PRÁCTICA#2

Bases de Datos Distribuidas

Descripción breve

Reporte escrito de práctica 2

- Urrutia González Brenda
- Olea García Alan
- Juárez Anguiano Mario Alexis

Equipo 03

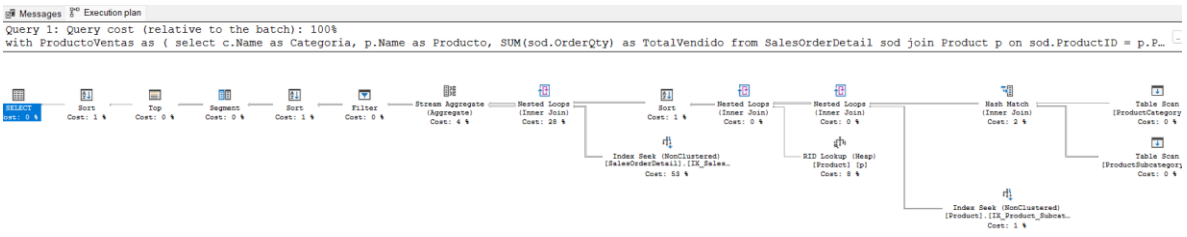
29/05/2025

4.Generar los planes de ejecución de las consultas en la base de datos practicaPE y proponer índices para mejorar el rendimiento de las consultas.

a.Listar el producto más vendido de cada una de las categorías registradas en la base de datos.

```
with ProductoVentas as (
    select
        c.Name as Categoria,
        p.Name as Producto,
        SUM(sod.OrderQty) as TotalVendido
    from SalesOrderDetail sod
    join Product p on sod.ProductID = p.ProductID
    join ProductSubcategory psc on p.ProductSubcategoryID = psc.ProductSubcategoryID
    join ProductCategory c on psc.ProductCategoryID = c.ProductCategoryID
    group by c.Name, p.Name
),
MaxVentasPorCategoria as (
    select
        Categoria,
        max(TotalVendido) as MaxVendido
    from ProductoVentas
    group by Categoria
)
select
    pv.Categoria,
    pv.Producto,
    pv.TotalVendido
from ProductoVentas pv
join MaxVentasPorCategoria mv
    on pv.Categoria = mv.Categoria AND pv.TotalVendido = mv.MaxVendido
order by pv.Categoria;
```

-Plan de ejecución

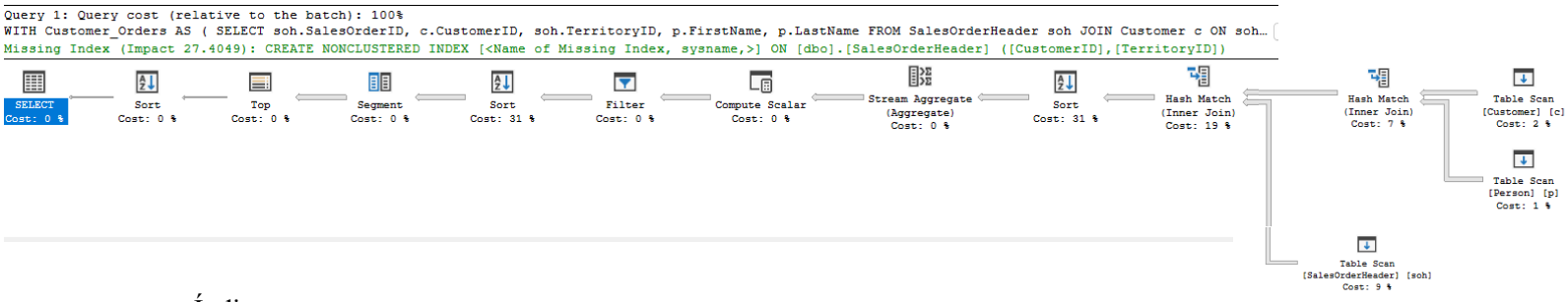


-Índice propuesto

```
CREATE NONCLUSTERED INDEX IX_SalesOrderDetail_ProductID_OrderQty
ON SalesOrderDetail(ProductID, OrderQty);
```

