

Capítulo 8: Consultas de bases de datos

Hemos llegado al capítulo donde comenzamos a manipular bases de datos propiamente dichas.

Ahora vamos a crear nuestra primer base de datos, y mediante un lenguaje de consultas denominado SQL, vamos a crear bases de datos, dentro de ellas tablas con sus atributos y dentro de ella, insertaremos datos. Todo esto lo haremos mediante SQL usando consultas DDL y DML. Existen consultas de tipo DCL, pero no lo veremos en este espacio.

SQL, es un lenguaje universal, podemos usar cualquier DBMS Relacional, y las consultas serán las mismas, independientemente de si usamos Microsoft SQL Server, Oracle, IBM DB2, MySQL, SQLite, PostgreSQL, etc.


Cada uno tiene sus particularidades ya que con el afán de diferenciarse y tratar de ser mejor que la competencia implementan algunas características que no tienen otros DBMS, pero en la manipulación desde el lado del programador, es prácticamente igual.

Para llevar a cabo el cursado de esta asignatura, recomiendo usar MySQL, porque es un motor gratis y robusto, o bien SQLite, que nos permite ejecutar de manera más ligera las consultas. No es tan potente este último, pero para proyectos pequeños es más que suficiente. No es lo mismo hacer una base de datos para un médico que quiere registrar sus pacientes, historias clínicas, y turnos (sirve cualquier DBMS), que hacer una base de datos para gestionar todos los vuelos y pasajeros que pasan por un aeropuerto. Este último va a almacenar una cantidad de información mucho mayor, que implica usar un DBMS más potente para evitar la ralentización de las búsquedas por ejemplo.


Dicho esto, pasemos a ver SQL. No vamos a ahondar en la instalación del DBMS, porque está más claro en el material audiovisual, pero a continuación comparto algunos videos que pueden ser de utilidad:

MySQL: Requiere instalación. Podemos instalar MySQL Workbench que incluye el motor de base de datos propiamente dicho y a su vez Workbench que es un

programa que permite manipular de forma visual la base de datos. También se puede instalar XAMPP, pero nos centraremos solo en workbench:

 [Cómo INSTALAR MySQL WORKBENCH en WINDOWS | Por Completo y ...](#)

SQLite: Es un motor liviano, muy fácil de utilizar y posee un software para explorar esta base de datos muy simple, poco potente, pero funcional para lo que necesitamos nosotros:

 [Instalación y Configuración de SQLite en Windows 10 para trabajar desde CM...](#)

SQL

SQL (Structured Query Language) es un lenguaje utilizado para administrar y manipular bases de datos relacionales. Proporciona un conjunto de comandos y sentencias que permiten crear, modificar, consultar y eliminar datos en una base de datos.

El lenguaje SQL se basa en un modelo relacional, donde los datos se organizan en tablas que contienen filas y columnas. Algunas de las operaciones comunes que se pueden realizar con SQL incluyen:

- Crear una tabla
- Insertar datos
- Consultar datos
- Actualizar datos
- Eliminar datos

Además de estas operaciones básicas, SQL también ofrece funciones y cláusulas adicionales que permiten realizar cálculos, agrupaciones, combinaciones de tablas, entre otras operaciones más avanzadas.

Lenguaje de definición de datos (DDL)

El Lenguaje de Definición de Datos (DDL, por sus siglas en inglés) es un conjunto de instrucciones utilizadas para definir la estructura y las características de una base de datos, como la creación, modificación y eliminación de tablas, índices, vistas, procedimientos almacenados y otras entidades relacionadas con la base de datos.

A continuación, se detallarán las sentencias más utilizadas en DDL, junto con ejemplos prácticos para ilustrar su uso:

CREATE: se utiliza para crear una nueva entidad en la base de datos, como una tabla, un índice o una vista. Por ejemplo, para crear una nueva tabla llamada "usuarios", la sentencia sería:

```
CREATE TABLE usuarios (  
  id INT PRIMARY KEY,  
  nombre VARCHAR(50),  
  email VARCHAR(100)  
);
```

ALTER: se utiliza para modificar la estructura de una entidad existente en la base de datos. Por ejemplo, para agregar una nueva columna "telefono" a la tabla "usuarios", la sentencia sería:

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

DROP: se utiliza para eliminar una entidad existente en la base de datos. Por ejemplo, para eliminar la tabla "usuarios", la sentencia sería:

```
DROP TABLE usuarios;
```

TRUNCATE: se utiliza para eliminar todos los registros de una tabla, sin eliminar la estructura de la tabla en sí. Por ejemplo, para eliminar todos los registros de la tabla "usuarios", la sentencia sería:

```
TRUNCATE TABLE usuarios;
```

