Módulo 5: Lenguaje SQL

SQL (Structured Query Language) es un lenguaje diseñado para gestionar y manipular bases de datos relacionales. Se utiliza para crear, modificar y consultar datos en tablas de bases de datos. SQL es el idioma estándar para comunicarse con bases de datos relacionales, permitiendo tanto la gestión de la estructura como la manipulación de los datos que contienen. Un programador necesita conocer SQL porque casi todas las aplicaciones modernas interactúan con bases de datos para almacenar, gestionar y recuperar información.

Estudiaremos este lenguaje como herramienta independiente en la programación destinada al uso de distintos sistemas de gestión de bases de datos.

El módulo se compone de tres partes que abordaremos en gradualmente, la primera de introducción a SQL con las consultas usando “SELECT” e “INSERT” simples, la segunda con el uso de “UPDATE” y “DELETE” y finalmente la tercera parte cubre consultas combinando diferentes tablas usando “JOIN”, “UNION” y operaciones de conjuntos sobre múltiples tablas

PARTE 1:

Introducción a SQL

El paradigma de SQL
SQL se basa en el paradigma declarativo, que es fundamentalmente diferente de los lenguajes de programación tradicionales. En SQL no se indica “cómo” hacer algo paso a paso de la forma en que estamos acostumbrados en los algoritmos de programación, sino que se describe “qué” se necesita obtener, por ello se dice que es declarativo.

Conceptos Fundamentales del Paradigma SQL
El lenguaje SQL toma como base conceptos que ya hemos abordado que repasamos aquí:
Modelo Relacional

- Los datos se organizan en tablas (relaciones)
- Cada tabla tiene filas (tuplas) y columnas (atributos)
- Las relaciones entre tablas se establecen mediante claves

Operaciones basadas en Conjuntos

- SQL trabaja con conjuntos de datos completos, no con registros individuales, por ejemplo, cuando hacemos:
  `UPDATE productos SET precio = precio * 1.1 WHERE categoria = 'electronica';`

  Esta operación afecta a todos los productos de electrónica simultáneamente.

Álgebra Relacional
SQL implementa operaciones matemáticas sobre conjuntos:

- Selección (WHERE): filtra filas
- Proyección (SELECT): elige columnas
- Unión (UNION): combina resultados
- Intersección (INTERSECT): elementos comunes

- Diferencia (EXCEPT): elementos no comunes

Naturaleza Descriptiva
Describes el resultado que buscas, no el algoritmo para obtenerlo. El motor de base de datos decide la estrategia óptima de ejecución. Es importante comprender que tiene grandes diferencias con la programación imperativa, que debemos de pensar de forma diferente según estas características distintivas

- Sin orden inherente: Las tablas son conjuntos, no listas ordenadas
- Idempotencia: La misma consulta siempre produce el mismo resultado
- Composición: Puedes anidar consultas y combinar operaciones
- Optimización automática: El motor decide cómo ejecutar tu consulta de forma eficiente

SQL está compuesto de sub-lenguajes destinados a diferentes operaciones en la gestión de las bases de datos. A saber: DDL es lenguaje de definición de datos (Data Definition Language) se encarga de la estructura de la base de datos, DML es el lenguaje de manipulación de datos (Data Manipulation Language) se encarga de manipular los datos dentro de las estructuras ya creadas. Es como el "operario" que trabaja con el contenido de las tablas que el DDL ya construyó y es el que principalmente abordaremos en este módulo, finalmente, DCL es el Lenguaje de Control de Datos (Data Control Language) destinado a permisos y seguridad.

¿Por qué tantos sub lenguajes?
Esta división tiene sentido porque de esta forma se independizan y diferencian:

- Responsabilidades: Los administradores de BD usan más DDL, los desarrolladores más DML
- Permisos: Puedes dar acceso a DML sin permitir DDL
- Momentos de tiempo: DDL se usa en diseño/mantenimiento, DML en operación diaria
- Impactos: DDL afecta estructura (más riesgoso), DML afecta contenido.

