Universidad Nacional Autónoma de México

Facultad de Ingeniería

Ingeniería en Computación

Bases de datos.

Tarea 21.

Profesor:
Ing. Fernando Arreola Franco

Alumno:
Santiago Martínez Ricardo

N. Cuenta:
318187251

Grupo:
04

Tarea 21:

Investigar casos de uso, restricciones, ejemplos de los siguientes comandos:

Dentro del manejador de postgres la información encontrada es la siguiente.

SELECT

La cláusula SELECT se utiliza para recuperar datos de una o varias tablas en una base de datos.

Casos de uso

Recuperar todas las columnas: Utilizando la cláusula SELECT seguida de un asterisco (*), se pueden recuperar todas las columnas de una tabla en particular. Por ejemplo:

```
SELECT * FROM tabla;
```

Selección de columnas específicas: Se pueden especificar las columnas específicas que se desean recuperar en la cláusula SELECT, separadas por comas. Esto permite seleccionar solo la información necesaria. Por ejemplo:

```
SELECT columna1, columna2 FROM tabla;
```

Uso de funciones y expresiones: Permite utilizar funciones incorporadas de PostgreSQL, como SUM, COUNT, AVG, entre otras, para realizar cálculos y agregaciones en los datos recuperados. También se puede usar para combinar y transformar los valores de las columnas.

Alias de columnas: Se pueden asignar alias a las columnas en la cláusula SELECT utilizando la palabra clave AS. Esto permite cambiar el nombre de las columnas en los resultados de la consulta.

Consultas con múltiples tablas: La cláusula SELECT se puede utilizar para seleccionar columnas de múltiples tablas mediante el uso de JOIN y especificando cómo se deben combinar las tablas.

Ordenamiento y limitación de resultados: Con la cláusula SELECT, se puede ordenar los resultados utilizando ORDER BY y limitar el número de filas devueltas utilizando LIMIT.

Restricciones

Sintaxis correcta: La cláusula SELECT debe seguir la sintaxis adecuada en PostgreSQL. Debe incluir la palabra clave "SELECT" seguida de las columnas que se desean seleccionar. Además, es importante tener en cuenta el uso correcto de las comillas y los operadores lógicos si se aplican condiciones.

Privilegios de acceso: La cláusula SELECT está sujeta a los privilegios de acceso en la base de datos. El usuario que realiza la consulta debe tener los permisos necesarios para acceder a las tablas y columnas especificadas en la cláusula SELECT.

Tipos de datos: Es importante considerar los tipos de datos de las columnas seleccionadas en la cláusula SELECT, ya que pueden influir en los resultados y en las operaciones posteriores.

Agrupación y funciones de agregación: Al utilizar funciones de agregación como COUNT, SUM, AVG, etc., es importante agrupar los resultados correctamente utilizando GROUP BY y especificar las columnas apropiadas.

Optimización de consultas: En consultas más complejas, es importante optimizar la cláusula SELECT para mejorar el rendimiento, considerando el uso de índices, la evitación de subconsultas innecesarias y el uso adecuado de cláusulas como WHERE.

Ejemplo

Se muestran todas las columnas de la tabla con:

```
SELECT * FROM nombre_de_tabla;
```

FROM

En PostgreSQL, la cláusula FROM se utiliza para especificar las tablas o vistas de las cuales se obtendrán los datos en una consulta.

Casos de uso

Consultas simples: La cláusula FROM se utiliza comúnmente en consultas simples para especificar la tabla o tablas de las cuales se desea recuperar datos.

Unión de tablas: Se puede utilizar la cláusula FROM para unir múltiples tablas en una consulta utilizando JOIN. Esto permite combinar datos de diferentes tablas basándose en condiciones específicas.

Consultas con subconsultas: La cláusula FROM también se puede utilizar para incluir subconsultas en una consulta principal. Esto permite realizar consultas más complejas utilizando los resultados de otras consultas.

Consultas con vistas: Si se han creado vistas en la base de datos, se pueden incluir en la cláusula FROM para utilizar los resultados predefinidos de esas vistas en una consulta.

