SELECT

Casos de uso

- Consulta de datos específicos: SELECT se utiliza para recuperar datos específicos que coincidan con ciertos criterios, como una consulta para obtener información sobre todos los clientes que residen en una ciudad específica.
- Operaciones de filtrado y ordenación: Puedes usar SELECT para filtrar datos según ciertas condiciones, como encontrar todos los productos con un precio superior a cierto valor. También puedes ordenar los resultados utilizando la cláusula ORDER BY.
- Agregación de datos: Con SELECT, puedes realizar operaciones de agregación como SUM, COUNT, AVG, etc., para calcular valores resumidos de los datos, como el total de ventas de un producto o el promedio de edad de los empleados.
- Unión de tablas: SELECT se usa en conjunción con la cláusula JOIN para combinar datos de múltiples tablas en una sola consulta, lo que permite realizar consultas complejas que involucran múltiples conjuntos de datos.
- Selección de columnas específicas: SELECT te permite especificar qué columnas deseas recuperar de una tabla, lo que te permite obtener solo la información necesaria para tu consulta.

Limitaciones

- Complejidad de la consulta: A medida que las consultas se vuelven más complejas, puede ser difícil mantener el rendimiento óptimo del sistema, especialmente en bases de datos grandes.
- Seguridad: Si no se configura correctamente, SELECT puede permitir el acceso no autorizado a datos confidenciales.
- Escritura de consultas eficientes: Es importante escribir consultas eficientes para evitar el tiempo de espera prolongado y el consumo excesivo de recursos del servidor.
- Limitaciones de la sintaxis: Algunas bases de datos pueden tener limitaciones en la sintaxis de SELECT que pueden restringir ciertas operaciones o consultas.

FROM

Casos de uso

- Selección de tablas principales: La cláusula FROM se utiliza para indicar la tabla principal de la cual se van a extraer los datos en una consulta. Por ejemplo, en una consulta para recuperar información de los clientes, la tabla de clientes sería especificada en la cláusula FROM.
- Unión de tablas: Además de especificar una tabla principal, la cláusula FROM puede incluir múltiples tablas si necesitas combinar datos de varias fuentes. Esto se hace comúnmente utilizando la cláusula JOIN junto con FROM para unir las tablas apropiadas en la consulta.
- Alias de tabla: Puedes utilizar alias de tabla en la cláusula FROM para abreviar el nombre de la tabla y hacer que la consulta sea más legible. Los alias se asignan utilizando la sintaxis "nombreTabla AS alias".

- Subconsultas: En algunos casos, la cláusula FROM también puede contener subconsultas, lo que significa que puedes utilizar el resultado de una consulta como si fuera una tabla en sí misma.

- Vistas: Puedes referenciar vistas en la cláusula FROM para utilizar los datos definidos por la vista en lugar de acceder directamente a las tablas subyacentes.
- Funciones de tabla: Algunas bases de datos permiten el uso de funciones de tabla en la cláusula FROM, lo que te permite generar conjuntos de datos dinámicamente dentro de una consulta.

Limitaciones

- Rendimiento: Si la consulta implica la combinación de múltiples tablas grandes, puede afectar el rendimiento de la consulta y el tiempo de respuesta de la base de datos.
- Complejidad: Consultas con múltiples tablas pueden volverse complejas, lo que puede dificultar su comprensión y mantenimiento.
- Seguridad: Acceder a tablas sensibles puede plantear problemas de seguridad si no se aplican adecuadamente los controles de acceso.

JOIN

Casos de uso

- Combinación de datos de múltiples tablas: La cláusula JOIN te permite combinar datos de dos o más tablas en una sola consulta. Esto es útil cuando necesitas información de diferentes tablas para obtener un conjunto completo de datos.
- Tipos de JOIN: Hay varios tipos de JOIN que puedes utilizar dependiendo de tus necesidades:

1. INNER JOIN: Devuelve solo las filas que tienen coincidencias en ambas tablas basadas en la condición de unión.
2. LEFT JOIN (o LEFT OUTER JOIN): Devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha. Si no hay coincidencias en la tabla derecha, se devuelve NULL para las columnas de la tabla derecha.
3. RIGHT JOIN (o RIGHT OUTER JOIN): Es lo contrario de LEFT JOIN, devuelve todas las filas de la tabla derecha y las filas coincidentes de la tabla izquierda. Si no hay coincidencias en la tabla izquierda, se devuelve NULL para las columnas de la tabla izquierda.
4. FULL JOIN (o FULL OUTER JOIN): Devuelve todas las filas cuando hay una coincidencia en una de las tablas. Las filas sin coincidencias en ambas tablas tendrán NULL en las columnas de la tabla opuesta.
5. CROSS JOIN: Devuelve el producto cartesiano de las filas de las tablas combinadas, es decir, cada fila de la primera tabla se combina con todas las filas de la segunda tabla.

- Condiciones de unión: Puedes especificar las condiciones de unión utilizando la cláusula ON, donde se establece la relación entre las columnas de las tablas que se están uniendo.
- Alias de tabla: Al igual que con la cláusula FROM, puedes usar alias de tabla para abreviar los nombres de las tablas en la cláusula JOIN, lo que hace que la consulta sea más legible.

- Uniones múltiples: Puedes combinar más de dos tablas utilizando múltiples cláusulas JOIN en una sola consulta.