La nomenclatura con "Language" refleja que cada uno tiene su propio conjunto de reglas, sintaxis y propósito específico dentro del ecosistema SQL.
Estudiaremos DDL y DCL detalladamente en los módulos siguientes, pero hagamos aquí una introducción para cubrir lo elemental:

DDL - Data Definition Language (Lenguaje de Definición de Datos)
Se encarga de la estructura de la base de datos, es decir de la definición de las tablas, los datos y las bases de datos mismas. Requiere permisos diferenciados, dado que su impacto es a nivel del modelo y no de los datos. Las operaciones en DDL son frecuentes en momentos de instalación de los sistemas informáticos y poco frecuentes en su operación y uso habitual, aunque la modificación de funciones y procedimientos almacenados puede estar vinculada también al mantenimiento de los sistemas.
Tareas propias de DDL son CREATE (crear), ALTER (modificar), DROP (eliminar), TRUNCATE (vaciar).
Veamos algunos ejemplos:

Crear tabla

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

Modificar estructura

```
ALTER TABLE empleados ADD COLUMN telefono VARCHAR(20);
```

Eliminar tabla

```
DROP TABLE empleados;
```

Veremos con mayor profundidad este el lenguaje en el módulo siguiente, pero cabe mencionar que todo lo referido relacionado con la creación de esquemas , tablas y columnas, índices, relaciones entre tablas, vistas y otras relacionadas están abarcadas en DDL.

PARTE 2:

Introducción a DML

Veamos detalladamente el sub-lenguaje de manipulación de datos DML (Data Manipulation Language) Como mencionamos anteriormente se encarga de manipular los datos dentro de las estructuras ya creadas. Es como el "operario" que trabaja con el contenido de las tablas que el DDL ya construyó.

Conceptos Fundamentales del DML

¿Qué hace el DML?
Consultar datos (SELECT): Es el uso más frecuente. Permite extraer información específica de una o varias tablas, usando filtros y condiciones. Por ejemplo, puedes pedir que te muestre el nombre y apellido de todos los clientes que viven en una ciudad determinada.

Insertar datos (INSERT): Se utiliza para agregar nuevos registros (filas) a una tabla. Por ejemplo, cuando un nuevo usuario se registra en una página web, su información se inserta en una tabla de usuarios.

Actualizar datos (UPDATE): Permite modificar los datos de registros existentes en una tabla. Por ejemplo, si un usuario cambia su dirección, se usa UPDATE para actualizar el campo de dirección en su registro.

Borrar datos (DELETE): Se utiliza para eliminar registros de una tabla. Por ejemplo, si un usuario cancela su cuenta, se puede borrar su registro de la base de datos.

- Consulta datos existentes (SELECT)
- Insertar nuevos datos (INSERT)
- Actualiza datos existentes (UPDATE)
- Elimina datos específicos (DELETE)

Desarrollaremos en esta primera etapa las instrucciones de selección e inserción de datos.

SELECT
SELECT es el comando m usado y uno de los más complejos en SQL, tiene un gran potencial y para dominarlo debemos ir desde los básico gradualmente a lo más complejo

Veamos un SELECT básico, una consulta simple tiene la forma:

```
SELECT nombre, precio FROM productos;
```

Para obtener todas las columnas de la tabla sin necesidad de detallar las usamos **"*"**

```
SELECT * FROM productos;
```

Filtros

Las operaciones con SELECT generalmente trabajan sobre un sub-conjunto de valores o parte de una tabla que cumple cortas condiciones, y muchas veces sobre una única fila, de allí la importancia de definir condiciones apropiadas dentro de los SELECT, para ello generalmente usamos WHERE

```
SELECT nombre, precio FROM productos
WHERE precio > 1000;
```

Y agrupamos condiciones lógicas con los operadores AND , OR, BETWEEN o IN

```
SELECT * FROM productos WHERE precio > 500 AND stock < 10;
```

```
SELECT * FROM productos WHERE categoria_id = 1 OR categoria_id = 3;
```

Podemos filtrar ciertos rangos de valores en una columna:

```
SELECT * FROM productos WHERE precio BETWEEN 100 AND 1000;
```

Podemos filtrar ciertos valores de una columna indicando una lista de ellos:

```
SELECT * FROM productos WHERE categoria_id IN (1, 2, 3);
```

Para el tratamiento de las cadenas tiene sus particularidades donde podemos usar = o LIKE

```
SELECT * FROM productos WHERE código = 'RZ21000-E';
```

En el caso de LIKE existe un comodín "%" que podemos ubicar en diferentes posiciones

La posición del % determina el tipo de coincidencia que se busca.

- A) % al final ('patron%')

Busca valores que comiencen con un patrón específico. Esto es útil para encontrar todos los registros que inician con un conjunto de caracteres.

```
SELECT * FROM clientes WHERE nombre LIKE 'Mar%';
```

Esta consulta devolverá registros con nombres como María, Mario, Marta, etc.