Restricciones

Sintaxis correcta: La cláusula FROM debe seguir la sintaxis adecuada en PostgreSQL. Debe incluir la palabra clave "FROM" seguida de la tabla o tablas de las cuales se obtendrán los datos. Además, si se realizan uniones o se utilizan subconsultas, es importante seguir las reglas de sintaxis correspondientes.

Tablas existentes: Las tablas especificadas en la cláusula FROM deben existir en la base de datos. Si se intenta hacer referencia a una tabla que no existe, se generará un error.

Privilegios de acceso: La cláusula FROM está sujeta a los privilegios de acceso en la base de datos. El usuario que realiza la consulta debe tener los permisos necesarios para acceder a las tablas o vistas especificadas en la cláusula FROM.

Conflictos de nombres: Si se utilizan múltiples tablas en la cláusula FROM y estas tienen columnas con nombres idénticos, es posible que se generen conflictos de nombres. Para evitar esto, se pueden utilizar alias de tabla para distinguir las columnas.

EJEMPLO

Supongamos que se tienen dos tablas: "clientes" y "pedidos". La tabla "clientes" almacena información sobre los clientes, mientras que la tabla "pedidos" contiene información sobre los pedidos realizados por los clientes.

Para obtener los nombres de los clientes y sus respectivos pedidos, puedes utilizar la cláusula FROM de la siguiente manera:

```
SELECT clientes.nombre, pedidos.descripcion

FROM clientes

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

JOIN

JOIN se utiliza para combinar datos de dos o más tablas basándose en una condición específica.

Casos de uso

Combinar datos relacionados: La cláusula JOIN se utiliza para combinar datos de dos o más tablas cuando existe una relación entre ellas. Por ejemplo, se puede combinar una tabla de clientes con una tabla de pedidos para obtener información completa sobre los pedidos realizados por cada cliente.

Consultas con múltiples tablas: La cláusula JOIN permite realizar consultas complejas que involucran múltiples tablas. Puedes unir tablas adicionales para obtener datos más detallados o para realizar cálculos basados en información de diferentes fuentes.

Consultas con relaciones uno a muchos: Si tienes una relación uno a muchos entre dos tablas, como una tabla de clientes y una tabla de pedidos, puedes utilizar la cláusula JOIN para obtener todos los pedidos relacionados con cada cliente.

Consultas con relaciones muchos a muchos: Si tienes una relación muchos a muchos entre dos tablas, como una tabla de productos y una tabla de categorías, puedes utilizar la cláusula JOIN para obtener todos los productos que pertenecen a una categoría específica.

Restricciones

Columnas de unión adecuadas: Al utilizar la cláusula JOIN, es importante especificar las columnas adecuadas para unir las tablas. Deben existir columnas relacionadas en ambas tablas para realizar la unión correctamente.

Sintaxis correcta: La cláusula JOIN debe seguir la sintaxis adecuada en PostgreSQL. Existen diferentes tipos de JOIN, como INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL JOIN, cada uno con su propia sintaxis y comportamiento.

Privilegios de acceso: Al realizar JOIN entre tablas, se debe tener en cuenta los privilegios de acceso necesarios para acceder a las tablas involucradas. El usuario que realiza la consulta debe tener los permisos adecuados para leer los datos en todas las tablas utilizadas.

Rendimiento de la consulta: En consultas con JOIN, es importante considerar el rendimiento de la consulta y utilizar índices adecuados en las columnas utilizadas para la unión. Un mal uso de JOIN o la falta de índices puede afectar negativamente el rendimiento de la consulta.

Ejemplo

Utilizando el mismo ejemplo que con FROM se tiene que:

```
SELECT clientes.nombre, pedidos.descripcion
FROM clientes
JOIN pedidos ON clientes.cliente_id = pedidos.cliente_id;
```

WHERE

WHERE se utiliza para filtrar filas de una tabla basándose en una condición específica.

Casos de uso

Filtrado de datos: La cláusula WHERE se utiliza para filtrar datos según una condición específica. Por ejemplo, puedes utilizarla para obtener solo los registros donde el valor de una columna sea igual a cierto valor, mayor que otro valor, o cumpla con cualquier otra condición lógica.

Búsqueda de registros específicos: Puedes utilizar WHERE para buscar registros que cumplan con múltiples condiciones combinadas con operadores lógicos como AND y OR. Por ejemplo, encontrar registros donde se cumpla una condición en una columna y otra condición en otra columna.

Consultas basadas en rangos: La cláusula WHERE se puede utilizar para realizar consultas basadas en rangos, como obtener registros con valores entre ciertos límites o dentro de un rango de fechas específico.

Eliminación y actualización de registros: WHERE también se utiliza en instrucciones DELETE y UPDATE para especificar qué registros se eliminarán o actualizarán en función de una condición.

Restricciones

Sintaxis correcta: Es importante seguir la sintaxis adecuada al utilizar WHERE en PostgreSQL. La cláusula WHERE debe ir después de la cláusula FROM y debe incluir una condición lógica válida. Además, los operadores y los valores utilizados en la condición deben ser compatibles con el tipo de datos de las columnas involucradas.

Tipos de datos compatibles: Los valores utilizados en la cláusula WHERE deben ser del tipo de datos correcto y ser compatibles con las columnas involucradas en la condición. De lo contrario, se pueden producir errores de tipo de datos.

Índices y rendimiento: Si tienes tablas grandes y necesitas realizar consultas eficientes utilizando WHERE, es importante considerar la creación de índices en las columnas utilizadas en las condiciones de filtrado. Los índices pueden mejorar el rendimiento de las consultas y acelerar el proceso de búsqueda.

Privilegios de acceso: La cláusula WHERE está sujeta a los privilegios de acceso en la base de datos. El usuario que realiza la consulta debe tener los permisos necesarios para acceder a las tablas y columnas involucradas, así como para ejecutar operaciones de filtrado.

Ejemplo

Suponiendo que se tiene una tabla llamada empleados que contiene información sobre los empleados de una fabrica,y se quiere seleccionar aquellos empleados cuyo salario sea mayor a $3000. Entonces se puede usar un:

```
SELECT nombre, salario
FROM empleados
WHERE salario > 3000;
```

HAVING

HAVING se utiliza para filtrar los resultados de una consulta que incluye una función de agregación, como SUM, COUNT o AVG.

Casos de uso

Filtrado de resultados de funciones de agregación: La cláusula HAVING se utiliza para filtrar los resultados de funciones de agregación basándose en una condición específica. Por ejemplo, puedes utilizarla para seleccionar solo aquellos grupos que cumplen con cierta condición, como seleccionar solo aquellos departamentos que tienen más de 10 empleados.

Filtrado después de la agrupación: HAVING se utiliza después de la cláusula GROUP BY para aplicar una condición de filtrado a los grupos formados por la agrupación.

Consultas con funciones de agregación múltiples: Si se tienen consultas que involucran múltiples funciones de agregación, puedes utilizar HAVING para filtrar los resultados basándote en las condiciones relacionadas con esas funciones.

Restricciones

Uso después de GROUP BY: La cláusula HAVING se utiliza después de la cláusula GROUP BY y antes de la cláusula ORDER BY en una consulta. Es necesario agrupar los datos utilizando GROUP BY antes de aplicar la cláusula HAVING.

Condiciones de filtrado válidas: Las condiciones utilizadas en HAVING deben ser válidas y pueden incluir operadores de comparación, operadores lógicos y funciones de agregación. Las columnas referenciadas en la condición deben estar incluidas en la cláusula GROUP BY o ser parte de una función de agregación.

Sintaxis correcta: Es importante seguir la sintaxis adecuada al utilizar HAVING en PostgreSQL. La cláusula HAVING debe seguir la estructura de "HAVING condicion", donde "condicion" es la expresión que define el filtro basado en las funciones de agregación.

Rendimiento de la consulta: Al utilizar HAVING en consultas con grandes conjuntos de datos, es importante tener en cuenta el rendimiento de la consulta. Pueden ser necesarios índices adecuados en las columnas utilizadas para la agrupación y en las columnas referenciadas en la condición de HAVING para optimizar el rendimiento.

Ejemplo

Se tiene una tabla llamada "empleados" que almacena información sobre los empleados de una empresa, se quiere obtener aquellos departamentos que tienen al menos 3 empleados. Bajo esta premisa se puede utilizar

