SQL: Subconsultas y Otros Predicados

Juannie

02-06-2024

Consultas Anidadas o Subconsultas.

Al escribir una consulta, a veces hay que expresar una condición que hace referencia a alguna tabla que, a su vez, se debe calcular. Las consultas empleadas para calcular esas tablas son subconsultas y aparecen como parte de las consultas principales.

Una subconsulta es una consulta incluida en una cláusula **WHERE** o **HAVING** de otra consulta. En algunas ocasiones, para expresar ciertas condiciones no hay más remedio que obtener el valor que buscamos como resultado de una consulta.

```
SELECT m.nombrem
FROM marineros AS m
WHERE m.categoría = (
    SELECT MAX(m.categoría)
    FROM marineros AS m
);
```

En este ejemplo, queremos ver los nombres de los marineros que tienen la categoría máxima, y para ello, debemos primero averiguar cuál es esa categoría máxima (usamos la función de agregación **MAX** que veremos luego).

Agregaciones u Operadores de Agregación

Además de recuperar datos, a menudo se desea llevar a cabo algún cálculo o resumen en las consultas. SQL permite el empleo de expresiones aritméticas, pero además permite el uso de operadores de agregación, tales como `MIN` o `SUM`, que representan una ampliación significativa del Álgebra Relacional. SQL soporta cinco operadores de agregación, que se pueden aplicar a cualquier columna de una relación dada, a saber:

- **COUNT** ([**DISTINCT**] columna): El número de valores (únicos) de la columna en cuestión.
- **SUM** ([**DISTINCT**] columna): Suma de todos los valores (únicos) de la columna en cuestión.
- **AVG** ([**DISTINCT**] columna): Promedio de todos los valores (únicos) de la columna en cuestión.
- **MAX** (columna): El valor máximo de la columna en cuestión.
- **MIN** (columna): El valor mínimo de la columna en cuestión.

Análisis del funcionamiento del operador de agregación `COUNT`

La cláusula `COUNT` en SQL es una función de agregación que se utiliza para contar el número de filas en un conjunto de resultados. Es una de las funciones de agregación más comunes y puede ser usada en varias formas para contar el total de filas, filas distintas, o filas con valores específicos.

Formas de Usar `COUNT`

1. **COUNT(*)**:
   - Cuenta todas las filas en el conjunto de resultados, sin importar si tienen valores `NULL` en alguna de las columnas.
   - Ejemplo:

     ```
     SELECT COUNT(*) AS total_filas
     FROM marineros;
     ```

     Este ejemplo cuenta todas las filas en la tabla `marineros`.

2. **COUNT(columna)**:
   - Cuenta todas las filas en las que la columna especificada no es `NULL`.
   - Ejemplo:

     ```
     SELECT COUNT(edad) AS total_edades
     FROM marineros;
     ```

     Este ejemplo cuenta todas las filas donde la columna `edad` no es `NULL` en la tabla `marineros`.

3. **COUNT(DISTINCT columna)**:
   - Cuenta el número de valores distintos en la columna especificada.
   - Ejemplo:

     ```
     SELECT COUNT(DISTINCT categoria) AS total_categorias
     FROM marineros;
     ```

     Este ejemplo cuenta el número de valores distintos en la columna `categoria` de la tabla `marineros`.

Ejemplos Detallados

```
SELECT COUNT(*) AS total_marineros
FROM marineros;
```

Ejemplo 1: Contar todas las filas en una tabla

- **Descripción**: Esta consulta cuenta el número total de marineros en la tabla `marineros`.

```
SELECT COUNT(edad) AS total_con_edad
FROM marineros;
```

Ejemplo 2: Contar filas con valores no nulos en una columna específica

- **Descripción**: Esta consulta cuenta el número de marineros que tienen un valor no nulo en la columna `edad`.

```
SELECT COUNT(DISTINCT categoria) AS total_categorias_distintas
FROM marineros;
```

Ejemplo 3: Contar valores distintos en una columna

- **Descripción**: Esta consulta cuenta el número de categorías distintas entre los marineros.

```sql
SELECT categoria, COUNT(*) AS total_marineros_por_categoria
FROM marineros
GROUP BY categoria;
```

Ejemplo 4: Contar filas después de agrupar

- **Descripción**: Esta consulta cuenta el número de marineros en cada categoría, agrupando los resultados por la columna `categoria`.