- B) % al principio ('%patron')

Busca valores que terminan con un patrón específico. Se utiliza para encontrar registros que finalizan con un determinado conjunto de caracteres.

```
SELECT * FROM productos WHERE nombre LIKE '%computadora';
```

Esta consulta devolverá registros como Laptop computadora, Computadora de escritorio, Computadora personal.

- C) % al principio y al final ('%patron%')

Busca valores que contienen el patrón en cualquier parte de la cadena.
Es la forma más flexible y se usa para encontrar registros donde un conjunto de caracteres se encuentra en cualquier posición.

```
SELECT * FROM articulos WHERE descripcion LIKE '%oferta%';
```

Esta consulta devolvería artículos que tienen la palabra oferta en su descripción, sin importar si está al principio, en medio o al final.

- D) Sin % (coincidencia exacta) equivalente a "="

Aunque no se use el comodín, LIKE también puede usarse para una coincidencia exacta, aunque el operador "=" es el más común para este fin.

```
SELECT * FROM usuarios WHERE nombre LIKE 'Juan';
```

Esta consulta es equivalente a usar WHERE nombre = 'Juan'.

Ordenamiento

En las consultas suele ser muy importante el orden en que se presentan los datos, más allá de los casos simples y especificar si el orden es ascendente o descendente, es posible aplicar un límite a la cantidad de filas presentadas, también utilizar funciones y operaciones sobre los datos y ordenar de acuerdo con esos resultados. Veamos algunos ejemplos

Especificamos que el ordenamiento sea por el precio y descendente

```
SELECT nombre, precio FROM productos  ORDER BY precio DESC;
```

Especificamos que el ordenamiento sea por categoría ascendente y luego el precio descendente

```
SELECT nombre, precio FROM productos
ORDER BY categoria_id ASC, precio DESC;
```

Establecer un límite a la cantidad de resultados a mostrar

```
SELECT * FROM productos ORDER BY precio DESC LIMIT 10;
```

Esta consulta devuelve los 10 productos más caros

Consejos y Buenas Prácticas en cuanto al ordenamiento:

- Siempre especifica ASC o DESC para claridad
- Usa nombres de columna, no posiciones numéricas
- Considera el rendimiento en tablas grandes
- Índices compuestos para ordenamiento múltiple

SELECT con Subconsultas

Aquí tenemos una pequeña introducción a las sub-consultas, pero recomendamos ver el documento especifico del módulo para ello.

Como introducir una sub-consulta simple en un filtro:

```
SELECT nombre, precio FROM productos WHERE precio > (SELECT AVG(precio) FROM productos);
```

Como introducir una sub-consulta en una columna de la consulta actual

```
SELECT    nombre,    precio,    (SELECT AVG(precio) FROM productos) AS precio_promedio,
   precio - (SELECT AVG(precio) FROM productos) AS diferenciaFROM productos;
```

Agrupación

En muchas oportunidades vamos a necesitar datos agrupados la clausula ORDER BY es la encargada de realizar esta tarea en una consulta.

¿Qué hace GROUP BY?

GROUP BY agrupa filas que tienen valores idénticos en las columnas especificadas y las convierte en una sola fila por grupo, permitiendo aplicar funciones de agregación.

Estas funciones de agregación pueden ser promedio (AVG), suma (SUM), máximo (MAX), etc.

La siguiente consulta muestra una fila por departamento e indica el promedio de salario

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

La sintaxis básica de la cláusula implica la existencia de una función de agregación y el detalle de las columnas luego del GROUP BY, debes indicar todas las columnas no agrupadas del SELECT en la cláusula GROUP BY y el orden definitivo será establecido en ORDER BY.

```
SELECT columnas_agrupacion, funciones_agregacion
FROM tabla
WHERE condiciones        -- Filtro ANTES del agrupamiento
GROUP BY columnas_agrupacion
HAVING condiciones_grupo   -- Filtro DESPUÉS del agrupamiento
ORDER BY columnas;
```

Funciones de Agregación Comunes

COUNT(*) para contar filas
COUNT(nombreColumna) Cuenta los no nulos de la columna
COUNT(DISTINCT nombreColumna), Cuenta los valores únicos
SUM(salario) as masa_salarial, -- Suma total
AVG(salario) as salario_promedio, -- Promedio
MIN(fecha_ingreso) as mas_antiguo, -- Mínimo
MAX(salario) as salario_maximo, -- Máximo

