Consultas y Planes de Ejecución

Pérez López Leonardo
Galicia Cobaxin Daniel

4 de junio de 2025

1. Consulta por división relacional

1.1. Código SQL

Esta consulta resuelve el inciso c) de la Práctica 2 implementando la división relacional.

```sql
SELECT distinct Salesorderid
FROM orderDetail AS OD
WHERE NOT EXISTS (SELECT *
          FROM (select productid
                from orderDetail
            where salesorderid=43676) as P
          WHERE NOT EXISTS (SELECT *
                FROM orderDetail AS OD2
                WHERE OD.salesorderid = OD2.salesorderid
                AND OD2.productid = P.productid));
```

Listing 1: consulta SQL

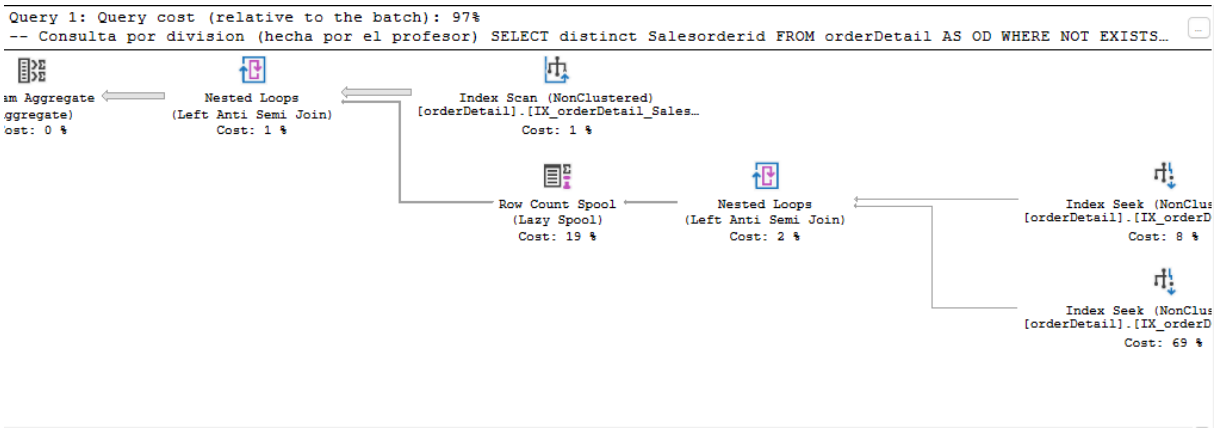

Figura 1: Plan de ejecución 1

¿Qué está haciendo esta consulta?

Está implementando una división relacional: selecciona las órdenes que contienen todos los productos que aparecen en la orden 43676. Se basa en una doble negación (NOT EXISTS ... WHERE NOT EXISTS), es decir, una verificación de subconjuntos.

¿Por qué es tan costosa?

1. Subconsultas correlacionadas:

- El WHERE NOT EXISTS (...) está correlacionado con cada fila de la tabla externa OD.
- Esto obliga a SQL Server a ejecutar la subconsulta por cada fila de orderDetail, lo cual dispara el uso de CPU, disco y memoria.

1. No puede usar bien los índices:

- Aunque tengas índices, el optimizador no puede predecir eficientemente cuántas veces ejecutará la subconsulta, por lo que recurre a Hash Match, Nested Loop, y muchas veces termina en Index Scan o Table Scan.

2. Consulta basada en CTEs y EXCEPT

2.1. Código SQL

Esta consulta resuelve el inciso c) de la Práctica 2 implementando CTEs y EXCEPT.

```
WITH ProductsOrder43676 AS (
    SELECT productid
    FROM orderDetail
    WHERE salesorderid = 43676
),
CandidateOrders AS (
    SELECT salesorderid, productid
    FROM orderDetail
    WHERE salesorderid <> 43676
)
SELECT salesorderid
FROM CandidateOrders
GROUP BY salesorderid
HAVING COUNT(DISTINCT productid) >= (SELECT COUNT(DISTINCT productid)
                                     FROM ProductsOrder43676)
         AND NOT EXISTS (
           SELECT productid
           FROM ProductsOrder43676
           EXCEPT
           SELECT productid
           FROM CandidateOrders c
           WHERE c.salesorderid = CandidateOrders.salesorderid
         );
```

