ', apellido) AS nombre_completo FROM empleados;
```

3.2 SUBSTRING()
Extrae una parte específica de una cadena.

```sql
SELECT SUBSTRING(nombre, 1, 3) FROM empleados;
```

3.3 LENGTH()
Devuelve la longitud de una cadena.

```sql
SELECT LENGTH(nombre) FROM empleados;
```

4. Funciones de Manipulación de Números

4.1 ROUND()
Redondea un número a un número específico de decimales.

```sql
SELECT ROUND(salario, 2) FROM empleados;
```

4.2 ABS()
Devuelve el valor absoluto de un número.

```sql
SELECT ABS(-10);
```

4.3 CEIL()
Redondea un número hacia arriba al entero más cercano.

```sql
SELECT CEIL(4.3);
```

4.4 FLOOR()
Redondea un número hacia abajo al entero más cercano.

```sql
SELECT FLOOR(4.7);
```

4.5 POWER()
Eleva un número a la potencia especificada.

```
SELECT POWER(2, 3);
```

5. Funciones de Manipulación de Datos
   Estas funciones permiten convertir tipos de datos.

5.1 CAST()
Convierte un valor de un tipo de datos a otro.

```
SELECT CAST(salario AS DECIMAL(10, 2)) FROM empleados;
```

5.2 CONVERT()
Similar a CAST(), convierte un valor de un tipo a otro.

```
SELECT CONVERT(VARCHAR(10), fecha) FROM empleados;
```