COMMENT: se utiliza para agregar un comentario a una entidad de la base de datos, como una tabla o una columna. Por ejemplo, para agregar un comentario a la columna "email" de la tabla "usuarios", la sentencia sería:

```
COMMENT ON COLUMN usuarios.email IS 'Dirección de correo electrónico del usuario';
```

GRANT: se utiliza para otorgar permisos a los usuarios o roles para acceder o manipular una entidad de la base de datos. Por ejemplo, para otorgar permisos de lectura y escritura a un usuario llamado "jane" en la tabla "usuarios", la sentencia sería:

```
GRANT SELECT, INSERT, UPDATE ON usuarios TO jane;
```

REVOKE: se utiliza para revocar permisos otorgados anteriormente a los usuarios o roles. Por ejemplo, para revocar los permisos otorgados a "jane" en la tabla "usuarios", la sentencia sería:

```
REVOKE SELECT, INSERT, UPDATE ON usuarios FROM jane;
```

El lenguaje de definición de datos (DDL) es una parte fundamental de la administración de una base de datos, ya que permite crear, modificar y eliminar entidades y objetos en la base de datos. Las sentencias DDL más utilizadas incluyen CREATE, ALTER, DROP, TRUNCATE, COMMENT, GRANT y REVOKE, y se utilizan junto con el lenguaje de manipulación de datos (DML) para gestionar y manipular datos en la base de datos.

Lenguaje de manipulación de datos (DML)

El Lenguaje de Manipulación de Datos (DML) es un sublenguaje de SQL que permite a los usuarios acceder y manipular los datos almacenados en una base de datos relacional. El DML se utiliza para realizar operaciones de lectura, inserción, actualización y eliminación de registros en las tablas de la base de datos. Algunas de las sentencias más utilizadas en DML son:

SELECT: Es la sentencia más comúnmente usada en SQL y se utiliza para recuperar datos de una o varias tablas. Por ejemplo, la siguiente consulta selecciona todos los registros de la tabla "clientes":

SELECT * FROM clientes;

Dentro de la cláusula SELECT de una consulta SQL, puedes especificar los campos específicos que deseas recuperar de una tabla en lugar de usar el asterisco (*) para seleccionar todos los campos. Esto te permite tener un mayor control sobre los datos que se devuelven en el resultado de la consulta.

Para especificar campos específicos, debes enumerarlos separados por comas después de la palabra clave SELECT. Por ejemplo, supongamos que tienes una tabla llamada "clientes" con los siguientes campos: "id", "nombre", "apellido", "email" y "telefono". Si solo deseas seleccionar los campos "nombre" y "email" de la tabla "clientes", puedes escribir la consulta de la siguiente manera:

SELECT nombre, email FROM clientes;

El resultado de esta consulta será una lista de todos los registros de la tabla "clientes", pero solo mostrará los campos "nombre" y "email" para cada registro.

Además de especificar campos específicos, también puedes utilizar funciones de agregación en la cláusula SELECT para realizar cálculos en los datos seleccionados. Algunas funciones de agregación comunes incluyen SUM, COUNT, AVG, MIN y MAX. Por ejemplo, si deseas obtener la suma de los valores en un campo llamado "monto" de la tabla "ventas", puedes escribir la siguiente consulta:

SELECT SUM(monto) FROM ventas;

Esto devolverá un único valor que representa la suma total de los valores en el campo "monto" de la tabla "ventas".

Por otro lado, la cláusula WHERE se utiliza para filtrar los registros de una tabla en función de ciertas condiciones. Puedes combinarla con la cláusula SELECT para especificar criterios más precisos sobre qué registros deseas recuperar.

Por ejemplo, si deseas seleccionar solo los registros de la tabla "clientes" donde el campo "apellido" sea igual a "Garcia", puedes escribir la siguiente consulta:

```
SELECT * FROM clientes WHERE apellido = 'Garcia';
```

Esto devolverá todos los registros de la tabla "clientes" que cumplen con la condición de que el campo "apellido" sea igual a "Garcia".

La cláusula SELECT en SQL es muy potente y versátil, permite seleccionar todos o algunos de los campos en una consulta. También puedes utilizar funciones de agregación en el SELECT para realizar cálculos en los datos seleccionados. La cláusula WHERE se utiliza para filtrar los registros basados en condiciones específicas. Más adelante veremos también las funciones de agrupación, mediante las cláusulas GROUP BY y HAVING, que nos permiten continuar filtrando nuestra consulta dentro de la cláusula SELECT.