Ejemplo con Datos

Supongamos la siguiente tabla `marineros`:

idm	nombrem	categoria	edad
1	John	1	30
2	Alice	2	25
3	Bob	1	NULL
4	Carol	2	17
5	Dave	3	35
6	Eve	2	NULL

```sql
SELECT COUNT(*) AS total_marineros
FROM marineros;
```

Ejemplo 1: Contar todas las filas

- **Resultado**: 6
- **Descripción**: Cuenta todas las filas en la tabla `marineros`.

```sql
SELECT COUNT(edad) AS total_con_edad
FROM marineros;
```

Ejemplo 2: Contar filas con valores no nulos en una columna

- **Resultado**: 4
- **Descripción**: Cuenta las filas donde `edad` no es `NULL`.

```sql
SELECT COUNT(DISTINCT categoria) AS total_categorias_distintas
FROM marineros;
```

Ejemplo 3: Contar valores distintos en una columna

- **Resultado**: 3
- **Descripción**: Cuenta el número de categorías distintas (1, 2, 3).

```sql
SELECT categoria, COUNT(*) AS total_marineros_por_categoria
FROM marineros
GROUP BY categoria;
```

Ejemplo 4: Contar filas después de agrupar

- **Resultado**:

categoria	total_marineros_por_categoria
1	2
2	3
3	1

- **Descripción**: Cuenta el número de marineros en cada categoría, agrupando por `categoria`.

Resumen

- `COUNT(\*)`: Cuenta todas las filas.
- `COUNT(columna)`: Cuenta filas donde la columna no es `NULL`.
- `COUNT(DISTINCT columna)`: Cuenta valores distintos en una columna.
- `COUNT con GROUP BY`: Cuenta filas en cada grupo definido por `GROUP BY`.

La cláusula `COUNT` es muy útil para obtener información agregada sobre los datos en una tabla, permitiendo realizar análisis y reportes basados en conteos y distribuciones.

Hice un análisis sobre este operador en particular porque lo utilizaremos en reiteradas ocasiones.

Ordenación de Datos Resultantes: Cláusula `ORDER BY`

Si se desea que, al hacer una consulta, los datos aparezcan en un orden determinado, es preciso utilizar la cláusula `ORDER BY` en la sentencia `SELECT`, que se escribe de la siguiente forma:

```sql
SELECT columnas
FROM tablas
WHERE condiciones
ORDER BY
    columna_a_ordenar [ASC | DESC],
    columna_a_ordenar [ASC | DESC],
    ...;
```

`ASC` es la opción predeterminada y no hace falta ponerla. Para obtener la tablas en orden descendente se utiliza `DESC`.

La cláusula `GROUP BY` y `HAVING`

Hasta ahora hemos aplicado las operaciones de agregación a todas las filas de la relación. Pero a menudo se desea aplicar operaciones de agregación a cada uno de los grupos de filas de una relación, donde el número de grupos depende del ejemplar de esa relación (es decir, no se conoce con antelación). Por ejemplo, consideremos la siguiente consulta:

Averiguar la edad del marinero más joven de cada categoría.

Si sabemos que las categorías son enteros que van del 1 al 10, se pueden escribir diez consultas de esta forma:

```sql
SELECT MIN(m.edad)
FROM marineros AS m
WHERE m.categoría = i

-- Donde i = 1, 2, 3, ..., 10.
```

Escribir estas diez consultas puede resultar tedioso, sin contar que quizás no sepamos cuantas categorías hay.

La cláusula `GROUP BY` permite escribir consultas como esta de manera mucho más simple:

```sql
SELECT m.categoría, MIN(m.edad)
FROM marineros AS m
GROUP BY m.categoría;
```

Claro, vamos a desglosar y explicar detalladamente cómo funciona la consulta SQL que has proporcionado:

Paso a Paso:

1. **Selección de Columnas (`SELECT m.categoria, MIN(m.edad)`)**:
   - `m.categoria`: Selecciona la columna `categoria` de la tabla `marineros`.
   - `MIN(m.edad)`: Aplica la función de agregación `MIN` a la columna `edad` para encontrar la edad mínima dentro de cada grupo.
2. **Origen de Datos (`FROM marineros AS m`)**:
   - La consulta obtiene los datos de la tabla `marineros`, a la que se le ha asignado un alias `m` para simplificar las referencias a sus columnas.
3. **Agrupación de Datos (`GROUP BY m.categoria`)**:
   - `GROUP BY m.categoria`: Agrupa las filas de la tabla `marineros` por la columna `categoria`. Esto significa que todas las filas con el mismo valor en la columna `categoria` se agruparán juntas.

