UNIDAD 2 - Sentencias y Sublenguajes

El Lenguaje SQL

Introducción al Lenguaje SQL

El Lenguaje de Consultas Estructuradas

SQL, o Lenguaje de Consultas Estructuradas (*Structured Query Language*), es un lenguaje de programación diseñado para gestionar y manipular bases de datos relacionales. A lo largo de los años, SQL se ha establecido como el estándar de facto en la manipulación de bases de datos debido a su potente capacidad para realizar consultas complejas de manera eficiente.

Tiene comandos para realizar las operaciones de gestión de nuestros datos (DML), como de gestión de objetos (DDL), gestión de usuarios y permisos (DCL) y gestión de transacciones (TCL). En el caso del DML una de los comandos mas utilizados es el de `SELECT` que se utiliza para consultar los datos alojados. `INSERT` para insertar, `UPDATE` para actualizar y `DELETE` para eliminar.

Características Clave de SQL

- **Intuitivo y fácil de aprender**: La sintaxis de SQL sigue un formato casi natural en inglés, lo que facilita su aprendizaje y uso. Por ejemplo, para obtener información de una base de datos, simplemente usamos comandos como `SELECT`, `FROM`, y `WHERE`.
- **Manipulación de datos eficaz**: SQL permite realizar operaciones de inserción, actualización, eliminación y consulta en datos almacenados de manera estructurada en bases de datos.
- **Interactivo y versátil**: SQL interactúa con el sistema de gestión de bases de datos para realizar tareas como crear y modificar esquemas de bases de datos, insertar y actualizar datos, además de gestionar usuarios y permisos.

Sintaxis Básica de Consultas SQL

La sintaxis de SQL es bastante directa. Aquí presentamos un ejemplo básico de cómo se estructura una consulta SQL típica:

markdownCopiar código

SELECT columna1, columna2 FROM nombre_tabla WHERE condicion;

- **SELECT**: Define qué columnas de datos deseamos recuperar.
- **FROM**: Especifica de qué tabla se deben extraer los datos.
- **WHERE**: Filtra registros bajo condiciones específicas.

Ejemplos de Consultas SQL

1. **Consulta Simple**:

   markdownCopiar código

   SELECT nombre, apellido FROM empleados WHERE departamento = 'Ventas';

   Este comando selecciona los nombres y apellidos de los empleados del departamento de ventas.

2. **Inserción de Datos**:

   markdownCopiar código

   INSERT INTO clientes (nombre, direccion) VALUES ('Ana Pérez', 'Calle Falsa 123');

   Añade un nuevo registro a la tabla de clientes.

3. **Actualización de Datos**:

   markdownCopiar código

   UPDATE productos SET precio = precio * 1.10 WHERE categoria = 'Electrónica';

   Aumenta el precio de todos los productos de electrónica en un 10%.

4. **Eliminación de Datos**:

   markdownCopiar código

   DELETE FROM pedidos WHERE fecha < '2021-01-01';

   Elimina registros de pedidos anteriores al año 2021.

Conclusión

SQL es un lenguaje esencial para cualquier profesional que trabaje con datos. Su simplicidad y poder hacen que sea indispensable para la gestión efectiva de bases de datos. A través de este curso, profundizaremos en cada uno de estos aspectos, asegurando que los estudiantes no solo comprendan cómo usar SQL, sino también cómo aplicarlo efectivamente para resolver problemas reales de datos.

Operadores en SQL

Operadores de Comparación en SQL

Los operadores de comparación en SQL son fundamentales para realizar consultas que requieren filtrado de datos basado en condiciones específicas. Estos operadores permiten comparar valores en las columnas de las tablas y son esenciales para manipular y gestionar datos de manera efectiva.

Principales Operadores de Comparación

1. **Igual (`=`)**: Compara si dos valores son iguales.
2. **Distinto (`!=` o `<>`)**: Verifica si dos valores son diferentes.
3. **Menor que (`<`)**: Comprueba si un valor es menor que otro.
4. **Mayor que (`>`)**: Evalúa si un valor es mayor que otro.
5. **Menor o igual que (`<=`)**: Determina si un valor es menor o igual a otro.
6. **Mayor o igual que (`>=`)**: Determina si un valor es mayor o igual a otro.