INSERT: Se utiliza para agregar nuevos registros a una tabla existente. Por ejemplo, la siguiente consulta inserta un nuevo registro en la tabla "clientes":

```
INSERT INTO clientes (nombre, apellido, edad) VALUES ('Juan', 'Pérez', 30);
```

UPDATE: Se utiliza para modificar los datos de un registro existente en una tabla. Por ejemplo, la siguiente consulta actualiza la edad del cliente con el ID 123:

sql

Copy code

```
UPDATE clientes SET edad = 35 WHERE id = 123;
```

DELETE: Se utiliza para eliminar registros de una tabla. Por ejemplo, la siguiente consulta elimina el registro del cliente con el ID 123:

sql

Copy code

```
DELETE FROM clientes WHERE id = 123;
```

Además de estas sentencias básicas, el DML también proporciona otras funcionalidades avanzadas, como subconsultas, funciones y expresiones condicionales. Estas características permiten a los usuarios escribir consultas más complejas y eficientes para manipular los datos de la base de datos.

Consultas multitaslas

Las consultas multitaslas son aquellas que involucran más de una tabla en su ejecución. Estas consultas son comunes en bases de datos que manejan grandes cantidades de información y se utilizan para obtener información más precisa y detallada.

En lugar de utilizar la cláusula JOIN, es posible realizar consultas multitaslas mediante la emparejamiento de las tablas en la cláusula WHERE, usando una o más condiciones de comparación.

Veamos un ejemplo sencillo. Supongamos que tenemos dos tablas: una tabla de clientes y otra tabla de órdenes. Queremos obtener el nombre de los clientes y la fecha de sus órdenes. Para ello, podemos emparejar las tablas en la cláusula WHERE de la siguiente manera:

sql

Copy code

```
SELECT clientes.nombre, ordenes.fecha  
FROM clientes, ordenes  
WHERE clientes.id = ordenes.cliente_id;
```

En este ejemplo, estamos seleccionando los campos "nombre" y "fecha" de las tablas "clientes" y "ordenes", respectivamente. La cláusula WHERE establece la relación entre ambas tablas mediante la comparación del campo "id" de la tabla "clientes" con el campo "cliente_id" de la tabla "ordenes".

Otro ejemplo podría ser la búsqueda de todos los productos que han sido comprados por un cliente en particular. Supongamos que tenemos una tabla de

clientes, una tabla de órdenes y una tabla de detalles de órdenes. Para obtener los productos de una orden específica, podríamos emparejar las tres tablas de la siguiente manera:

python

Copy code

```
SELECT productos.nombre, detalles_orden.cantidad
FROM clientes, ordenes, detalles_orden, productos
WHERE clientes.id = ordenes.cliente_id
AND ordenes.id = detalles_orden.orden_id
AND detalles_orden.producto_id = productos.id
AND clientes.nombre = 'Juan';
```

En este caso, estamos seleccionando el nombre del producto y la cantidad de la tabla "productos" y "detalles_orden", respectivamente. La cláusula WHERE establece la relación entre las cuatro tablas mediante la comparación de los campos "id" y "cliente_id" de la tabla "clientes", los campos "id" y "orden_id" de la tabla "ordenes", y los campos "producto_id" e "id" de las tablas "detalles_orden" y "productos", respectivamente.

Es importante tener en cuenta que el uso de la cláusula JOIN suele ser más eficiente y legible que el emparejamiento de tablas en la cláusula WHERE, especialmente en consultas más complejas. No obstante, en algunos casos puede ser útil y necesario utilizar el emparejamiento de tablas en la cláusula WHERE para lograr el resultado deseado.

Join

Las consultas multitaslas nos permiten obtener información de varias tablas relacionadas. Para ello, utilizamos la cláusula JOIN que nos permite unir dos o más tablas en una sola consulta.

La cláusula JOIN se utiliza en combinación con la cláusula ON para especificar las condiciones de unión entre las tablas. Existen varios tipos de JOIN que podemos utilizar según nuestras necesidades:

INNER JOIN: Nos devuelve los registros que tienen coincidencias en ambas tablas, es decir, los registros que cumplen la condición especificada en la cláusula ON. Veamos un ejemplo:

sql

Copy code

```
SELECT *
```

```
FROM clientes
```

```
INNER JOIN pedidos
```

```
ON clientes.id = pedidos.cliente_id;
```

En este ejemplo, estamos seleccionando todos los registros de las tablas clientes y pedidos que tengan el mismo valor en el campo cliente_id.

