Consultas SQL

Continuamos incorporando más funciones y cláusulas que nos permiten obtener conjuntos de datos y/o validar información que visualizamos desde el aplicativo al momento de ejecutar los casos de prueba.

Funciones de Agregación

En SQL, las funciones de agregación son funciones que se utilizan para realizar cálculos sobre un conjunto de valores. Estas funciones operan sobre un conjunto de valores y devuelven un único valor de resultado. Las funciones de agregación más comunes son

SUM: devuelve la suma de los valores de una columna.
COUNT: devuelve la cantidad de filas en una tabla o la cantidad de valores distintos de una columna.
AVG: devuelve el promedio de los valores de una columna.
MAX: devuelve el valor máximo de una columna.
MIN: devuelve el valor mínimo de una columna.

Veamos algunos ejemplos

Ejemplo 1: Seleccionar la cantidad de registros de la tabla "productos

```sql
SELECT COUNT(*) FROM productos;
```

Ejemplo 2: Seleccionar la suma de los valores en la columna "cantidad" de la tabla "ventas".

```sql
SELECT SUM(cantidad) FROM ventas;
```

Ejemplo 3: Calcular el promedio de la columna "precio" en la tabla "productos".

```sql
SELECT AVG(precio) FROM productos;
```

Ejemplo 4: Calcular el promedio de la columna "edad" en la tabla "usuarios" para aquellos usuarios que tienen un estado civil de "casado".

```sql
SELECT AVG(edad) FROM usuarios WHERE estado_civil = 'casado';
```

Ejemplo 5: Calcular el promedio de la columna "nota" en la tabla "alumnos" para aquellos alumnos que tienen un nombre que comienza con la letra "A".

```sql
SELECT AVG(nota) FROM alumnos WHERE nombre LIKE 'A%';
```

Ejemplo 6: Encontrar el valor mínimo de la columna "precio" en la tabla "productos"

```sql
SELECT MIN(precio) FROM productos;
```

Ejemplo 7: Encontrar la edad mínima de los usuarios en la tabla "usuarios" para aquellos usuarios que tienen un estado civil de "soltero".

```sql
SELECT MIN(edad) FROM usuarios WHERE estado_civil = 'soltero';
```

Ejemplo 8: Encontrar el valor máximo de la columna "precio" en la tabla "productos".

```sql
SELECT MAX(precio) FROM productos;
```

Ejemplo 9: Encontrar la edad máxima de los usuarios en la tabla "usuarios" para aquellos usuarios que tienen un estado civil de "casado".

```sql
SELECT MAX(edad) FROM usuarios WHERE estado_civil = 'casado';
```

Estas funciones se utilizan comúnmente en combinación con la cláusula GROUP BY para calcular valores agregados para cada grupo de valores en una columna. Por ejemplo, si quisieras calcular la cantidad total de ventas para cada producto en una tabla de ventas, podrías utilizar la función SUM y agrupar los resultados por el nombre del producto.

Es importante tener en cuenta que las funciones de agregación solo se pueden utilizar en la cláusula SELECT y no se pueden utilizar en la cláusula WHERE o en la cláusula ORDER BY. Además, algunas bases de datos pueden tener funciones de agregación adicionales o distintas de las mencionadas anteriormente.

Veamos la cláusula GROUP BY y cómo se utiliza con las funciones de agregación.

Cláusula GROUP BY

La cláusula GROUP BY para agrupar los datos de una tabla por valores en una o varias columnas, veamos algunos ejemplos:

Ejemplo 1: Seleccionar la cantidad de registros por ciudad en la tabla "clientes".

```sql
SELECT ciudad, COUNT(*) FROM clientes GROUP BY ciudad;
```

Ejemplo 2: Seleccionar la cantidad de productos vendidos por categoría en la tabla "ventas"

```sql
SELECT categoria, SUM(cantidad) FROM ventas GROUP BY categoria;
```

