Práctica 2
Bases de Datos Distribuidas

Equipo 7
Pérez López Leonardo
Galicia Cobaxin Daniel

30 de mayo de 2025

1. Copias correspondientes a la base de datos

```
-- copia de la tabla Sales Order Header
SELECT * INTO orderHeader
FROM AdventureWorks2019.Sales.SalesOrderHeader;

-- copia de la tabla Sales Order Detail
SELECT * INTO orderDetail
FROM AdventureWorks2019.Sales.SalesOrderDetail;

-- copia de la tabla Customer
SELECT * INTO customer
FROM AdventureWorks2019.Sales.Customer;

-- copia de la tabla Customer
SELECT * INTO Salesterritoty
FROM AdventureWorks2019.Sales.SalesTerritory;

-- copia de la tabla Product
SELECT * INTO Product
FROM AdventureWorks2019.Production.Product;

-- copia de la tabla ProductCategory
SELECT * INTO ProductCategory
FROM AdventureWorks2019.Production.ProductCategory;

-- copia de la tabla ProductSubcategory
SELECT * INTO ProductSubcategory
FROM AdventureWorks2019.Production.ProductSubcategory;

-- copia de la tabla Person
SELECT BusinessEntityID, PersonType, NameStyle, Title,
FirstName, MiddleName, LastName, Suffix, EmailPromotion,
rowguid, ModifiedDate
INTO Person
FROM AdventureWorks2019.Person.Person;
```

2. Consulta a: Listar el producto más vendido de cada una de las categorias registradas en la base de datos

2.1. Consulta a.1

```sql
WITH VentasProducto AS (
    SELECT
        pc.Name AS Categoria,
        p.ProductID,
        p.Name AS Producto,
        SUM(od.OrderQty) AS CantidadVendida
    FROM orderDetail od
    JOIN Product p ON od.ProductID = p.ProductID
    JOIN ProductSubcategory ps ON p.ProductSubcategoryID = ps.ProductSubcategoryID
    JOIN ProductCategory pc ON ps.ProductCategoryID = pc.ProductCategoryID
    GROUP BY pc.Name, p.ProductID, p.Name
),
ProductoMaximo AS (
    SELECT
        Categoria,
        MAX(CantidadVendida) AS MaxCantidad
    FROM VentasProducto
    GROUP BY Categoria
)
SELECT vp.Categoria, vp.Producto, vp.CantidadVendida
FROM VentasProducto vp
JOIN ProductoMaximo pm
  ON vp.Categoria = pm.Categoria AND vp.CantidadVendida = pm.MaxCantidad
ORDER BY vp.Categoria;
```

2.2. Consulta a.2

```sql
WITH ProductosVendidos AS (
    SELECT
        od.ProductID,
        SUM(od.OrderQty) AS CantidadVendida
    FROM orderDetail od
    GROUP BY od.ProductID
),
ProductoConCategoria AS (
    SELECT
        pv.ProductID,
        pv.CantidadVendida,
        p.Name AS Producto,
        pc.Name AS Categoria
    FROM ProductosVendidos pv
    JOIN Product p ON pv.ProductID = p.ProductID
    JOIN ProductSubcategory ps ON p.ProductSubcategoryID = ps.ProductSubcategoryID
    JOIN ProductCategory pc ON ps.ProductCategoryID = pc.ProductCategoryID
),
Ranking AS (
    SELECT *,
           RANK() OVER (PARTITION BY Categoria ORDER BY CantidadVendida DESC) AS rk
    FROM ProductoConCategoria
)
SELECT Categoria, Producto, CantidadVendida
FROM Ranking
WHERE rk = 1
ORDER BY Categoria;
```

3. Consulta b: Listar el nombre de los clientes con más órdenes por cada uno de los territorios registrados en la base de datos