Listing 2: consulta SQL

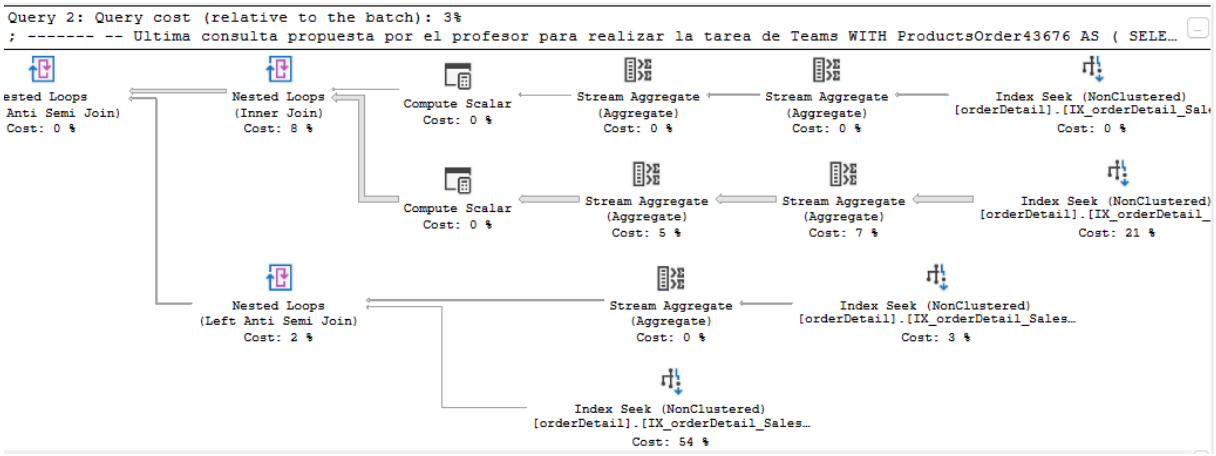

Figura 2: Plan de ejecución 2

¿Por qué esta consulta es mucho más rápida?

- Procesamiento en bloques (set-based):
  Esta consulta trabaja por grupos (GROUP BY y EXCEPT) y no está correlacionada. Eso permite a SQL Server evaluar grandes volúmenes de datos en menos pasos.
- Plan de ejecución más limpio:
  Usa Index Seek, Hash Aggregate, y Merge Join eficientemente.
  Evita subconsultas correlacionadas que obligan a SQL Server a evaluarlas muchas veces.
- Optimización predictiva posible:
  Al trabajar con CTEs, GROUP BY, y EXCEPT, el optimizador puede construir un plan de ejecución más predecible y paralelo, aplicando índices cuando corresponde.

3. Conclusiones

Durante el desarrollo de esta última parte del ejercicio, pude observar de forma muy clara cómo el enfoque que se utilice para resolver una misma consulta puede tener un impacto enorme en el rendimiento del sistema. Al comparar dos consultas que devolvían exactamente el mismo resultado—una utilizando la técnica clásica de división relacional con subconsultas correlacionadas, y otra con CTEs y el operador `EXCEPT`—fue evidente que la eficiencia entre ambas era drásticamente diferente.

La consulta basada en división relacional alcanzó un 97 % de *query cost*, mientras que la otra apenas un 3 %, aun cuando ambas estaban trabajando sobre las mismas tablas e índices. Esto me permitió entender que, aunque ciertos enfoques pueden ser válidos desde el punto de vista lógico, no siempre son prácticos ni eficientes en términos de rendimiento, especialmente cuando se aplican a bases de datos grandes como AdventureWorks.

También aprendí que las subconsultas correlacionadas—aunque útiles para ciertas tareas—tienden a ser más pesadas porque se ejecutan repetidamente por cada fila, lo cual afecta negativamente los tiempos de respuesta. Por el contrario, estructuras como

los CTEs, combinadas con operadores como `GROUP BY` y `EXCEPT`, permiten a SQL Server optimizar mejor el plan de ejecución, haciendo que el procesamiento sea mucho más rápido.