SQL Avanzado

Gestión y Arquitectura de Datos, Universidad de San Andrés

Si encuentran algún error en el documento o hay alguna duda, mandenmé un mail a rodriguezf@udesa.edu.ar y lo revisamos.

1. Cláusula HAVING

HAVING se usa para filtrar resultados de grupos, a diferencia de WHERE que filtra registros individuales.

```sql
SELECT departamento, COUNT(*) as cantidad_empleados
FROM empleados
GROUP BY departamento
HAVING COUNT(*) > 2;
```

departamento	cantidad_empleados
Ventas	3
IT	4

2. Subconsultas

Las subconsultas son consultas anidadas dentro de otras consultas.

2.1. Subconsulta en WHERE

```sql
SELECT nombre, salario
FROM empleados
WHERE salario > (
    SELECT AVG(salario)
    FROM empleados
);
```

nombre	salario
María	60000
Juan	55000

2.2. Subconsulta en FROM

```sql
SELECT dept_info.departamento,
       dept_info.promedio_salario
FROM (
    SELECT departamento,
           AVG(salario) as promedio_salario
    FROM empleados
    GROUP BY departamento
) as dept_info
WHERE dept_info.promedio_salario > 50000;
```

departamento	promedio_salario
IT	55000
Ventas	52000

2.3. Subconsulta con EXISTS

```sql
SELECT nombre
FROM empleados e
WHERE EXISTS (
    SELECT 1
    FROM ventas v
    WHERE v.id_empleado = e.id
    AND v.monto > 10000
);
```

nombre
Juan
María

3. Funciones de Ventana (Window Functions)

Las funciones de ventana permiten realizar cálculos a través de un conjunto de filas relacionadas con la fila actual.

3.1. RANK()

La función RANK() asigna un número de rango a cada fila dentro de una partición, saltando números cuando hay empates. En este ejemplo, ordenamos por salario descendente y asignamos rangos, donde el salario más alto recibe el rango 1.

```
SELECT
    nombre,
    salario,
    RANK() OVER (ORDER BY salario DESC) as ranking
FROM empleados;
```

nombre	salario	ranking
María	60000	1
Juan	55000	2
Carlos	55000	2
Pedro	45000	4

3.2. PARTITION BY

PARTITION BY divide el conjunto de resultados en grupos o particiones. En este caso, calculamos el promedio de salario para cada departamento por separado, mostrando el resultado en cada fila correspondiente a ese departamento.

```
SELECT
    nombre,
    departamento,
    salario,
    AVG(salario) OVER (
        PARTITION BY departamento
    ) as promedio_dept
FROM empleados;
```

nombre	departamento	salario	promedio_dept
María	IT	60000	55000
Juan	IT	50000	55000
Carlos	Ventas	55000	50000
Pedro	Ventas	45000	50000

3.3. ROW_NUMBER()

ROW_NUMBER() asigna un número secuencial único a cada fila dentro de una partición. A diferencia de RANK(), no hay empates - cada fila recibe un número único. En este ejemplo, numeramos los empleados por departamento según su salario.

```
SELECT
    nombre,
    departamento,
    salario,
    ROW_NUMBER() OVER (
        PARTITION BY departamento
        ORDER BY salario DESC
    ) as num_fila
FROM empleados;
```

nombre	departamento	salario	num_fila
María	IT	60000	1
Juan	IT	50000	2
Carlos	Ventas	55000	1
Pedro	Ventas	45000	2

3.4. LAG() y LEAD()

LAG() y LEAD() permiten acceder a datos de filas anteriores o siguientes respectivamente. LAG() obtiene el valor de la fila anterior, mientras que LEAD() obtiene el valor de la fila siguiente. En este ejemplo, mostramos el salario anterior y siguiente para cada empleado.

```
SELECT
    nombre,
    salario,
    LAG(salario) OVER (ORDER BY salario) as salario_anterior,
    LEAD(salario) OVER (ORDER BY salario) as
   salario_siguiente
FROM empleados;
```

nombre	salario	salario_anterior	salario_siguiente
Pedro	45000	NULL	50000
Juan	50000	45000	55000
Carlos	55000	50000	60000
María	60000	55000	NULL

4. Common Table Expressions (CTE)

Las CTEs proporcionan una forma más legible de escribir subconsultas complejas.

```sql
WITH salarios_dept AS (
    SELECT
        departamento,
        AVG(salario) as promedio_salario
    FROM empleados
    GROUP BY departamento
)
SELECT e.nombre,
       e.salario,
       s.promedio_salario
FROM empleados e
JOIN salarios_dept s
    ON e.departamento = s.departamento
WHERE e.salario > s.promedio_salario;
```

nombre	salario	promedio_salario
María	60000	55000
Carlos	55000	50000

5. Optimización de Consultas

- Usar índices apropiadamente
- Evitar SELECT *
- Limitar resultados con WHERE antes de GROUP BY
- Usar EXPLAIN para analizar el plan de ejecución

- Preferir JOINs sobre subconsultas cuando sea posible
- Usar CTEs para mejorar la legibilidad
- Considerar el uso de índices compuestos para consultas frecuentes