Funciones de Fechas

Es posible que necesitemos trabajar información teniendo en cuenta fechas. Las funciones de fecha en SQL son muy útiles para trabajar con datos que incluyen fechas. En la cláusula GROUP BY, se pueden utilizar estas funciones para agrupar los datos por período de tiempo, como por año, mes, semana o día, como también en la cláusula SELECT. Veamos algunos ejemplos más agregando trabajo de fechas:

Ejemplo 1: Seleccionar la cantidad de pedidos realizados por día en la tabla "pedidos

```
SELECT DATE(fecha), COUNT(*) FROM pedidos GROUP BY DATE(fecha);
```

Ejemplo 2: Seleccionar la cantidad de ventas totales y promedio por mes en la tabla "ventas"

```
SELECT MONTH(fecha), SUM(total), AVG(total) FROM ventas GROUP BY MONTH(fecha);
```

En estos dos ejemplos, la cláusula GROUP BY se utiliza para agrupar los datos de una tabla por valores en una o varias columnas. La cláusula SELECT se utiliza para seleccionar los valores a agrupar, y las funciones de agregación como SUM, COUNT y AVG se utilizan para realizar cálculos sobre los datos agrupados

Palabra clave AS

La palabra clave AS se utiliza para asignar un alias o un nombre alternativo a una columna o una tabla. Es opcional, ya que SQL permite especificar los nombres de columna y tabla directamente sin utilizar AS. Sin embargo, a menudo se utiliza para hacer que el resultado de la consulta sea más fácil de leer y entender

Ejemplo 1: Agrupar los datos por año y mes a partir de la fecha en la columna "fecha" en la tabla "pedidos" y contar la cantidad de pedidos para cada mes

```
SELECT YEAR(fecha) AS año, MONTH(fecha) AS mes, COUNT(*) AS cantidad_pedidos
FROM pedidos
GROUP BY YEAR(fecha), MONTH(fecha);
```

Ejemplo 2: Agrupar los datos por día a partir de la fecha en la columna "fecha" en la tabla "ventas" y calcular la suma total de las ventas para cada día.

```
SELECT DATE(fecha) AS dia, SUM(venta_total) AS total_ventas
FROM ventas
GROUP BY DATE(fecha);
```

Ejemplo 3: Agrupar los datos por semana a partir de la fecha en la columna "fecha" en la tabla "registros" y contar la cantidad de registros para cada semana.

```
SELECT YEARWEEK(fecha) AS semana, COUNT(*) AS cantidad_registros
FROM registros
GROUP BY YEARWEEK(fecha);
```

En estos ejemplos vimos cómo se utilizan las funciones YEAR, MONTH, DATE y YEARWEEK para agrupar los datos por período de tiempo, y la cláusula GROUP BY se utiliza para agrupar los resultados por estas funciones de fecha. Luego, se aplican funciones de agregación como COUNT y SUM para realizar cálculos

sobre los datos agrupados. Además, utilizamos AS para clarificar los datos que vemos en esas columnas calculadas.

Consultas con más de una tabla

Las consultas SQL con más de una tabla se realizan utilizando la cláusula JOIN. La cláusula JOIN se utiliza para combinar filas de dos o más tablas basándose en una columna común. Existen varios tipos de JOIN en SQL, pero los más comunes son INNER JOIN, LEFT JOIN y RIGHT JOIN

INNER JOIN

INNER JOIN se utiliza para combinar filas de dos o más tablas basándose en una columna común. Puede ser útil en situaciones en las que se necesita combinar datos de diferentes tablas para obtener información más completa. Veamos primero la sintaxis para combinar datos de más de una tabla utilizando inner join.