3.1. Consulta b.1

```
WITH OrdenesPorCliente AS (
    SELECT
        st.Name AS Territorio,
        c.CustomerID,
        COUNT(oh.SalesOrderID) AS TotalOrdenes
    FROM orderHeader oh
    JOIN customer c ON oh.CustomerID = c.CustomerID
    JOIN Salesterritoty st ON oh.TerritoryID = st.TerritoryID
    GROUP BY st.Name, c.CustomerID
),
MaximoPorTerritorio AS (
    SELECT
        Territorio,
        MAX(TotalOrdenes) AS MaxOrdenes
    FROM OrdenesPorCliente
    GROUP BY Territorio
)
SELECT
    opc.Territorio,
    c.CustomerID,
    p.FirstName + ' ' + ISNULL(p.MiddleName + ' ', '') + p.LastName AS NombreCompleto,
    opc.TotalOrdenes
FROM OrdenesPorCliente opc
JOIN MaximoPorTerritorio mpt
  ON opc.Territorio = mpt.Territorio AND opc.TotalOrdenes = mpt.MaxOrdenes
JOIN customer c ON opc.CustomerID = c.CustomerID
JOIN Person p ON c.PersonID = p.BusinessEntityID
ORDER BY opc.Territorio;
```

3.2. Consulta b.2

```
WITH OrdenesPorCliente AS (
    SELECT
        c.CustomerID,
        st.Name AS Territorio,
        COUNT(oh.SalesOrderID) AS TotalOrdenes
    FROM orderHeader oh
    JOIN customer c ON oh.CustomerID = c.CustomerID
    JOIN Salesterritoty st ON oh.TerritoryID = st.TerritoryID
    GROUP BY c.CustomerID, st.Name
),
RankingClientes AS (
    SELECT
        opc.CustomerID,
        opc.Territorio,
        opc.TotalOrdenes,
        RANK() OVER (PARTITION BY opc.Territorio ORDER BY opc.TotalOrdenes DESC) AS rk
    FROM OrdenesPorCliente opc
),
TopClientes AS (
    SELECT
        rc.CustomerID,
        rc.Territorio,
        rc.TotalOrdenes
    FROM RankingClientes rc
```

```
    WHERE rk = 1
)
SELECT
    t.Territorio,
    CONCAT(p.FirstName, ' ', ISNULL(p.MiddleName + ' ', ''), p.LastName) AS
        NombreCliente,
    t.TotalOrdenes
FROM TopClientes t
JOIN customer c ON t.CustomerID = c.CustomerID
JOIN Person p ON c.PersonID = p.BusinessEntityID
ORDER BY t.Territorio;
```

3.3. Consulta b.3

```
SELECT
    t.Name AS Territorio,
    CONCAT(p.FirstName, ' ', ISNULL(p.MiddleName + ' ', ''), p.LastName) AS
        NombreCliente,
    x.TotalOrdenes
FROM Salesterritoty t
CROSS APPLY (
    SELECT TOP 1 WITH TIES
        c.CustomerID,
        COUNT(oh.SalesOrderID) AS TotalOrdenes
    FROM orderHeader oh
    JOIN customer c ON oh.CustomerID = c.CustomerID
    WHERE oh.TerritoryID = t.TerritoryID
    GROUP BY c.CustomerID
    ORDER BY COUNT(oh.SalesOrderID) DESC
) x
JOIN customer c ON x.CustomerID = c.CustomerID
JOIN Person p ON c.PersonID = p.BusinessEntityID
ORDER BY t.Name;
```

4. Consulta c: Listar los datos generales de las órdenes que tengan al menos los mismos productos de la orden SalesOrderID = 43676

