Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

Handbook de SQL

Introducción a

SQL o *Structured Query Language*, es el lenguaje que usamos para trabajar con bases de datos. Es como un puente entre tú y los datos: te permite consultar, modificar o administrar la información almacenada. Aunque hay un estándar (ANSI SQL), cada sistema de bases de datos, como MySQL ,SQL Server u Oracle, tiene sus propios trucos y extensiones, así que no todo lo que aprendas funcionará igual en todos lados.

Componentes de

SQL se basa en comandos que nos ayudan a trabajar con las bases de datos. Estos comandos se dividen en dos tipos principales:

1. **DDL (Data Definition Language)**:
    - **CREATE**: Crea tablas, bases de datos o estructuras nuevas.
    - **DROP**: Borra tablas o estructuras.
    - **ALTER**: Modifica las tablas agregando o cambiando columnas.
2. **DML (Data Manipulation Language)**:
    - **SELECT**: Extrae datos que cumplan con un criterio.
    - **INSERT**: Agrega datos nuevos a las tablas.
    - **UPDATE**: Cambia valores específicos de las tablas.
    - **DELETE**: Elimina registros de una tabla.

Cláusulas importantes

Al trabajar con datos, es clave usar estas cláusulas para afinar los resultados de tus consultas:

- **FROM**: Especifica la tabla de donde se obtendrán los datos.
- **WHERE**: Define condiciones para filtrar los registros.
- **ORDER BY**: Ordena los resultados (ascendente o descendente).
- **GROUP BY**: Agrupa los datos para cálculos más avanzados.
- **HAVING**: Filtra grupos de datos después de agruparlos.

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

Operadores Lógicos

Los operadores lógicos permiten construir condiciones más complejas en tus consultas. Estos son los principales:

- **AND**: Devuelve registros que cumplen todas las condiciones.
  - Ejemplo: SELECT * FROM empleados WHERE salario > 3000 AND departamento = 'Ventas';
- **OR**: Devuelve registros que cumplen al menos una condición.
  - Ejemplo: SELECT * FROM clientes WHERE ciudad = 'Madrid' OR ciudad = 'Barcelona';
- **NOT**: Excluye registros que cumplen la condición.
  - Ejemplo: SELECT * FROM productos WHERE NOT categoria = 'Electrónica';

También puedes combinar varios operadores para condiciones más detalladas:

```
SELECT * FROM empleados
WHERE (salario > 2000 AND edad < 50) OR departamento = 'TI';
```

Operadores de comparación

Estos se utilizan para comparar valores en las condiciones:

- =: Igual a
- <>: Diferente de
- >: Mayor que
- <: Menor que
- >=: Mayor o igual
- <=: Menor o igual
- BETWEEN: Dentro de un rango. Ejemplo: salario BETWEEN 2000 AND 5000
- LIKE: Busca patrones. Ejemplo: nombre LIKE 'J%' (nombres que empiezan con "J").
- IN: Busca valores dentro de una lista. Ejemplo: ciudad IN ('Madrid', 'Barcelona')

Funciones agregadas

Las funciones agregadas son herramientas útiles para hacer cálculos o análisis sobre grupos de datos. Algunas de las más comunes son:

- **AVG**: Calcula el promedio.
  - Ejemplo: SELECT AVG(salario) FROM empleados;
- **SUM**: Suma los valores de una columna.

  - Ejemplo: SELECT SUM(precio) FROM ventas;
- **COUNT**: Cuenta la cantidad de registros.
  - Ejemplo: SELECT COUNT(*) FROM clientes;
- **MAX** y **MIN**: Encuentran el valor máximo y mínimo, respectivamente.
  - Ejemplo: SELECT MAX(edad), MIN(edad) FROM personas;

Joins (Uniones)

Los *joins* combinan datos de dos o más tablas basándose en una relación común. Aquí están los principales:

1. **INNER JOIN**: Devuelve los registros que tienen coincidencias en ambas tablas.
   - Ejemplo: SELECT * FROM empleados INNER JOIN departamentos ON empleados.id_dep = departamentos.id;
2. **LEFT JOIN**: Muestra todos los registros de la tabla izquierda y las coincidencias de la derecha (si existen).
   - Ejemplo: SELECT * FROM empleados LEFT JOIN proyectos ON empleados.id = proyectos.id_emp;
3. **RIGHT JOIN**: Lo opuesto a LEFT JOIN; prioriza la tabla derecha.
4. **FULL JOIN**: Combina todos los registros, coincidan o no.