Sintaxis 1: Combinar dos tablas por una columna común

```
SELECT *
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

En este modelo, se seleccionan todas las columnas de las tablas "tabla1" y "tabla2" utilizando INNER JOIN para combinar las filas basadas en la columna común "columna_comun", se visualizarán todas las columnas de ambas tablas.

Sintaxis 2: Seleccionar columnas específicas de dos tablas con INNER JOIN

```
SELECT tabla1.columna1, tabla2.columna2
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun;
```

En este modelo, se seleccionan dos columnas específicas de las tablas "tabla1" y "tabla2". Las filas se combinan utilizando INNER JOIN basándose en la columna común "columna_comun"

Sintaxis 3: Combinar tres tablas utilizando INNER JOIN

```
SELECT tabla1.columna1, tabla2.columna2, tabla3.columna3
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna_comun = tabla2.columna_comun
INNER JOIN tabla3
ON tabla2.columna_comun = tabla3.columna_comun;
```

En este modelo, se combinan las filas de tres tablas, "tabla1", "tabla2" y "tabla3", utilizando INNER JOIN. Las filas de "tabla1" y "tabla2" se combinan utilizando la columna común "columna_comun", y las filas de "tabla2" y "tabla3" se combinan utilizando la misma columna común.

Algunos ejemplos concretos:

Ejemplo 1: Combinar dos tablas "clientes" y "pedidos" por una columna común "id_cliente".

```
SELECT clientes.nombre, pedidos.fecha
FROM clientes
INNER JOIN pedidos
ON clientes.id_cliente = pedidos.id_cliente;
```

En este ejemplo, se seleccionan el nombre del cliente y la fecha del pedido de las tablas "clientes" y "pedidos", respectivamente. Las filas se combinan utilizando INNER JOIN basándose en la columna común "id_cliente"

Ejemplo 2: Combinar tres tablas "productos", "pedidos" y "proveedores" utilizando INNER JOIN.

```
SELECT productos.nombre, proveedores.nombre, pedidos.cantidad
FROM productos
INNER JOIN pedidos
ON productos.id_producto = pedidos.id_producto
INNER JOIN proveedores
ON productos.id_proveedor = proveedores.id_proveedor;
```

En este ejemplo, se seleccionan el nombre del producto, el nombre del proveedor y la cantidad pedida del pedido de las tablas "productos", "pedidos" y "proveedores", respectivamente. Las filas se combinan utilizando INNER JOIN. Las filas de "productos" y "pedidos" se combinan utilizando la columna común "id_producto", y las filas de "productos" y "proveedores" se combinan utilizando la columna común "id_proveedor"

Ejemplo 3: Combinar dos tablas "empleados" y "departamentos" utilizando INNER JOIN y seleccionar solo los empleados que trabajan en el departamento "Ventas".

```
SELECT empleados.nombre, departamentos.nombre
FROM empleados
INNER JOIN departamentos
ON empleados.id_departamento = departamentos.id_departamento
WHERE departamentos.nombre = 'Ventas';
```

En este ejemplo, se seleccionan el nombre del empleado y el nombre del departamento de las tablas "empleados" y "departamentos", respectivamente. Las filas se combinan utilizando INNER JOIN basándose en la columna común "id_departamento". La cláusula WHERE se utiliza para filtrar los resultados y seleccionar solo los empleados que trabajan en el departamento "Ventas.

LEFT JOIN

LEFT JOIN es una herramienta útil en SQL para mostrar todas las filas de la tabla izquierda y las filas correspondientes de la tabla derecha, y nulos en los campos de la tabla derecha si no hay una correspondencia en la tabla izquierda. Veamos algunos ejemplos:

Ejemplo 1: Combinar dos tablas "clientes" y "pedidos" por una columna común "id_cliente", y mostrar todos los clientes, incluso aquellos que no han realizado ningún pedido.

```
SELECT clientes.nombre, pedidos.fecha
FROM clientes
LEFT JOIN pedidos
ON clientes.id_cliente = pedidos.id_cliente;
```

En este ejemplo, se seleccionan el nombre del cliente y la fecha del pedido de las tablas "clientes" y "pedidos", respectivamente. Las filas se combinan utilizando LEFT JOIN basándose en la columna común "id_cliente". Como resultado, se muestran todos los clientes, incluso aquellos que no tienen un pedido registrado en la tabla "pedidos".

Ejemplo 2: Combinar dos tablas "estudiantes" y "cursos" utilizando LEFT JOIN, y mostrar todos los estudiantes, incluso aquellos que no están inscritos en ningún curso.