4.1. Consulta c.1

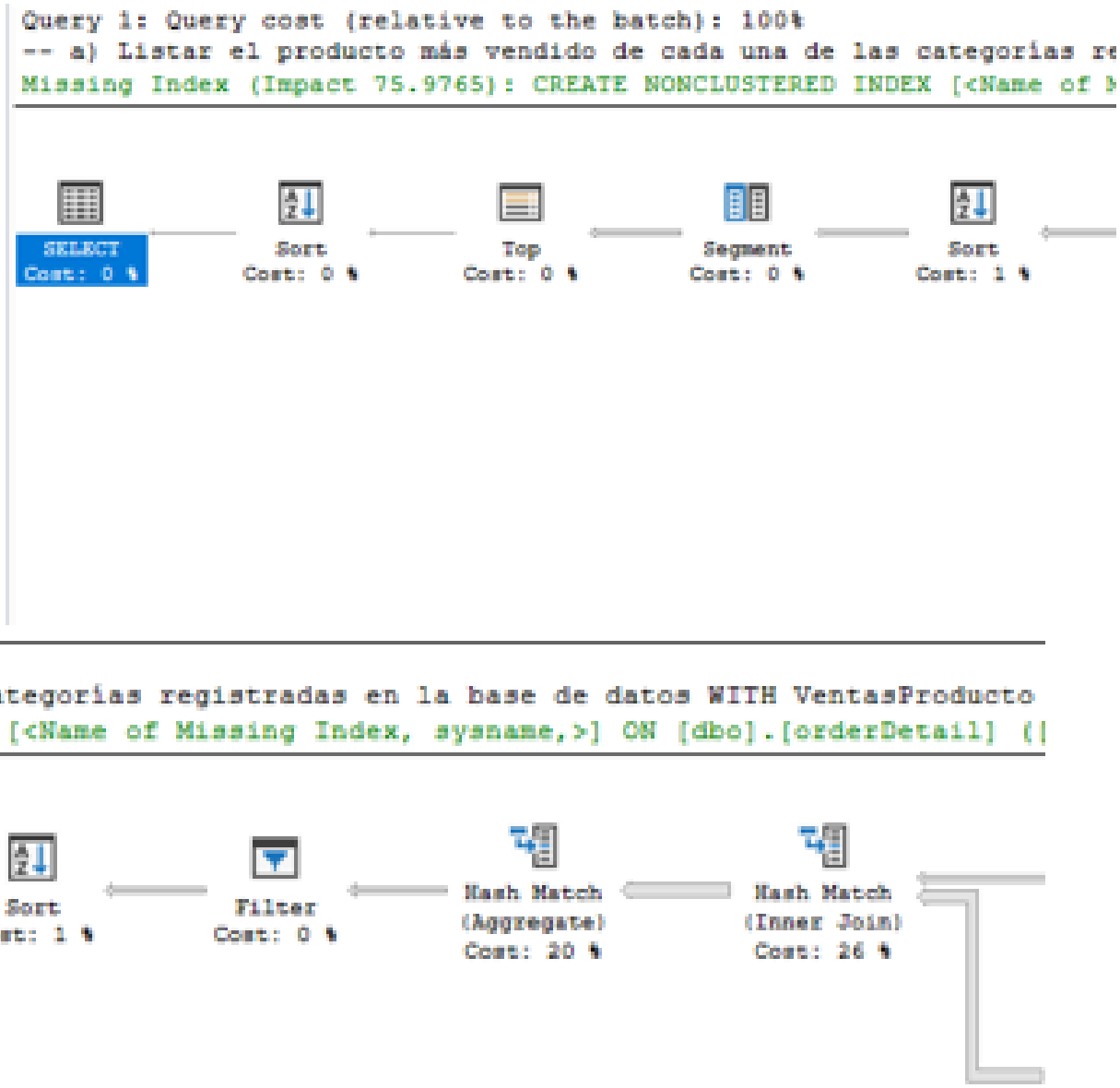
```
WITH ProductosRequeridos AS (
    SELECT ProductID
    FROM orderDetail
    WHERE SalesOrderID = 43676
),
ConteoRequeridos AS (
    SELECT COUNT) AS TotalRequeridosFROM ProductosRequeridos),OrdenesConProductos AS
        (SELECT od.SalesOrderID, COUNT(DISTINCT od.ProductID) AS CoincidenciasFROM
        orderDetail odJOIN ProductosRequeridos pr ON od.ProductID = pr.ProductIDGROUP BY
        od.SalesOrderID),OrdenesValidas AS (SELECT ocp.SalesOrderIDFROM
        OrdenesConProductos ocp, ConteoRequeridos crWHERE ocp.Coincidencias =
        cr.TotalRequeridos)SELECT oh.*FROM orderHeader ohJOIN OrdenesValidas ov ON
        oh.SalesOrderID = ov.SalesOrderIDORDER BY oh.SalesOrderID;SELECT DISTINCT
        salesorderidFROM orderDetail AS ODWHERE NOT EXISTS (SELECT *FROM (SELECT
        productidFROM orderDetailWHERE SalesOrderID=43676) AS PWHERE NOT EXISTS (SELECT
        *FROM orderDetail AS OD2WHERE OD.SalesOrderID=OD2.SalesOrderIDAND
        OD2.ProductID=P.ProductID));
```