Detalle del Funcionamiento:

1. **Agrupación (`GROUP BY m.categoria`)**:
   - Las filas de la tabla `marineros` se agrupan en conjuntos donde cada conjunto contiene todas las filas con un valor particular de `categoria`.
   - Por ejemplo, si hay tres categorías distintas (1, 2 y 3), se crearán tres grupos: uno para cada categoría.
2. **Cálculo de la Edad Mínima (`MIN(m.edad)`)**:
   - Dentro de cada grupo creado por el `GROUP BY`, la función `MIN(m.edad)` se aplica a la columna `edad` para encontrar el valor mínimo de la edad en ese grupo.
   - Esto resulta en una única edad mínima para cada categoría.

Ejemplo con Datos:

Supongamos la siguiente tabla `marineros`:

idm	nombrem	categoria	edad
1	John	1	30
2	Alice	2	25

idm	nombrem	categoria	edad
3	Bob	1	28
4	Carol	2	27
5	Dave	3	35

Aplicación de la Consulta:

1. **Agrupación**:
   - Grupo 1: `categoria` = 1 → (John, Bob)
   - Grupo 2: `categoria` = 2 → (Alice, Carol)
   - Grupo 3: `categoria` = 3 → (Dave)
2. **Cálculo de `MIN(m.edad)` en cada grupo**:
   - Grupo 1: `MIN(m.edad)` = `MIN(30, 28)` = 28
   - Grupo 2: `MIN(m.edad)` = `MIN(25, 27)` = 25
   - Grupo 3: `MIN(m.edad)` = 35 (solo un valor)

Resultado Final:

categoria	MIN(m.edad)
1	28
2	25
3	35

Resumen del Funcionamiento:

- **Agrupación**: Las filas se agrupan por `categoria`.
- **Cálculo de Edad Mínima**: Dentro de cada grupo, se calcula la edad mínima usando `MIN(m.edad)`.
- **Selección y Presentación**: La consulta selecciona la categoría y la edad mínima para cada grupo y las presenta como resultado final.

Esta consulta es útil para resumir datos y encontrar valores agregados (como la edad mínima) dentro de grupos específicos definidos por la columna de categoría.

La forma general del uso de la cláusula `GROUP BY` es:

```
SELECT [DISTINCT] columnas
FROM tablas
[WHERE condiciones]
GROUP BY columnas_según_las_cuales_se_quiere_agrupar
[HAVING condición_sobre_grupo]
[ORDER BY columna_a_ordenar [ASC|DESC], columna2_a_ordenar [ASC|DESC]];
```

Lo que está entre corchetes son cláusulas opcionales.

Las columnas que aparecen en el `SELECT` deben sí o sí estar incluidas en las columnas que se usan para agrupar. El motivo es que cada fila del resultado de la consulta se corresponde con un *grupo*, que es un conjunto de filas que concuerdan con los valores de las *columnas_según_las_cuales_se_quiere_agrupar*.

La cláusula `HAVING` permite especificar condiciones a los agrupamientos realizados con `GROUP BY`; del mismo modo que existe la cláusula `WHERE` para las filas individuales en la sentencia `SELECT`.

Al utilizar la cláusula `HAVING` no se incluyen aquellos grupos que no cumplan una determinada condición. La cláusula `HAVING` siempre va detrás de la cláusula `GROUP BY` y no puede existir sin ésta.

Veamos mediante un ejemplo el uso de estas cláusulas:

> Averiguar la edad del marinero más joven que tiene derecho a voto (es decir, tiene 18 años o más) para cada categoría que tenga, como mínimo, dos marineros con derecho a voto.

```sql
SELECT m.categoría, MIN(m.edad) AS edad_mínima
FROM marineros AS m
WHERE m.edad >= 18
GROUP BY m.categoría
HAVING COUNT (*) > 1;
```

Paso a Paso:

1. **Selección de Columnas (`SELECT m.categoria, MIN(m.edad) AS edad_minima`)**:
   - `m.categoria`: Selecciona la columna `categoria` de la tabla `marineros`.
   - `MIN(m.edad) AS edad_minima`: Aplica la función de agregación `MIN` a la columna `edad` para encontrar la edad mínima dentro de cada grupo y la etiqueta como `edad_minima`.
2. **Origen de Datos (`FROM marineros AS m`)**:
   - La consulta obtiene los datos de la tabla `marineros`, a la que se le ha asignado un alias `m` para simplificar las referencias a sus columnas.
3. **Condición de Filtrado (`WHERE m.edad >= 18`)**:
   - `WHERE m.edad >= 18`: Filtra las filas para incluir solo aquellos marineros cuya edad es 18 o mayor. Esto excluye a los marineros menores de 18 años de la consulta.
4. **Agrupación de Datos (`GROUP BY m.categoria`)**:
   - `GROUP BY m.categoria`: Agrupa las filas de la tabla `marineros` por la columna `categoria`. Esto significa que cualquier cálculo de agregación (como `MIN` o `COUNT`) se realizará por cada categoría individualmente.
5. **Condición de Agrupación (`HAVING COUNT(*) > 1`)**:
   - `HAVING COUNT(*) > 1`: Filtra los grupos resultantes del `GROUP BY` para incluir solo aquellos grupos que tienen más de una fila. En otras palabras, se seleccionan solo las categorías que tienen más de un marinero.