Además, SQL proporciona operadores para situaciones más específicas:

- **BETWEEN**: Verifica si un valor está dentro de un rango.
- **LIKE**: Se utiliza para buscar un patrón específico en una columna.
- **IN**: Permite especificar múltiples valores posibles para una columna.

Uso de Operadores de Comparación en Consultas SQL

Los operadores de comparación son más comúnmente usados junto con la cláusula `WHERE`, que permite especificar las condiciones bajo las cuales una consulta será ejecutada. Aquí algunos ejemplos de cómo se utilizan en consultas reales:

1. **Selección Simple**:

   markdownCopiar código

   SELECT * FROM empleados WHERE salario >= 3000;

   Esta consulta selecciona todos los registros de empleados donde el salario es mayor o igual a 3000.

2. **Uso de BETWEEN**:

   markdownCopiar código

   SELECT * FROM productos WHERE precio BETWEEN 10 AND 50;

   Recupera todos los productos cuyo precio está entre 10 y 50.

3. **Combinando LIKE e IN**:

   markdownCopiar código

   SELECT * FROM clientes WHERE nombre LIKE 'A%' AND ciudad IN ('Madrid', 'Barcelona');

   Esta consulta selecciona todos los clientes cuyos nombres comienzan con la letra 'A' y están ubicados en Madrid o Barcelona.

Consideraciones Especiales

- **NULL**: Especial atención debe darse al comparar `NULL`, un valor especial que representa la ausencia de un dato. Operadores como `=` o `<>` no funcionan con `NULL`. En su lugar, se utiliza `IS NULL` o `IS NOT NULL`.
- **Case Sensitivity**: La sensibilidad a mayúsculas y minúsculas en las comparaciones puede variar según la configuración del servidor de bases de datos.

Conclusión

El entendimiento profundo de los operadores de comparación en SQL es crucial para diseñar consultas efectivas y precisas, permitiendo a los desarrolladores y analistas de datos filtrar y gestionar grandes volúmenes de datos eficientemente. Estos operadores son la base para realizar análisis de datos complejos y para la toma de decisiones informadas basadas en datos precisos y bien filtrados.

Sublenguajes SQL

Sentencias Complementarias en SQL

El manejo efectivo de datos en una base de datos relacional es crucial para mantener la integridad, precisión y accesibilidad de la información. SQL, con su rica suite de comandos, permite a los usuarios manipular datos de forma eficiente. A continuación, exploraremos las sentencias fundamentales de SQL que permiten modificar, insertar y eliminar datos.

Modificar Datos con `UPDATE`

La sentencia `UPDATE` se utiliza para modificar los datos existentes en una tabla. Es fundamental especificar una condición de filtrado con `WHERE` para evitar cambios no deseados en otros registros.

Ejemplo de uso: UPDATE clientes SET nombre = 'Ana María', ciudad = 'Sevilla' WHERE id_cliente = 101;

Este comando actualiza el nombre y la ciudad del cliente cuyo `id_cliente` es 101.

Insertar Datos con `INSERT INTO`

`INSERT INTO` permite añadir nuevos registros a la tabla. Se debe especificar claramente en qué columnas se insertarán los datos y luego proporcionar los valores correspondientes.

Ejemplo de uso: INSERT INTO clientes (id_cliente, nombre, ciudad) VALUES (102, 'Juan Martín', 'Madrid');

Este comando inserta un nuevo registro en la tabla `clientes` con los datos proporcionados.

Eliminar Datos con `DELETE`

La sentencia `DELETE` se utiliza para eliminar registros de una tabla. Al igual que con `UPDATE`, es crucial usar la cláusula `WHERE` para limitar los registros afectados.

Ejemplo de uso: DELETE FROM clientes WHERE ciudad = 'Barcelona';

Este comando eliminará todos los registros de clientes que estén ubicados en Barcelona.

Recomendaciones con `UPDATE` y `DELETE`

Tanto para el actualizar datos como para eliminar, lo recomendable en términos de buenas prácticas es realizar, en la medida de ser posible, estas operaciones utilizando las PK en las tablas, por cuestiones de seguridad para no eliminar (o modificar) datos que no queremos que se elminen, como tambien por performance.