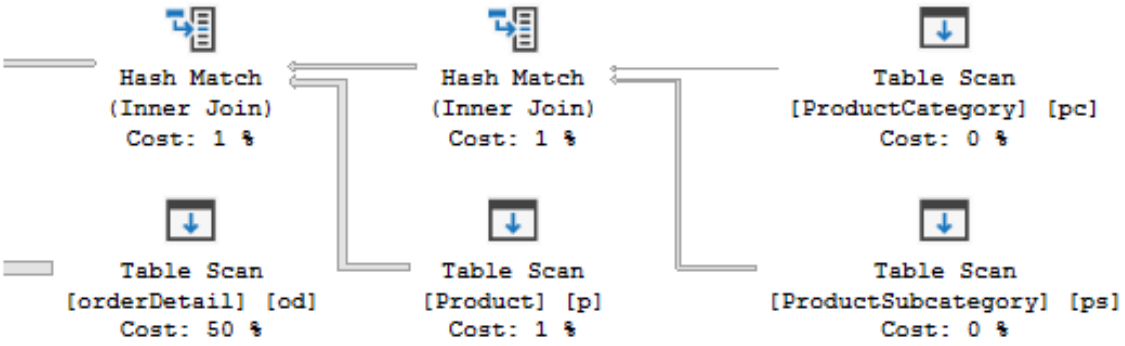
4.2. Consulta c.2

```sql
WITH productos_objetivo AS (
    SELECT ProductID
    FROM orderDetail
    WHERE SalesOrderID = 43676
),
conteo_objetivo AS (
    SELECT COUNT(DISTINCT ProductID) AS totalProductos
    FROM productos_objetivo
),
ordenes_con_productos AS (
    SELECT od.SalesOrderID
    FROM orderDetail od
    JOIN productos_objetivo po ON od.ProductID = po.ProductID
    GROUP BY od.SalesOrderID
    HAVING COUNT(DISTINCT od.ProductID) = (SELECT totalProductos FROM conteo_objetivo)
)
SELECT oh.*
FROM orderHeader oh
JOIN ordenes_con_productos o ON oh.SalesOrderID = o.SalesOrderID
WHERE oh.SalesOrderID <> 43676
ORDER BY oh.OrderDate;
```