```
SELECT estudiantes.nombre, cursos.nombre
FROM estudiantes
LEFT JOIN cursos
ON estudiantes.id_estudiante = cursos.id_estudiante;
```

En este ejemplo, se seleccionan el nombre del estudiante y el nombre del curso de las tablas "estudiantes" y "cursos", respectivamente. Las filas se combinan utilizando LEFT JOIN basándose en la columna común "id_estudiante". Como resultado, se muestran todos los estudiantes, incluso aquellos que no están inscritos en ningún curso registrado en la tabla "cursos".

Ejemplo 3: Combinar dos tablas "empleados" y "jefes" utilizando LEFT JOIN, y mostrar todos los empleados, incluso aquellos que no tienen un jefe asignado.

```sql
SELECT empleados.nombre, jefes.nombre
FROM empleados
LEFT JOIN jefes
ON empleados.id_jefe = jefes.id_jefe;
```

En este ejemplo, se seleccionan el nombre del empleado y el nombre del jefe de las tablas "empleados" y "jefes", respectivamente. Las filas se combinan utilizando LEFT JOIN basándose en la columna común "id_jefe". Como resultado, se muestran todos los empleados, incluso aquellos que no tienen un jefe asignado registrado en la tabla "jefes".

RIGHT JOIN

RIGHT JOIN es una herramienta útil en SQL para mostrar todas las filas de la tabla derecha y las filas correspondientes de la tabla izquierda, y nulos en los campos de la tabla izquierda si no hay una correspondencia en la tabla derecha. Veamos algunos ejemplos:

Ejemplo 1: Combinar dos tablas "pedidos" y "clientes" utilizando RIGHT JOIN, y mostrar todos los pedidos, incluso aquellos que no están asociados con un cliente.

```sql
SELECT clientes.nombre, pedidos.fecha
FROM clientes
RIGHT JOIN pedidos
ON clientes.id_cliente = pedidos.id_cliente;
```

En este ejemplo, se seleccionan el nombre del cliente y la fecha del pedido de las tablas "clientes" y "pedidos", respectivamente. Las filas se combinan utilizando RIGHT JOIN basándose en la columna común "id_cliente". Como resultado, se muestran todos los pedidos, incluso aquellos que no están asociados con un cliente registrado en la tabla "clientes".

Ejemplo 2: Combinar dos tablas "cursos" y "estudiantes" utilizando RIGHT JOIN, y mostrar todos los cursos, incluso aquellos que no tienen ningún estudiante inscrito.

```
SELECT estudiantes.nombre, cursos.nombre
FROM estudiantes
RIGHT JOIN cursos
ON estudiantes.id_estudiante = cursos.id_estudiante;
```

En este ejemplo, se seleccionan el nombre del estudiante y el nombre del curso de las tablas "estudiantes" y "cursos", respectivamente. Las filas se combinan utilizando RIGHT JOIN basándose en la columna común "id_estudiante". Como resultado, se muestran todos los cursos, incluso aquellos que no tienen ningún estudiante inscrito registrado en la tabla "estudiantes"..

Ejemplo 3: Combinar dos tablas "jefes" y "empleados" utilizando RIGHT JOIN, y mostrar todos los jefes, incluso aquellos que no tienen empleados asignados.

```
SELECT empleados.nombre, jefes.nombre
FROM empleados
RIGHT JOIN jefes
ON empleados.id_jefe = jefes.id_jefe;
```

En este ejemplo, se seleccionan el nombre del empleado y el nombre del jefe de las tablas "empleados" y "jefes", respectivamente. Las filas se combinan utilizando RIGHT JOIN basándose en la columna común "id_jefe". Como resultado, se muestran todos los jefes, incluso aquellos que no tienen empleados asignados registrado en la tabla "empleados".