Limitaciones

- Rendimiento: Al unir grandes conjuntos de datos, el rendimiento de la consulta puede verse afectado, especialmente si las columnas de unión no están indexadas correctamente.
- Complejidad: Consultas con múltiples uniones pueden volverse complicadas y difíciles de mantener, especialmente si hay múltiples condiciones de unión.
- Ambigüedad: Si no se especifican correctamente las condiciones de unión, puede resultar en resultados incorrectos o ambiguos.

WHERE

Casos de uso

- Filtrado de filas: La cláusula WHERE te permite especificar una condición que las filas deben cumplir para ser incluidas en el resultado de la consulta. Por ejemplo, puedes usar WHERE para recuperar solo los registros de clientes que residan en una ciudad específica.
- Operadores de comparación: Puedes utilizar una variedad de operadores de comparación en la cláusula WHERE, como "=", "<>", "<", ">", "<=", ">=", para comparar valores en columnas con valores específicos o con otras columnas.
- Operadores lógicos: Puedes combinar múltiples condiciones utilizando operadores lógicos como AND, OR, y NOT. Esto te permite construir condiciones más complejas para filtrar los datos según tus necesidades.
- Operadores de texto: Para comparar cadenas de texto, puedes utilizar operadores como LIKE para buscar patrones, o IN para verificar si un valor está contenido en una lista de valores.
- Funciones: Puedes utilizar funciones en la cláusula WHERE para realizar operaciones más complejas en los datos antes de aplicar el filtro. Por ejemplo, puedes utilizar funciones de fecha y hora para filtrar registros por fecha.
- Subconsultas: Puedes utilizar subconsultas en la cláusula WHERE para filtrar datos basados en los resultados de otra consulta.

Limitaciones

- Rendimiento: Si la condición WHERE no se optimiza adecuadamente, puede afectar negativamente el rendimiento de la consulta, especialmente en grandes conjuntos de datos.
- Complejidad: A medida que se agregan más condiciones a la cláusula WHERE, la consulta puede volverse más compleja y difícil de entender y mantener.
- Índices: Es importante tener en cuenta que las condiciones en la cláusula WHERE pueden no aprovechar índices existentes si no se escriben correctamente.

HAVING

Casos de uso

- Filtrado de resultados agregados: La cláusula HAVING te permite filtrar los resultados de una consulta después de que se hayan aplicado funciones de agregación utilizando GROUP BY. Esto significa que puedes especificar condiciones para restringir qué grupos se incluyen en el resultado final.
- Condiciones de agregación: Puedes usar la cláusula HAVING para especificar condiciones basadas en los resultados de funciones de agregación, como la suma total de un grupo, el número de elementos en un grupo, etc.
- Operadores de comparación: Al igual que con la cláusula WHERE, puedes utilizar una variedad de operadores de comparación en la cláusula HAVING, como "=", "<>", "<", ">", "<=", ">=", para comparar valores agregados con valores específicos o con otras columnas.
- Operadores lógicos: Puedes combinar múltiples condiciones utilizando operadores lógicos como AND, OR, y NOT en la cláusula HAVING para construir condiciones más complejas para filtrar los resultados agregados.
- No se puede utilizar sin GROUP BY: Es importante tener en cuenta que la cláusula HAVING solo se puede utilizar en consultas que incluyan la cláusula GROUP BY. Esto se debe a que HAVING se aplica a grupos de filas definidos por la cláusula GROUP BY.

Limitaciones

- Limitado a consultas con GROUP BY: Como se mencionó anteriormente, la cláusula HAVING solo se puede utilizar en consultas que incluyan la cláusula GROUP BY, lo que puede limitar su uso en ciertos casos.
- Rendimiento: Al igual que con la cláusula WHERE, es importante optimizar las consultas que utilizan la cláusula HAVING para evitar impactos en el rendimiento, especialmente en grandes conjuntos de datos.

Correlacionadas

Casos de uso

- Filtrado basado en condiciones de la fila actual: Las subconsultas correlacionadas son útiles cuando necesitas filtrar el resultado de la consulta externa basándote en valores específicos de la fila actual de esa consulta externa.
- Cálculos basados en la fila actual: También puedes usar subconsultas correlacionadas para realizar cálculos que dependan de los valores de la fila actual de la consulta externa.
- Referencia de columnas de la consulta externa: En una subconsulta correlacionada, puedes hacer referencia a columnas de las tablas de la consulta externa, lo que te permite comparar o filtrar según los valores de esas columnas.
- Ejecución repetida: Es importante tener en cuenta que las subconsultas correlacionadas se ejecutan una vez por cada fila en el resultado de la consulta externa. Por lo tanto, pueden ser menos eficientes que otras formas de consultas si no se escriben cuidadosamente.

Limitaciones

- Limitaciones de rendimiento: Debido a su naturaleza repetitiva, las subconsultas correlacionadas pueden tener un impacto significativo en el rendimiento, especialmente

en grandes conjuntos de datos. Es importante optimizarlas para mejorar el rendimiento de las consultas.

Ejemplo

```
SELECT Nombre, Departamento, Salario
FROM Empleados e
WHERE Salario > (SELECT AVG(Salario)
    FROM Salarios s
    WHERE s.Departamento = e.Departamento);
```