LEFT JOIN: Nos devuelve todos los registros de la tabla de la izquierda (primera tabla que se especifica en la consulta) y los registros que tienen coincidencias en la tabla de la derecha (segunda tabla que se especifica en la consulta). En caso de no haber coincidencias, se devolverán NULL. Veamos un ejemplo:

sql

Copy code

```
SELECT *
```

```
FROM clientes
```

```
LEFT JOIN pedidos
```

```
ON clientes.id = pedidos.cliente_id;
```

En este ejemplo, estamos seleccionando todos los registros de la tabla clientes y los registros de la tabla pedidos que tengan el mismo valor en el campo cliente_id. Si un cliente no tiene pedidos, se devolverá NULL para los campos de la tabla pedidos.

RIGHT JOIN: Es similar al LEFT JOIN, pero devuelve todos los registros de la tabla de la derecha y los registros que tienen coincidencias en la tabla de la izquierda. En caso de no haber coincidencias, se devolverán NULL. Veamos un ejemplo:

sql

Apunte teórico de Bases de Datos

Copy code

```
SELECT *  
FROM clientes  
RIGHT JOIN pedidos  
ON clientes.id = pedidos.cliente_id;
```

En este ejemplo, estamos seleccionando todos los registros de la tabla pedidos y los registros de la tabla clientes que tengan el mismo valor en el campo cliente_id. Si un pedido no tiene cliente asociado, se devolverá NULL para los campos de la tabla clientes.

Es importante mencionar que, en algunos casos, podemos obtener el mismo resultado utilizando LEFT JOIN o RIGHT JOIN, dependiendo del orden en el que se especifican las tablas en la consulta.

Las consultas multitablas con JOIN nos permiten obtener información más completa y detallada, permitiendo relacionar datos de varias tablas en una sola consulta.

Group By

El comando GROUP BY en SQL se utiliza para agrupar las filas que tienen el mismo valor en una o más columnas y aplicar una función agregada como SUM, AVG, COUNT, etc. a las filas agrupadas.

La sintaxis básica del comando GROUP BY es la siguiente:

sql

Copy code

```
SELECT column1, SUM(column2)  
FROM table_name  
WHERE condition  
GROUP BY column1;
```

Esto devolverá los valores únicos de la columna1 y la suma de los valores de la columna2 para cada valor de la columna1.

Aquí hay un ejemplo práctico:

Supongamos que tenemos una tabla llamada "ventas" que tiene los siguientes datos:

ID	Producto	Cantidad	Precio
1	A	10	100
2	B	5	200
3	A	7	150
4	C	12	50
5	B	8	180
6	A	3	120

Si queremos obtener la cantidad total vendida de cada producto, podemos utilizar el siguiente comando:

sql

Copy code

```
SELECT Producto, SUM(Cantidad)
FROM ventas
GROUP BY Producto;
```

Este comando nos devolverá:

Producto	SUM(Cantidad)
A	20
B	13
C	12

Aquí estamos agrupando las filas por la columna "Producto" y sumando la cantidad vendida para cada producto.

También podemos utilizar el comando GROUP BY con varias columnas. Por ejemplo, si queremos obtener la cantidad total vendida de cada producto a un precio determinado, podemos utilizar el siguiente comando:

sql

Apunte teórico de Bases de Datos

Copy code

```
SELECT Producto, Precio, SUM(Cantidad)
```

```
FROM ventas
```

```
GROUP BY Producto, Precio;
```

Este comando nos devolverá:

Producto	Precio	SUM(Cantidad)
A	100	10
A	120	3
A	150	7
B	180	8
B	200	5
C	50	12

Aquí estamos agrupando las filas por las columnas "Producto" y "Precio" y sumando la cantidad vendida para cada combinación de producto y precio.

También podemos utilizar el comando GROUP BY con funciones agregadas como COUNT, AVG, MIN, MAX, etc. Por ejemplo, si queremos obtener el precio máximo y mínimo de cada producto, podemos utilizar el siguiente comando:

sql

Copy code

```
SELECT Producto, MAX(Precio), MIN(Precio)
```

```
FROM ventas
```

```
GROUP BY Producto;
```

Este comando nos devolverá:

Producto	MAX(Precio)	MIN(Precio)
A	150	100
B	200	180
C	50	50

Aquí estamos agrupando las filas por la columna "Producto" y obteniendo el precio máximo y mínimo para cada producto utilizando las funciones MAX y MIN.

Order By

El comando ORDER BY se utiliza en consultas SQL para ordenar los resultados en un orden específico. Se puede ordenar por una o varias columnas y en orden ascendente o descendente.