Tambien es recomendable realizar una consulta con SELECT utilizando esos registros a modificar (o eliminar) y de esa forma estar seguros de lo que vamos a realizar, por ejemplo

Antes de realizar esta operación:

UPDATE clientes SET nombre = 'Ana María', ciudad = 'Sevilla' **WHERE id_cliente = 101**;

Es recomendable realizar esta consulta:

SELECT * FROM clientes **WHERE id_cliente = 101**;

Antes de realizar esta operación:

DELETE FROM clientes **WHERE ciudad = 'Barcelona';**

Es recomendable realizar esta consulta:

SELECT * FROM clientes **WHERE ciudad = 'Barcelona';**

Consideraciones de Seguridad y Mantenimiento

1. **Backup Regular**: Antes de realizar operaciones de actualización o eliminación, es recomendable hacer copias de seguridad de los datos para evitar pérdidas accidentales.
2. **Transacciones**: Para asegurar la integridad de los datos durante las actualizaciones complejas, se pueden utilizar transacciones. Esto permite revertir todas las operaciones en caso de error.
3. **Limitar Accesos**: Es importante limitar los permisos de usuario en la base de datos para prevenir modificaciones no autorizadas.

Uso de Cláusulas Complementarias

- **`RETURNING`**: Algunos sistemas de gestión de bases de datos permiten usar `RETURNING` con `INSERT`, `UPDATE`, y `DELETE` para obtener detalles sobre los registros afectados.
- **Control de Condiciones**: El uso adecuado de `WHERE` en `UPDATE` y `DELETE` es fundamental para evitar afectar a registros no deseados.

Ejemplo de Transacción:

BEGIN TRANSACTION; UPDATE cuentas SET saldo = saldo - 100 WHERE id_cliente = 101; UPDATE cuentas SET saldo = saldo + 100 WHERE id_cliente = 102; COMMIT;

Este bloque asegura que ambas operaciones, la deducción y la adición de saldo, se realicen correctamente. Si algo falla, se puede revertir todo el proceso con `ROLLBACK`.

Conclusión

El dominio de estas sentencias de SQL no solo mejora la eficiencia en la gestión de datos sino que también fortalece la seguridad y la integridad de las bases de datos. A través de ejemplos y prácticas continuas, los usuarios pueden adquirir habilidades avanzadas en la manipulación de datos, preparándolos para enfrentar desafíos más complejos en el mundo del desarrollo y análisis de bases de datos.

Práctica y Aplicación de Consultas

Desarrollando habilidades con SQL

1. Utilizar Sintaxis Correcta y Estructurada

Una consulta SQL bien estructurada no solo facilita la comprensión sino también el mantenimiento del código. Para ello, sigue estas prácticas:

- **Indentación:** Divide cada cláusula en líneas separadas para mejorar la legibilidad.
- **Nombres Descriptivos:** Usa nombres de tablas y columnas que reflejen su propósito.
- **Orden Correcto:** Las cláusulas deben seguir esta secuencia:
    - `SELECT` – Qué datos recuperar.
    - `FROM` – De dónde recuperar los datos.
    - `WHERE` – Filtrar los datos.
    - `GROUP BY` – Agrupar los datos.
    - `HAVING` – Filtrar los datos agrupados.
    - `ORDER BY` – Ordenar los resultados.
- **Finalizar con `;`:** Esto asegura que las consultas sean interpretadas correctamente, especialmente en scripts largos.

Ejercicio Práctico: Reescribe esta consulta siguiendo las mejores prácticas de indentación:

SELECT name, age FROM users WHERE age > 30 ORDER BY name;

2. Selección Específica de Campos

Usar `SELECT *` es tentador, pero ineficiente. Seleccionar solo las columnas necesarias reduce la transferencia de datos y mejora el rendimiento. Además, al trabajar con aplicaciones, recibir columnas no requeridas puede causar conflictos en el código.

Ejemplo:

Ineficiente:
SELECT * FROM employees;

-

Eficiente:
SELECT first_name, last_name, department FROM employees;

-

Ejercicio Práctico: Convierte la siguiente consulta para que seleccione únicamente las columnas `title` y `price`:

SELECT * FROM products;

3. Uso de Índices

Los índices aceleran las consultas al permitir búsquedas más rápidas en columnas específicas. Sin embargo, no abuses de ellos, ya que pueden ralentizar las operaciones de escritura (INSERT, UPDATE, DELETE).