```
SELECT departamento, COUNT(*) AS cantidad_empleados
FROM empleados
GROUP BY departamento
HAVING COUNT(*) >= 3;
```

CORRELACIONADAS

Las consultas correlacionadas permiten relacionar subconsultas internas con la consulta principal utilizando valores de las filas actuales de la consulta principal

Casos de uso

Filtros basados en condiciones relacionadas: Las consultas correlacionadas son útiles cuando se requiere filtrar los resultados de la consulta principal basándose en condiciones relacionadas con valores de las filas actuales. Por ejemplo, puedes utilizar una consulta correlacionada para seleccionar todos los empleados cuyo salario sea mayor que el salario promedio de su departamento.

Cálculos basados en datos relacionados: Las consultas correlacionadas también se utilizan para realizar cálculos basados en valores relacionados de la consulta principal. Por ejemplo, se puede utilizar una consulta correlacionada para calcular el total de ventas de un producto sumando las cantidades vendidas en cada transacción relacionada.

Subconsultas dinámicas: Las consultas correlacionadas permiten realizar subconsultas dinámicas, donde los resultados de la subconsulta dependen de los valores de las filas actuales de la consulta principal. Esto brinda flexibilidad y permite adaptar las subconsultas a diferentes condiciones en tiempo de ejecución.

Restricciones

Rendimiento de la consulta: Las consultas correlacionadas pueden tener un impacto significativo en el rendimiento, especialmente cuando se ejecutan en grandes conjuntos de datos. Es importante optimizar y evaluar cuidadosamente el rendimiento de las consultas correlacionadas utilizando índices adecuados en las columnas involucradas y asegurándose de que las consultas sean eficientes.

Cuidado con las consultas anidadas: Las consultas correlacionadas a menudo se utilizan en combinación con subconsultas anidadas. Sin embargo, es importante tener en cuenta que un uso excesivo de consultas anidadas y correlacionadas puede complicar la comprensión de la consulta y dificultar el mantenimiento y la optimización.

Limitaciones en la modificación de datos: Las consultas correlacionadas no se pueden utilizar directamente para realizar modificaciones de datos, como

inserciones, actualizaciones o eliminaciones. En tales casos, se requiere un enfoque diferente, como el uso de instrucciones UPDATE o DELETE.

Ejemplo

Continuando con el ejemplo de "clientes" y "órdenes", donde la tabla "órdenes" registra las órdenes de compra realizadas por los clientes. El objetivo parte de mostrar aquellos clientes que hayan realizado más de una orden. Con ayuda de una consulta correlacionada esto se puede resolver de la siguiente manera:

SELECT nombre_cliente

FROM clientes c

WHERE (SELECT COUNT(*) FROM órdenes o WHERE orden.id_cliente = cliente.id) > 1;

Donde la la subconsulta (SELECT COUNT(*) FROM órdenes o WHERE orden.id_cliente = cliente.id) se correlaciona con la tabla "clientes" utilizando la condición orden.id_cliente = cliente.id. Esto significa que para cada fila de la tabla "clientes", se ejecutará la subconsulta y contará cuántas órdenes tienen un ID de cliente coincidente.

Luego, la condición > 1 se utiliza para filtrar los resultados y seleccionar solo aquellos clientes que tienen más de una orden.

Referencias

- PostgreSQL Global Development Group. (2021). PostgreSQL 13.4 Documentation. Consultado el 24 de Mayo del 2023 en https://www.postgresql.org/docs/13/