Subconsultas

Las subconsultas son consultas dentro de otras consultas. Son útiles para resolver problemas más complejos.

- **Ejemplo básico**:
  Encuentra los empleados cuyo salario es mayor al promedio:

```
SELECT nombre FROM empleados WHERE salario > (SELECT AVG(salario) FROM empleados);
```

- **Uso con IN**: Busca valores que coincidan con una lista generada por otra consulta:

```
SELECT producto FROM inventario WHERE id IN (SELECT id FROM ventas WHERE cantidad > 100);
```

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

Creación y manejo de bases de datos

también permite crear, modificar y eliminar bases de datos y tablas. Aquí tienes los comandos básicos:

Crear una base de datos

Para iniciar, puedes crear una nueva base de datos con este comando:

```
CREATE DATABASE mi_base_de_datos;
```

Crear tablas

Parte	Descripción
tabla	Es el nombre de la tabla que se va a crear.
campo1 campo2	Es el nombre del campo o de los campos que se van a crear en la nueva tabla. La nueva tabla debe contener, al menos, un campo.
tipo	Es el tipo de datos de campo en la nueva tabla. (Ver Tipos de Datos)
tamaño	Es el tamaño del campo sólo se aplica para campos de tipo texto.
índice1 índice2	Es una cláusula `CONSTRAINT` que define el tipo de indice a crear. Esta cláusula en opcional.
índice multicampos	Es una cláusula `CONSTRAINT` que define el tipo de indice multicampos a crear. Un índice multi campo es aquel que está indexado por el contenido de varios campos. Esta cláusula en opcional.

Define una tabla con sus columnas y tipos de datos:

```
CREATE TABLE empleados (
  id INT PRIMARY KEY,
  nombre VARCHAR(50),
  salario DECIMAL(10, 2)
);
```

Modificar tablas

Si necesitas cambiar algo en una tabla existente:

- **Agregar columnas**:

```
ALTER TABLE empleados ADD fecha_ingreso DATE;
```

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

- **Cambiar columnas**:

```
ALTER TABLE empleados MODIFY salario DECIMAL(12, 2);
```

Eliminar tablas o bases de datos

Para borrar una tabla o toda la base de datos:

- **Eliminar una tabla**:

```
DROP TABLE empleados;
```

- **Eliminar una base de datos**:

```
DROP DATABASE mi_base_de_datos;
```

Cláusula CONSTRAINT

La cláusula **CONSTRAINT** se utiliza para definir reglas o restricciones en las tablas. Estas restricciones aseguran la integridad de los datos en la base de datos. Aquí están las más comunes:

Tipos de restricciones

1. **PRIMARY KEY**
   Define una clave primaria que identifica de forma única cada registro.

```
CREATE TABLE empleados (
  id INT CONSTRAINT pk_empleados PRIMARY KEY,
  nombre VARCHAR(50)
);
```

2. **FOREIGN KEY**
   Relaciona una tabla con otra, asegurando que los valores de una columna correspondan a los de otra tabla.

```
CREATE TABLE pedidos (
  id INT PRIMARY KEY,
  id_cliente INT CONSTRAINT fk_cliente FOREIGN KEY
REFERENCES clientes(id)
);
```

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

3. **UNIQUE**
   Asegura que los valores de una columna sean únicos.

```
CREATE TABLE usuarios (
  email VARCHAR(100) CONSTRAINT unq_email UNIQUE
);
```

4. **CHECK**
   Valida que los datos cumplan ciertas condiciones.

```
CREATE TABLE productos (
  precio DECIMAL(10, 2) CONSTRAINT chk_precio CHECK (precio > 0)
);
```

5. **NOT NULL**
   Obliga a que una columna siempre tenga un valor (no puede ser NULL).

```
CREATE TABLE clientes (
  nombre VARCHAR(50) NOT NULL
);
```

Agregar o eliminar restricciones con ALTER TABLE

Puedes añadir o eliminar restricciones después de crear la tabla:

- **Agregar una restricción**:

```
ALTER TABLE empleados ADD CONSTRAINT fk_departamento
FOREIGN KEY (id_dep) REFERENCES departamentos(id);
```

- **Eliminar una restricción**:

```
ALTER TABLE empleados DROP CONSTRAINT fk_departamento;
```

Consultas de Selección

Las consultas de selección en SQL se utilizan para obtener datos de una o más tablas. La base de todo es el comando **SELECT**, que permite extraer columnas, filas o datos específicos según tus necesidades.