b.Listar el nombre de los clientes con más ordenes por cada uno de los territorios registrados en la base de datos.

```
WITH Customer_Orders AS (
    SELECT
        soh.SalesOrderID,
        c.CustomerID,
        soh.TerritoryID,
        p.FirstName,
        p.LastName
    FROM SalesOrderHeader soh
    JOIN Customer c ON soh.CustomerID = c.CustomerID
    JOIN Person p ON c.PersonID = p.BusinessEntityID
),
OrdenesPorCliente AS (
    SELECT
        TerritoryID,
        CustomerID,
        FirstName,
        LastName,
        COUNT(*) AS Orders
    FROM Customer_Orders
    GROUP BY TerritoryID, CustomerID, FirstName, LastName
),
MaxOrdenesPorTerritorio AS (
    SELECT
        TerritoryID,
        MAX(Orders) AS MaxOrders
    FROM OrdenesPorCliente
    GROUP BY TerritoryID
)
SELECT
    o.TerritoryID,
    o.FirstName,
    o.LastName,
    o.Orders
FROM OrdenesPorCliente o
JOIN MaxOrdenesPorTerritorio m
    ON o.TerritoryID = m.TerritoryID AND o.Orders = m.MaxOrders
ORDER BY o.TerritoryID;
```



-Índices propuestos

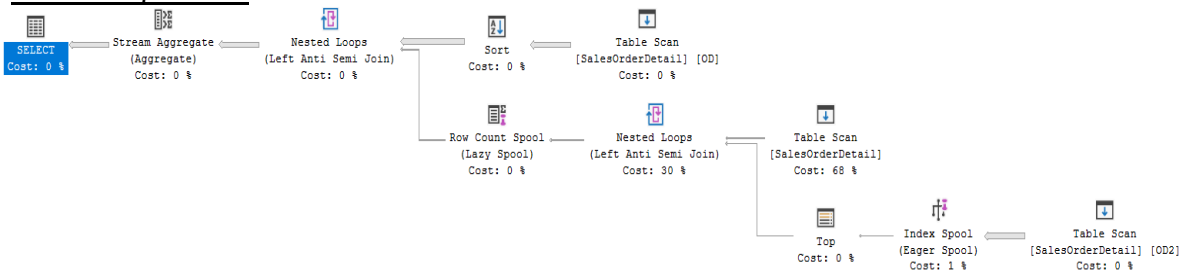
```
CREATE NONCLUSTERED INDEX IX_SalesOrderHeader_Customer_Territory
ON SalesOrderHeader (CustomerID, TerritoryID);

CREATE NONCLUSTERED INDEX IX_Customer_PersonID
ON Customer(PersonID);
```

```
CREATE NONCLUSTERED INDEX IX_Person_EntityID
ON Person(BusinessEntityID);
```

-Listar los datos generales de las ordenes que tengan al menos los mismos productos de la orden con salesorderid = 43676.

```
SELECT DISTINCT Salesorderid
FROM Sales.SalesOrderDetail AS OD
WHERE NOT EXISTS
                        (
                                SELECT *
                                FROM (SELECT productid
                                from Sales.SalesOrderDetail
                                where salesorderid=43676) as P
                                WHERE NOT EXISTS
                                                (
                                                        SELECT *
                                                        FROM Sales.SalesOrderDetail
AS OD2
                                                        WHERE   OD.salesorderid  =
OD2.salesorderid
                                                        AND     (OD2.productid     =
P.productid)
                                                )
                                );
```