Detalle del Funcionamiento:

1. **Filtrado Inicial (`WHERE m.edad >= 18`)**:
   - Antes de agrupar, la consulta filtra los marineros para incluir solo aquellos cuya edad es 18 o mayor.
2. **Agrupación (`GROUP BY m.categoria`)**:
   - Las filas filtradas se agrupan en conjuntos donde cada conjunto contiene todas las filas con un valor particular de `categoria`.
   - Por ejemplo, si hay tres categorías distintas (1, 2 y 3), se crearán tres grupos: uno para cada categoría.
3. **Cálculo de la Edad Mínima (`MIN(m.edad) AS edad_minima`)**:
   - Dentro de cada grupo creado por el `GROUP BY`, la función `MIN(m.edad)` se aplica a la columna `edad` para encontrar el valor mínimo de la edad en ese grupo.

- Esto resulta en una única edad mínima para cada categoría.

4. **Filtrado de Grupos (`HAVING COUNT(*) > 1`)**:
   - `COUNT(*)` cuenta el número de filas en cada grupo.
   - `HAVING COUNT(*) > 1` filtra los grupos para incluir solo aquellos que tienen más de una fila. Esto excluye a las categorías que tienen solo un marinero.

Ejemplo con Datos:

Supongamos la siguiente tabla `marineros`:

idm	nombrem	categoría	edad
1	John	1	30
2	Alice	2	25
3	Bob	1	28
4	Carol	2	17
5	Dave	3	35
6	Eve	2	20

Aplicación de la Consulta:

1. **Filtrado Inicial (`WHERE m.edad >= 18`)**:
   - Se excluye a Carol (edad 17).
   - Quedan los marineros: John, Alice, Bob, Dave, Eve.
2. **Agrupación (`GROUP BY m.categoria`)**:
   - Grupo 1: `categoria` = 1 → (John, Bob)
   - Grupo 2: `categoria` = 2 → (Alice, Eve)
   - Grupo 3: `categoria` = 3 → (Dave)
3. **Cálculo de `MIN(m.edad)` en cada grupo**:
   - Grupo 1: `MIN(m.edad)` = `MIN(30, 28)` = 28
   - Grupo 2: `MIN(m.edad)` = `MIN(25, 20)` = 20
   - Grupo 3: `MIN(m.edad)` = 35
4. **Filtrado de Grupos (`HAVING COUNT(*) > 1`)**:
   - Grupo 1: tiene 2 marineros (John, Bob)
   - Grupo 2: tiene 2 marineros (Alice, Eve)
   - Grupo 3: tiene 1 marinero (Dave) → Este grupo se excluye

Resultado Final:

categoría	edad_mínima
1	28
2	20

Resumen del Funcionamiento:

- **Filtrado Inicial**: Solo se consideran los marineros mayores de 18 años.
- **Agrupación**: Las filas se agrupan por `categoria`.
- **Cálculo de Edad Mínima**: Dentro de cada grupo, se calcula la edad mínima usando `MIN(m.edad)`.
- **Filtrado de Grupos**: Solo se incluyen los grupos (categorías) que tienen más de un marinero.
- **Selección y Presentación**: La consulta selecciona la categoría y la edad mínima para cada grupo y las presenta como resultado final.

Esta consulta es útil para resumir datos y encontrar valores agregados (como la edad mínima) dentro de grupos específicos que cumplen ciertos criterios (en este caso, teniendo más de un miembro mayor de 18 años).

> **Importante**: Sólo pueden aparecer en la cláusula `HAVING` las columnas que aparecen en la cláusula `GROUP BY`, a menos que aparezca como argumento de algún operador de agregación de la cláusula `HAVING`.

Otros Predicados

Predicado BETWEEN

Para expresar una condición que quiere encontrar un valor entre unos límites concretos podemos usar el predicado **BETWEEN**

Por ejemplo, queremos ver los marineros cuyo rango de edad está entre 20 y 35 años:

```
SELECT m.nombrem
FROM marineros AS m
WHERE m.edad BETWEEN 20 AND 35;
```

Predicado IN

Para comprobar si un elemento coincide con los elementos de una lista utilizaremos **IN**, y para ver si no coinciden **NOT IN**.