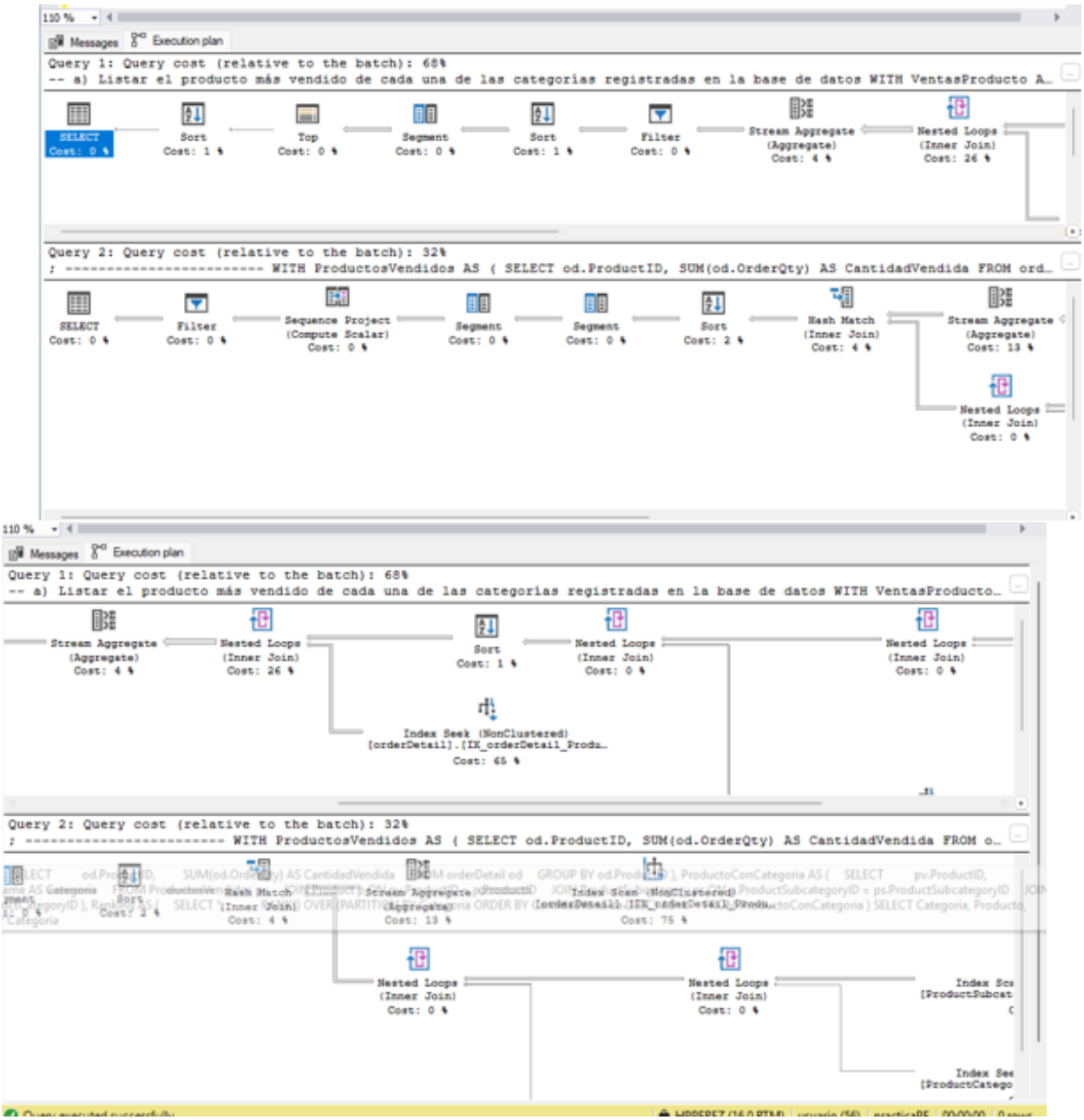
5. Planes de ejecucion

5.1. Consulta 1.1 sin indices

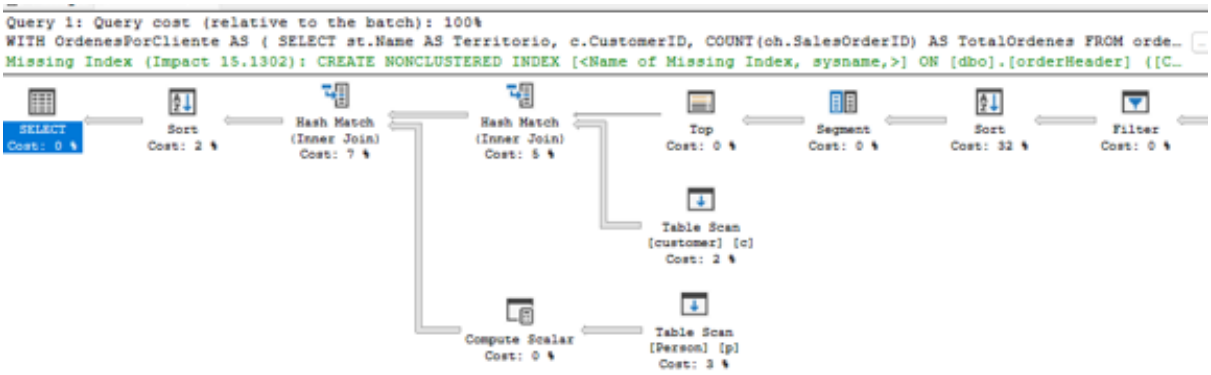