La sintaxis básica es la siguiente:

sql

Copy code

```
SELECT columna1, columna2, columna3
```

```
FROM tabla
```

```
ORDER BY columna1 ASC/DESC, columna2 ASC/DESC, ...;
```

Donde columna1, columna2, columna3 son las columnas por las cuales se desea ordenar y ASC o DESC indican si se quiere un orden ascendente o descendente.

A continuación se presenta un ejemplo de cómo se puede utilizar la cláusula ORDER BY:

Supongamos que tenemos la siguiente tabla "estudiantes":

id	nombre	edad	promedio
1	Ana	18	9.5
2	Benjamín	20	8.5
3	Carlos	19	7.2
4	Daniela	21	9.0
5	Eduardo	18	6.5

Para ordenar la tabla por nombre en orden ascendente se puede utilizar la siguiente consulta:

sql

Copy code

```
SELECT * FROM estudiantes
```

```
ORDER BY nombre ASC;
```

Lo cual producirá la siguiente tabla ordenada:

id	nombre	edad	promedio
1	Ana	18	9.5
2	Benjamín	20	8.5
3	Carlos	19	7.2
4	Daniela	21	9.0
5	Eduardo	18	6.5

Si se quisiera ordenar por promedio en orden descendente y luego por edad en orden ascendente, se podría utilizar la siguiente consulta:

sql

Copy code

```
SELECT * FROM estudiantes
```

```
ORDER BY promedio DESC, edad ASC;
```

Lo cual producirá la siguiente tabla ordenada:

id	nombre	edad	promedio
1	Ana	18	9.5
4	Daniela	21	9.0
2	Benjamín	20	8.5
3	Carlos	19	7.2
5	Eduardo	18	6.5

Es importante tener en cuenta que se pueden ordenar los resultados por cualquier columna de la tabla, incluso si no se están mostrando en la consulta. Además, se pueden utilizar funciones para ordenar los resultados, por ejemplo, ordenar por la longitud de una cadena.

En resumen, el comando ORDER BY es útil para ordenar los resultados de una consulta de acuerdo a un orden específico, y se puede utilizar con una o varias columnas y en orden ascendente o descendente.

Subconsultas

Las subconsultas en SQL son consultas anidadas dentro de otra consulta. Estas subconsultas se pueden utilizar para realizar operaciones complejas en una sola consulta. La sintaxis básica de una subconsulta es la siguiente:

sql

Copy code

```
SELECT columna1, columna2, ... FROM tabla1 WHERE columnaX OPERADOR  
(SELECT columnaY FROM tabla2 WHERE condicion);
```

En la cláusula WHERE de la consulta externa se especifica una condición que debe cumplirse y en la subconsulta se especifica una columna y una tabla que se utilizará para realizar la comparación. El resultado de la subconsulta se utiliza para compararlo con el valor de la columna de la tabla principal.

Existen dos tipos de subconsultas: las subconsultas escalares y las subconsultas de varias columnas.

Las subconsultas escalares son aquellas que devuelven un solo valor, y se utilizan en la cláusula SELECT, WHERE o HAVING. Por ejemplo:

sql

Copy code

```
SELECT nombre, apellido, (SELECT COUNT(*) FROM pedidos WHERE cliente_id =  
clientes.id) AS cantidad_pedidos FROM clientes;
```

En este ejemplo, la subconsulta se utiliza en la cláusula SELECT para contar la cantidad de pedidos que tiene cada cliente y se muestra como una columna adicional llamada "cantidad_pedidos".

Las subconsultas de varias columnas son aquellas que devuelven varias columnas y se utilizan en la cláusula FROM o JOIN. Por ejemplo:

sql

Copy code

```
SELECT c.nombre, p.descripcion FROM clientes c, (SELECT * FROM pedidos  
WHERE fecha = '2022-01-01') p WHERE c.id = p.cliente_id;
```

En este ejemplo, la subconsulta se utiliza en la cláusula FROM para seleccionar los pedidos realizados en una fecha determinada y se unen con la tabla de clientes mediante el campo "cliente_id".

También es posible utilizar operadores como IN o EXISTS para comparar una columna con los resultados de una subconsulta. Por ejemplo:

sql

Copy code

```
SELECT nombre FROM clientes WHERE id IN (SELECT cliente_id FROM pedidos  
WHERE fecha = '2022-01-01');
```

En este ejemplo, la subconsulta devuelve una lista de identificadores de clientes que han realizado pedidos en la fecha especificada y se utiliza el operador IN para compararlos con la columna "id" de la tabla de clientes.

Es importante tener en cuenta que las subconsultas pueden afectar el rendimiento de la consulta, especialmente cuando se utilizan en tablas grandes o complejas. Por lo tanto, se recomienda utilizarlas con precaución y optimizar las consultas siempre que sea posible.