-Plan de ejecución



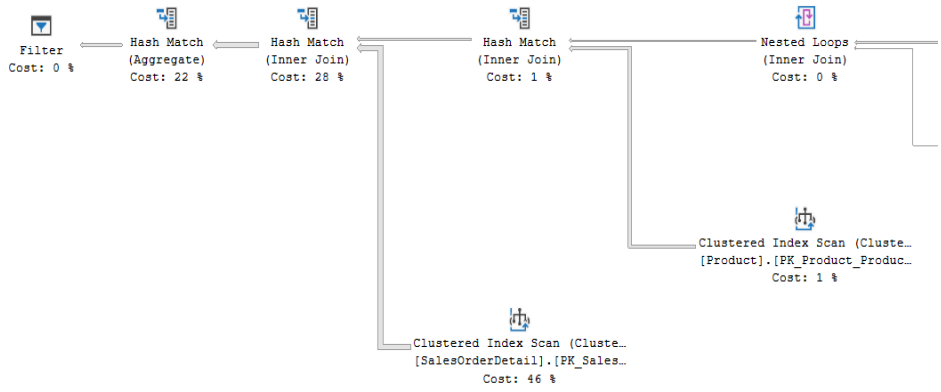
-Índice propuesto

```
CREATE NONCLUSTERED INDEX IX_SalesOrderDetail_ProductID_OrderQty
ON SalesOrderDetail(ProductID, OrderQty);
```

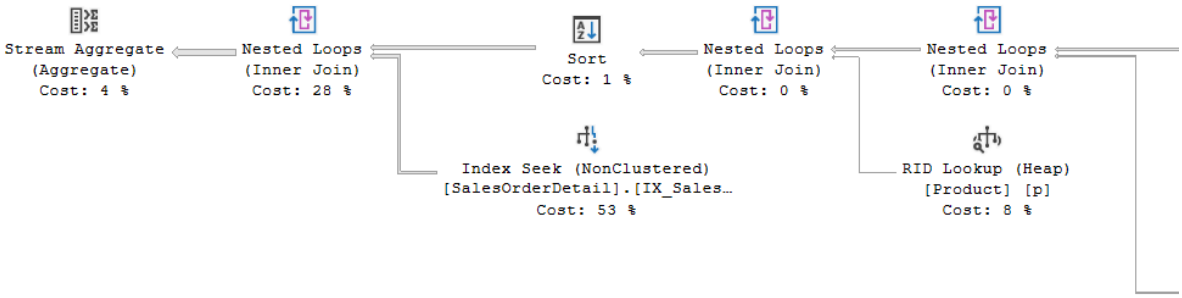
5.Generar los planes de ejecución de las consultas en la base de datos AdventureWorks y comparar con los planes de ejecución del punto 4.

a)

Con la base Adventure:



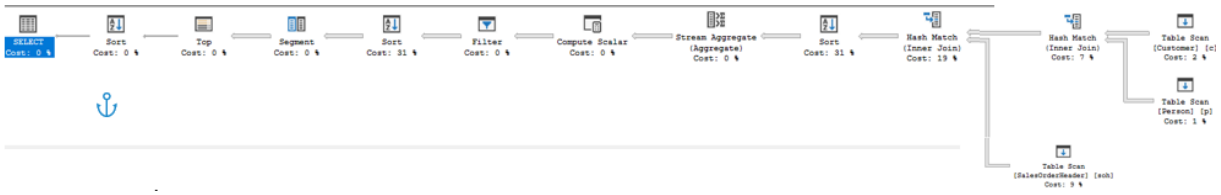
Con la base practicaPE:



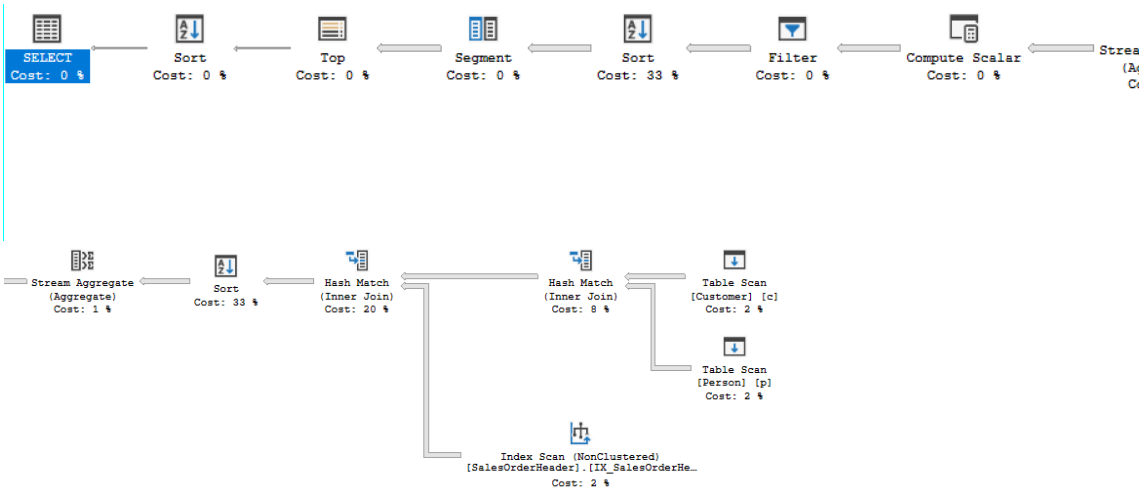
En el plan de ejecución sin índices el uso de hash match indica que SQL Server necesita cargar muchos datos para hacer los joins y agregaciones, probablemente porque no hay un índice adecuado que permita buscar y filtrar de manera eficiente.
En el segundo plan de ejecución el índice nonclustered con columnas incluidas ayuda a reducir la necesidad de leer todas las filas (scan), ya que se puede buscar directamente por ProductID y obtener las columnas necesarias sin ir a la tabla base, de igual manera el uso de Nested Loops en lugar de Hash Match indica que el motor aprovecha índices para hacer joins más eficientes.

b)

Con la base Adventure:



Con la base practicaPE:



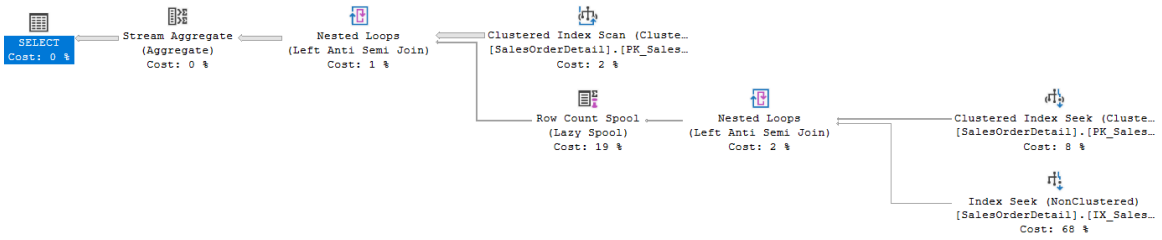
Después de crear el índice IX_SalesOrderHeader_Customer_Territory, la consulta mejoró en los siguientes aspectos clave:

1. Disminución drástica de lecturas
   - Antes del índice: 780 lecturas lógicas en la tabla SalesOrderHeader.
   - Después del índice: solo 88 lecturas.

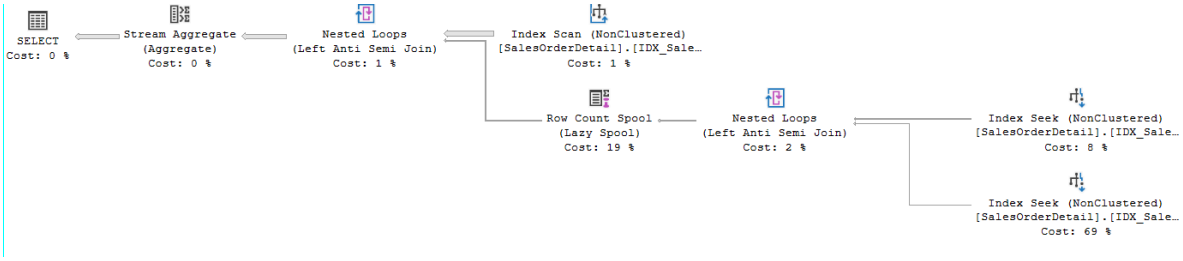
   Lo cual significa que SQL Server ya no lee toda la tabla, sino que accede solo a las filas que le interesan.
2. Uso de Index Scan en lugar de Table Scan
   - Ya no se hace un recorrido completo de la tabla.
   - SQL utiliza el índice como atajo para encontrar más rápido los datos agrupados por CustomerID y TerritoryID.

c)

Con la base Adventure:



Con la base practicaPE:

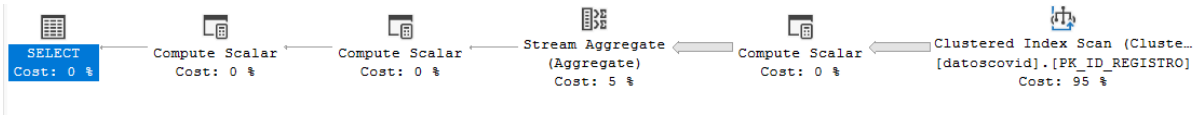


En el plan de ejecución usando la base de Adventure se obliga a escanear más datos y a utilizar múltiples Nested Loops para realizar comparaciones y evitar duplicados; a su vez, el costo relativo de acceso es alto (68 %), ya que el acceso no está optimizado.

En cambio, en el segundo plan de ejecución (base practicaPE), se observa la mejora al contar con un índice nonclustered sobre las columnas ProductID y OrderQty. Esto permite que SQL Server realice Index Seek y Index Scan más específicos, accediendo directamente a los datos relevantes sin recorrer toda la tabla base.

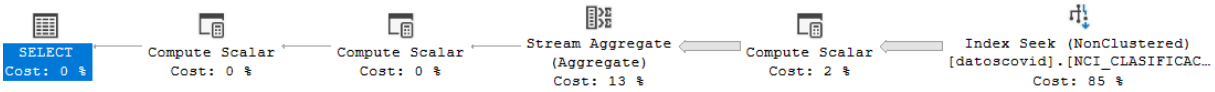
6.Generar los planes de ejecución de las consultas 3, 4 y 5 de la práctica de consultas en la base de datos Covid y proponer índices para mejorar el rendimiento.

Consulta 3. Listar el porcentaje de casos confirmados en cada una de las siguientes morbilidades a nivel nacional: diabetes, obesidad e hipertensión.



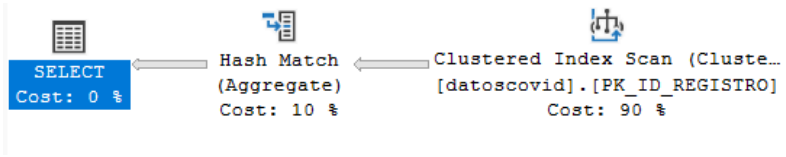
-Índice propuesto

```
CREATE NONCLUSTERED INDEX NCI_CLASIFICACION_FINAL
ON datoscovid (CLASIFICACION_FINAL)
INCLUDE (DIABETES, HIPERTENSION, OBESIDAD);
```



Consulta 4. Listar los municipios que no tengan casos confirmados en todas las morbilidades: hipertensión, obesidad, diabetes, tabaquismo.

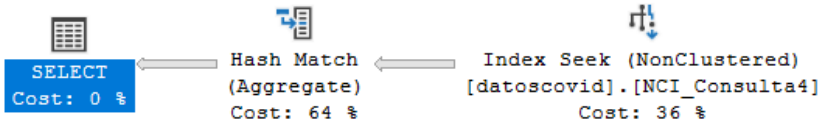
-Plan de ejecución sin índices



-Índice propuesto

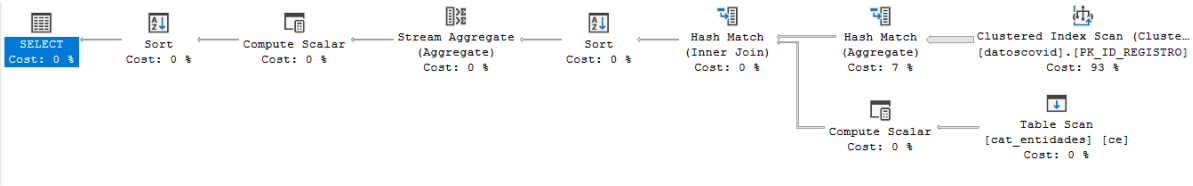
```
CREATE NONCLUSTERED INDEX NCI_Consulta4
ON datoscovid (CLASIFICACION_FINAL, HIPERTENSION, OBESIDAD, DIABETES, TABAQUISMO)
INCLUDE (ENTIDAD_RES, MUNICIPIO_RES);
```

-Plan de ejecución con índices



Consulta 5. Listar los estados con más casos recuperados con neumonía.