Por ejemplo, queremos ver los marineros cuya edad sea 15, 20 y 30 años:

```
SELECT m.nombrem
FROM marineros AS m
WHERE m.edad IN (15, 20, 30);
```

Predicado IS

Para comprobar si un valor es nulo utilizamos **IS NULL**, para averiguar si no es nulo, **IS NOT NULL**.

Como ejemplo, supongamos que la tabla *marineros* tiene la columna *hijos*, en la que se pone la cnatidad de hijos que tiene. Bajo ese supuesto, se puede escribir la consulta pidiendo el nombre de los marineros sin hijos, así:

```
SELECT m.nombrem
FROM marineros AS m
WHERE m.hijos IS NULL;
```

Predicado ALL o ANY/SOME

Estos predicados sirven para saber si una columna cumple con la condición de que todas sus filas (**ALL**) o algunas de sus filas (**ANY/SOME**) satisfacen alguna condición. Se utilizan antes de hacer una subconsulta.

Por ejemplo, escribimos una consulta que liste los idb de los barcos que fueron reservados por **todos** los marineros mayores de 18 años:

```
SELECT DISTINCT r.idb
FROM reservas AS r
WHERE r.idm = ALL (
    SELECT m.idm
    FROM marineros AS m
    WHERE m.edad >= 18
);
```

Esta consulta intenta devolver los idb distintos de la tabla reservas donde r.idm cumple con una condición ALL sobre una subconsulta que selecciona todos los idm de marineros con edad mayor o igual a 18. Sin embargo, esta estructura puede resultar en una tabla vacía debido a una mala interpretación del uso de ALL.

El problema radica en cómo ALL interactúa con la subconsulta. La condición r.idm = ALL (...) será verdadera solo si r.idm es igual a todos los valores devueltos por la subconsulta. Es decir, r.idm debe ser igual a cada idm devuelto por la subconsulta, lo cual es casi imposible, ya que r.idm solo puede tener un valor y no puede coincidir con múltiples valores diferentes a la vez.

Necesitamos una solución que garantice que todos los marineros que han reservado un barco sean mayores de 18 años, no que cada reserva individual se compare con todos los marineros mayores de 18 años:

```
SELECT r.idb
FROM reservas AS r
GROUP BY r.idb
HAVING 18 < ALL (
    SELECT m.edad
    FROM reservas AS r2 INNER JOIN marineros AS m ON r2.idm = m.idm
    WHERE r2.idb = r.idb
);
```

Esta consulta asegura que solo se devuelvan los idb de los barcos que han sido reservados exclusivamente por marineros mayores de 18 años, utilizando la cláusula ALL correctamente.

Para ejemplificar el uso de **ANY/SOME**, podemos pedir que liste los idb de los barcos que fueron reservados por **al menos un** o **algún** marinero mayor de 18 años:

```
SELECT DISTINCT r.idb
FROM reservas AS r
WHERE r.idm = ANY (
    SELECT m.idm
    FROM marineros AS m
    WHERE m.edad >= 18
);
```

Predicado EXIST

Para comprobar si una consulta produce alguna fila de resultado, es decir, si existe un resultado, podemos usar el predicado **EXISTS**. Así mismo, para comprobar si no existe se aplicaría **NOT EXISTS**.

Por ejemplo, para mostrar el idm y nombre de los marineros que reservaron un bote determinado podemos también escribir:

```sql
SELECT m.idm, m.nombrem
FROM marineros AS m
WHERE EXISTS (
    SELECT r.idm
    FROM reservas AS r
    WHERE r.idb = 103 AND r.idm = m.idm
);
```

Nota: El predicado **EXISTS** sólo comprueba la existencia de filas que cumplan alguna condición, por lo que realmente no es necesario seleccionar ningún campo en particular en la consulta anidada. Una convención normalmente utilizada es poner un '1' en la selección, de la siguiente manera:

```sql
SELECT m.idm, m.nombrem
FROM marineros AS m
WHERE EXISTS (
    SELECT 1
    FROM reservas AS r
    WHERE r.idb = 103 AND r.idm = m.idm
);
```

... o también podemos escribir la consulta con un asterisco:

```sql
SELECT m.idm, m.nombrem
FROM marineros AS m
WHERE EXISTS (
    SELECT *
    FROM reservas AS r
    WHERE r.idb = 103 AND r.idm = m.idm
);
```

Estas tres consultas devuelven la misma relación:

idm	nombrem
22	Domínguez
31	Lorca