Ejemplo: Crea un índice en la columna `email`:

CREATE INDEX idx_email ON users(email);

Ejercicio Práctico: Identifica qué columna sería adecuada para un índice en esta consulta y crea el índice:

SELECT order_date FROM orders WHERE customer_id = 123;

4. Evitar Consultas Anidadas Innecesarias

Reformula las subconsultas como `JOINs` siempre que sea posible. Esto mejora la legibilidad y el rendimiento.

Ejemplo:

Subconsulta:
SELECT name FROM customers WHERE id IN (SELECT customer_id FROM orders);

-

Mejor con `JOIN`:
SELECT customers.name
FROM customers
INNER JOIN orders ON customers.id = orders.customer_id;

-

Ejercicio Práctico: Reformula esta consulta usando un `JOIN`:

SELECT name FROM students WHERE id IN (SELECT student_id FROM enrollments WHERE course_id = 1);

5. Uso Apropiado de `JOIN`

Cada tipo de `JOIN` tiene un propósito específico. Usar el adecuado asegura resultados correctos y eficientes.

Ejemplo:

INNER JOIN: Solo devuelve filas que coinciden en ambas tablas.
```
 SELECT orders.id, customers.name
FROM orders
INNER JOIN customers ON orders.customer_id = customers.id;
```

-

LEFT JOIN: Incluye todas las filas de la tabla izquierda, incluso si no hay coincidencias.
```
 SELECT orders.id, customers.name
FROM orders
LEFT JOIN customers ON orders.customer_id = customers.id;
```

-

Ejercicio Práctico: Escribe una consulta que devuelva todos los productos y, si están asociados, también sus categorías.

6. Utilizar Operadores Eficientes

Operadores como `EXISTS` y comparaciones directas (`>`, `<`) son más eficientes que otras alternativas.

Ejemplo:

Ineficiente:
```
 SELECT * FROM orders WHERE customer_id IN (SELECT id FROM customers);
```

-

Eficiente:
```
 SELECT * FROM orders WHERE EXISTS (SELECT 1 FROM customers WHERE customers.id = orders.customer_id);
```

-

Ejercicio Práctico: Reformula esta consulta para usar `EXISTS`:

```
SELECT * FROM products WHERE id IN (SELECT product_id FROM inventory WHERE quantity > 0);
```

7. Limitar los Resultados de la Consulta

Usa `LIMIT` o `TOP` para obtener solo las filas necesarias. Esto es útil en grandes bases de datos o para previsualizar datos.

Ejemplo:

SELECT * FROM sales ORDER BY sale_date DESC LIMIT 10;

Ejercicio Práctico: Escribe una consulta que devuelva los 5 clientes más recientes.

8. Evitar Funciones en `WHERE`

Aplicar funciones en columnas puede desactivar los índices. En su lugar, usa rangos directos.

Ejemplo:

Ineficiente:
SELECT * FROM orders WHERE YEAR(order_date) = 2023;

-

Eficiente:
SELECT * FROM orders WHERE order_date BETWEEN '2023-01-01' AND '2023-12-31';

-

Ejercicio Práctico: Optimiza esta consulta evitando funciones en el `WHERE`:

SELECT * FROM employees WHERE UPPER(name) = 'JOHN';

9. Optimización de `GROUP BY` y `HAVING`

Usa `WHERE` para filtrar datos antes de agruparlos con `GROUP BY`. Aplica filtros finales con `HAVING`.

Ejemplo:

SELECT department, COUNT(*) AS total_employees
FROM employees

```
WHERE hire_date >= '2020-01-01'
GROUP BY department
HAVING COUNT(*) > 10;
```

Ejercicio Práctico: Escribe una consulta que agrupe ventas por producto y devuelva solo los productos con más de 100 ventas.

10. Pruebas y Ajustes

Monitorea el plan de ejecución SQL (`EXPLAIN` o `EXPLAIN PLAN`) para identificar cuellos de botella y optimizar las consultas.

Ejemplo:

```
EXPLAIN SELECT * FROM orders WHERE customer_id = 123;
```

Ejercicio Práctico: Usa `EXPLAIN` para analizar la consulta:

```
SELECT * FROM products WHERE price > 100;
```

Material complementario

- 7 razones para aprender SQL | CampusMPV.es
- 5 Bases de datos para la empresa | Francisco Palazón
- Practicar SQL | w3schools