-Plan de ejecución sin índices



-Índice propuesto

```
CREATE NONCLUSTERED INDEX NCI_Consulta5
ON datoscovid (FECHA_DEF, CLASIFICACION_FINAL, NEUMONIA)
INCLUDE (ENTIDAD_RES);
```

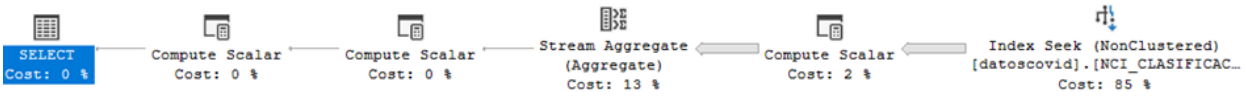
-Plan de ejecución con índices

7.Comparar los planes de ejecución del punto 6 con los planes de ejecución de otro equipo.

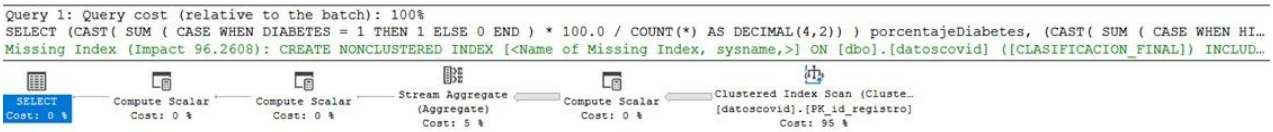
La comparación se hizo con el equipo 4, los planes de ejecución son los siguientes:

-Consulta 3

Plan de ejecución de nuestro equipo:



Plan de ejecución equipo 4:

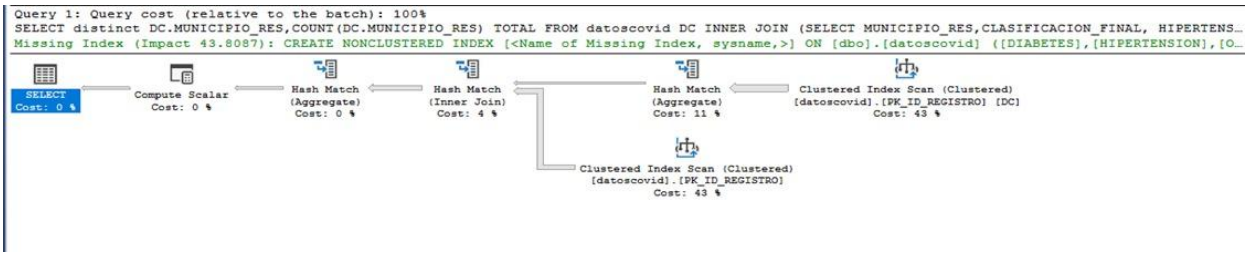


Nuestro plan muestra un *Index Seek* en el índice no agrupado (en la columna CLASIFICACION_FINAL), que es más eficiente para filtrar, porque busca directamente los valores que cumplen la condición WHERE CLASIFICACION_FINAL IN (1,2,3), mientras que **el plan del otro equipo** usa un *Clustered Index Scan* sobre la clave primaria (PK), que escanea toda la tabla, lo cual es más costoso (95%)

-Consulta 4

Plan de ejecución de nuestro equipo:

Plan de ejecución equipo 4:

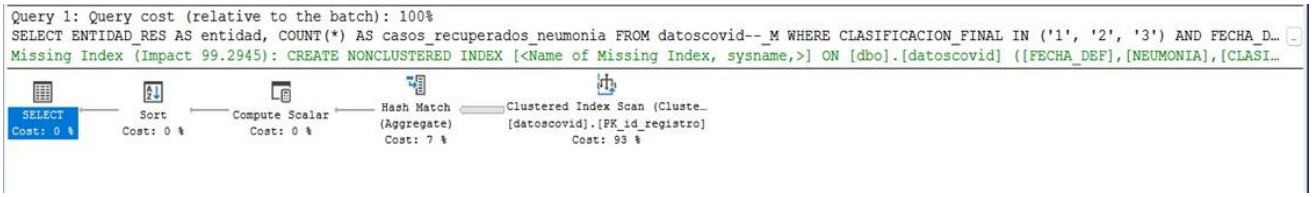


Nuestro plan es más eficiente en la parte de acceso a datos, usando un Index Seek en lugar de Scan. El otro equipo usa Clustered Index Scan, que es más costoso y menos eficiente en tablas grandes.