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

Sintaxis básica

SELECT columnas FROM tabla;

- Ejemplo:

  SELECT nombre, salario FROM empleados;

  Esto devuelve las columnas nombre y salario de la tabla empleados.

Filtrar resultados con WHERE

Para mostrar datos que cumplen ciertas condiciones:

SELECT * FROM productos WHERE precio > 100;

- **Ejemplo con varias condiciones**:

  SELECT * FROM clientes WHERE ciudad = 'Madrid' AND edad > 30;

Ordenar resultados con ORDER BY

Puedes organizar los resultados de forma ascendente (por defecto) o descendente:

SELECT nombre, salario FROM empleados ORDER BY salario DESC;

Agrupar datos con GROUP BY

Ideal para usar con funciones agregadas y agrupar registros:

SELECT departamento, AVG(salario) FROM empleados GROUP BY departamento;

Filtrar grupos con HAVING

A diferencia de **WHERE**, **HAVING** se aplica a los grupos creados con **GROUP BY**:

```
SELECT departamento, COUNT(*) AS total_empleados
FROM empleados
GROUP BY departamento
HAVING COUNT(*) > 5;
```

Mostrar registros únicos con DISTINCT

Elimina filas duplicadas en los resultados:

Gabriel Armando Márquez Gómez
Actuaria 7mo semestre

```
SELECT DISTINCT ciudad FROM clientes;
```

Selección de todos los campos

Si quieres obtener todas las columnas, usa el asterisco *:

```
SELECT * FROM empleados;
```

Usar alias para renombrar columnas o tablas

- Para columnas:

```
SELECT nombre AS empleado, salario AS sueldo FROM empleados;
```

- Para tablas (útil en joins):

```
SELECT e.nombre, d.nombre AS departamento
FROM empleados e
INNER JOIN departamentos d ON e.id_dep = d.id;
```

¿Qué es el operador LIKE?

- Permite buscar valores en una columna que coincidan con un patrón específico.
- Se utiliza comúnmente con la cláusula **WHERE** para filtrar resultados según condiciones específicas.

Sintaxis básica:

```
SELECT columnas
FROM tabla
WHERE columna LIKE 'patrón';
```

Caracteres comodines en LIKE

Para definir patrones, se utilizan dos comodines principales:

1. **%**: Representa **cero, uno o más caracteres**.
    - Ejemplo: LIKE 'A%' encuentra cualquier texto que comience con "A".
    - Ejemplo: LIKE '%z' encuentra cualquier texto que termine con "z".
    - Ejemplo: LIKE '%abc%' encuentra cualquier texto que contenga "abc" en cualquier posición.
2. **_**: Representa **exactamente un carácter**.
    - Ejemplo: LIKE 'A_' encuentra cualquier texto de dos caracteres que comience con "A".

- Ejemplo: LIKE 'C_t' encuentra "Cat", "Cut", pero no "Cart".

Uso avanzado con rangos y exclusiones

En algunos motores de bases de datos (como SQL Server), puedes usar corchetes [ ] para definir rangos o listas específicas de caracteres:

- **Rango de caracteres:**
  LIKE '[A-D]%' busca valores que comiencen con letras entre "A" y "D".
- **Lista específica:**
  LIKE '[AEIOU]%' busca valores que comiencen con una vocal.
- **Exclusiones:**
  LIKE '[!A-E]%' busca valores que no comiencen con las letras "A" a "E".

Ejemplos prácticos

Buscar palabras que comiencen con una letra específica

```
SELECT * FROM empleados WHERE nombre LIKE 'J%';
```

Buscar palabras que terminen con ciertos caracteres

```
SELECT * FROM productos WHERE descripcion LIKE '%mento';
```

Buscar valores con una estructura fija

```
SELECT * FROM cuentas WHERE numero LIKE '123_';
```

Buscar valores dentro de un rango de caracteres

```
SELECT * FROM clientes WHERE ciudad LIKE '[A-C]%';
```