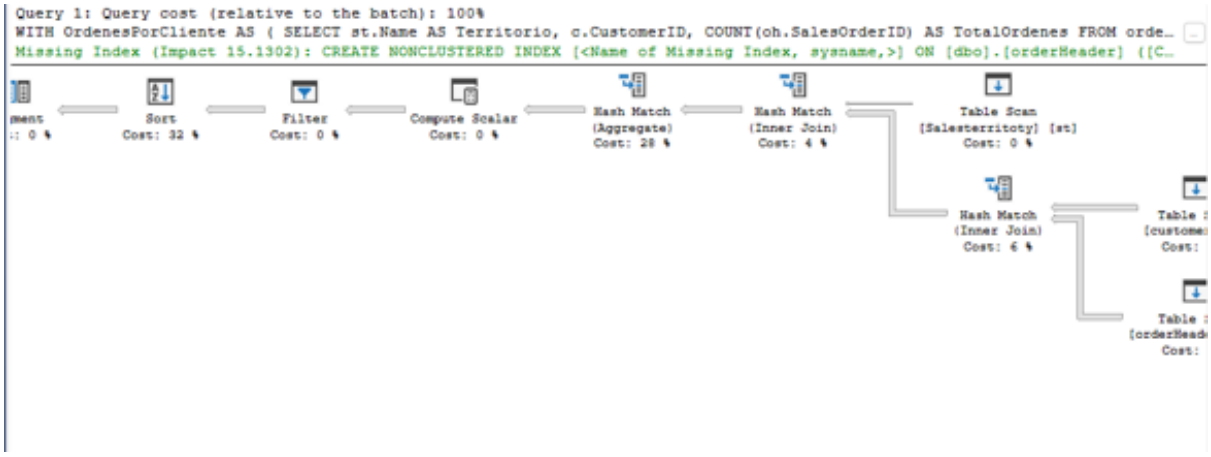


5.2. Consulta 1.1 con indices vs Consulta 1.2 con indices

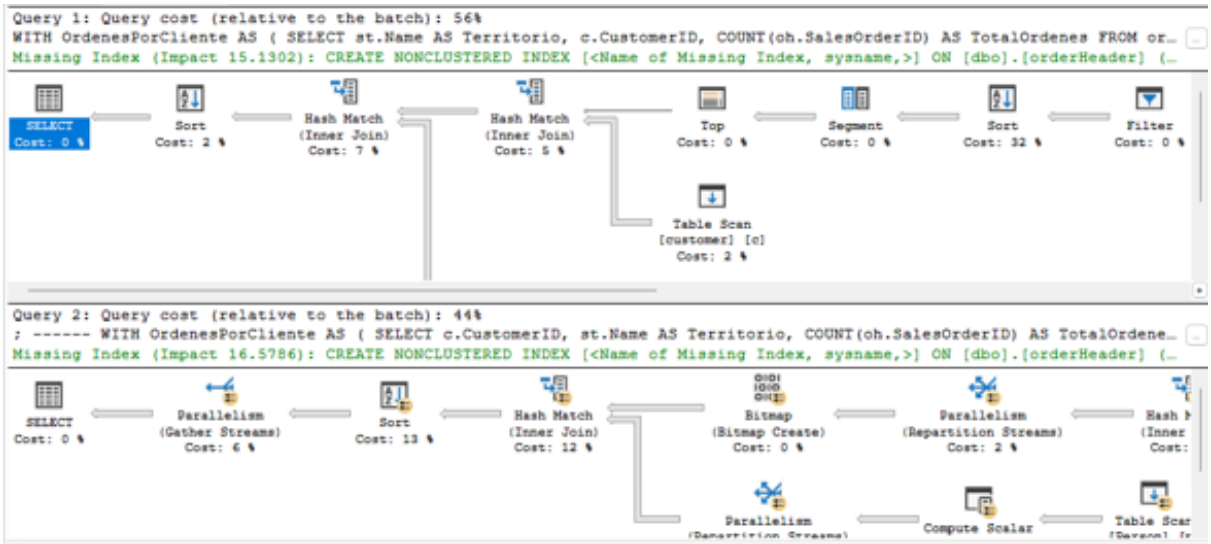


5.3. Consulta 2.1 sin indices