-Consulta 5

Plan de ejecución de nuestro equipo:

Plan de ejecución equipo 4:



Nuestro plan es mucho más eficiente en la búsqueda porque usa un **Index Seek** en un índice no agrupado, mientras que el otro equipo hace un scan total con costo alto. El otro equipo tiene un cuello de botella en el scan total, lo que puede afectar el rendimiento en tablas grandes.

Conclusiones

Para entender como funcionaban los planes de ejecución, primero tuvimos que investigar más a fondo sobre los distintos “join” y está fue nuestro resumen:

El nested Loops Join se utiliza cuando una de las tablas es pequeña o altamente selectiva, y la otra tiene un índice adecuado. Es ideal para realizar búsquedas rápidas por cada fila de la tabla externa

El merge Join es eficiente cuando ambas tablas son grandes y están ordenadas por la columna de unión. Aprovecha el orden para hacer un emparejamiento secuencial sin escanear de más,

El Hash Join se aplica cuando las tablas son grandes y no están indexadas ni ordenadas. SQL Server construye una tabla hash en memoria con una de las entradas y compara con la otra, lo que permite procesar grandes volúmenes de datos sin depender de índices.

Las explicaciones de nuestros índices son las siguientes:

El índice en ProductID del **inciso a** uso porque permite que la búsqueda de registros en SalesOrderDetail relacionados con cada producto sea rápida y no se tenga que hacer un escaneo completo y OrderQty es la columna que se suma en la consulta para obtener el total vendido, por lo que, al incluir esta columna en el índice, el motor puede satisfacer la consulta directamente desde el índice sin necesidad de leer los datos de la tabla base (heap o clustered index), por otra parte El índice IX_SalesOrderHeader_Customer_Territory del **inciso b** fue usado porque la consulta hace esto:

1. Agrupa los datos por CustomerID y TerritoryID para contar cuántas órdenes tiene cada cliente en cada territorio.
2. Filtra para obtener el cliente con más órdenes por territorio.
3. Usa estas columnas en el JOIN y en el GROUP BY.

El índice creado coincide exactamente con esas columnas, por eso SQL Server lo elige para:

- Buscar más rápido los datos agrupados.
- Evitar ordenar toda la tabla desde cero.
- Reducir la cantidad de páginas leídas, lo cual acelera todo el proceso.

Por otra parte, en el **inciso c** se agregó el índice porque en el plan de ejecución “SalesOrderID” va primero porque la consulta filtra y agrupa principalmente por pedido y “ProductID” está en la clave porque se usa para verificar productos dentro de cada pedido.

Por lo que, este índice permite buscar rápidamente los productos de un pedido específico y facilitar el acceso eficiente a las filas cuando se hacen las subconsultas con NOT EXISTS.

Por último, los índices agregados a la base de datos de “covidhistorico” fueron usados por:

-Consulta 3

- CLASIFICACION_FINAL es la columna usada para filtrar, debe ir en la clave para hacer un **index seek** eficiente.
- Las columnas que se usan en el cálculo (morbilidades) están en INCLUDE para evitar acceder a la tabla base (evita key lookup).
- La consulta queda cubierta y el acceso es rápido.

-Consulta 4

- La clave del índice cubre todas las condiciones del WHERE para acelerar el filtrado.
- Las columnas del GROUP BY están en INCLUDE para cubrir la consulta y evitar lecturas adicionales.

-Consulta 5

- La clave del índice incluye las columnas usadas en filtro para hacer un seek eficiente.
- ENTIDAD_RES está en INCLUDE porque se usa para el join y agrupamiento.
- Reduce lecturas y mejora la velocidad del filtrado y agrupamiento.