GROUP_CONCAT(nombre) as lista_nombres -- MySQL: concatenar

Orden de Agrupamiento: GROUP BY NO garantiza orden el orden de las columnas mostradas se lo indicamos con ORDER BY

PARTE 2:

Operaciones de inserción

La cláusula INSERT se utiliza para agregar datos a una tabla
Un ejemplo de INSERT básico podría ser :

```
INSERT INTO productos (nombre, precio, stock, categoria_id)
VALUES ('Laptop Gaming', 1500.00, 5, 1);
```

Y una inserción de varios registros podría expresarse de esta forma

```
INSERT INTO productos (nombre, precio, stock, categoria_id)
VALUES
  ('Mouse Gamer', 50.00, 20, 2),
  ('Teclado Mecánico', 120.00, 15, 2),
  ('Monitor 4K', 400.00, 8, 3);
```

INSERT con Subconsultas, si los datos dependen de otra tabla podemos expresar asi :

```
INSERT INTO productos_descontinuados (nombre, precio, fecha_descontinuado)
SELECT nombre, precio, CURRENT_DATE
FROM productos
WHERE stock = 0 AND fecha_creacion < '2020-01-01';
INSERT con Valores por Defecto
```

También es posible insertar una fila sin especificar todos sus valores, en tal caso se tomarán los valores por defecto.

```
INSERT INTO productos (nombre, precio) -- stock tomará valor DEFAULT
VALUES ('Producto Nuevo', 99.99);
```

O Insertar registro con todos los valores por defecto

```
INSERT INTO productos () VALUES ();
```

Atención: Debemos tener en cuenta que la cláusula INSERT es mucho más compleja de lo que parece, el software de la base de datos debe realizar una serie de operaciones adicionales a agregar el dato a la tabla, debemos considerar que si existen claves foráneas es necesario validarlas, si existen índices también deben actualizarse, etc.
Por ejemplo, MYSQL antes de insertar verifica:

- Permisos del usuario
- Sintaxis de la consulta
- Existencia de la tabla y columnas
- Tipos de datos compatibles

Restricciones NOT NULL
Valores DEFAULT

Luego inicia una transacción (si no hay una activa) y para ello:

Obtiene locks necesarios
Reserva espacio en el buffer pool
Prepara el rollback en caso de error
Si todo sale bien confirma la transacción, sino realiza el rollback

Y luego continua con otras operaciones, que no profundizaremos, esta lista es solo a modo de llamado de atención, para que consideremos la carga de trabajo y que como programadores debemos ser conscientemente responsables de estas actividades. Profundizaremos esto en el módulo siguiente cuando estudiemos los sistemas de gestión de bases de datos, para comprender completamente el impacto.

PARTE 3:

Conclusiones

Un programador necesita conocer SQL porque las aplicaciones interactúan con bases de datos para almacenar, gestionar y recuperar información. SQL es el lenguaje estándar para comunicarse con estas bases de datos relacionales.

Razones clave

- Manipulación de datos: La mayoría de las funcionalidades de un sitio web dependen de datos. Un programador web debe saber cómo insertar nuevos usuarios, actualizar perfiles, eliminar contenido obsoleto, y consultar información para mostrarla en la página. Sin SQL, no sería posible manejar esta información.
- Fundamento del desarrollo back-end: El back-end (o "la parte de atrás" de un sitio web) es el que se encarga de la lógica y la interacción con el servidor y la base de datos. Un desarrollador back-end, en particular, utiliza SQL a diario para construir las API que conectan la interfaz del usuario (el front-end) con la base de datos.
- Optimización del rendimiento: Con el tiempo, las bases de datos crecen y las consultas se vuelven más lentas. Saber SQL permite a los programadores optimizar consultas para que la carga de la página sea más rápida, mejorando la experiencia del usuario y la eficiencia del servidor.
- Comprensión de los ORM: Muchos frameworks web utilizan ORM (Object-Relational Mappers), que son herramientas que permiten interactuar con la base de datos usando código del lenguaje de programación (como Python o JavaScript) en lugar de SQL puro. Sin embargo, para usar un ORM de manera efectiva, un desarrollador debe entender los conceptos subyacentes de las bases de datos y cómo las consultas SQL funcionan, especialmente para solucionar problemas y optimizar el rendimiento.
- Seguridad: El conocimiento de SQL es crucial para prevenir vulnerabilidades de seguridad como la inyección SQL, que es un ataque común donde los piratas informáticos insertan código malicioso en un formulario web para acceder a la base de datos o destruirla.