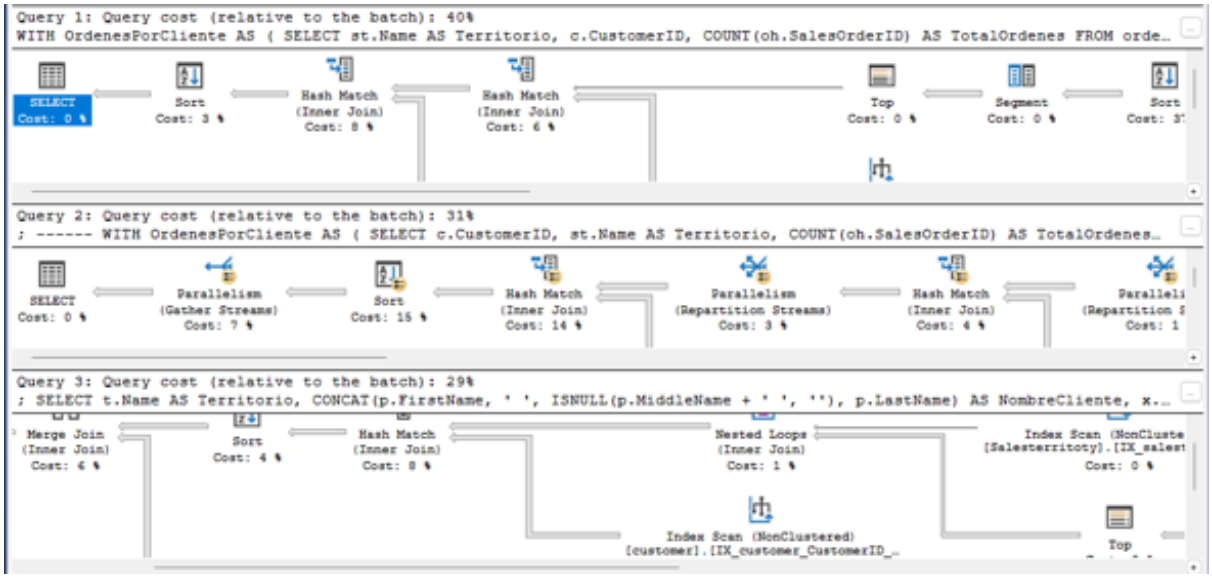




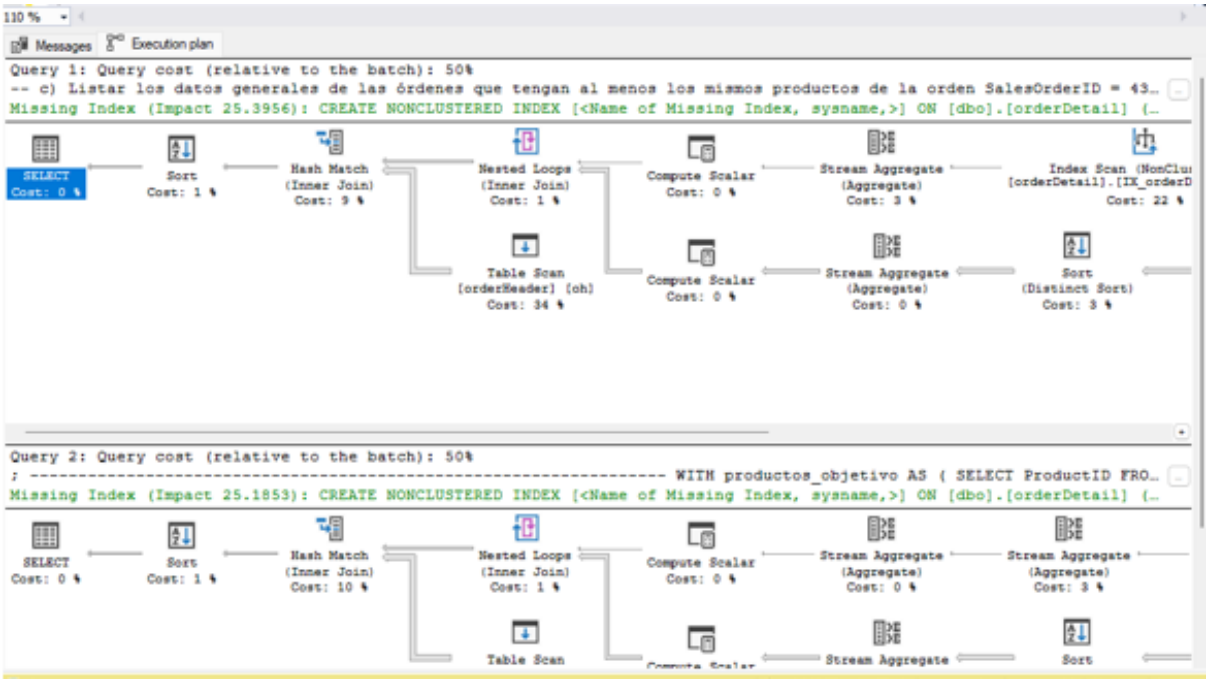
5.4. Consulta 2.1 sin indices vs Consulta 2.2 sin indices



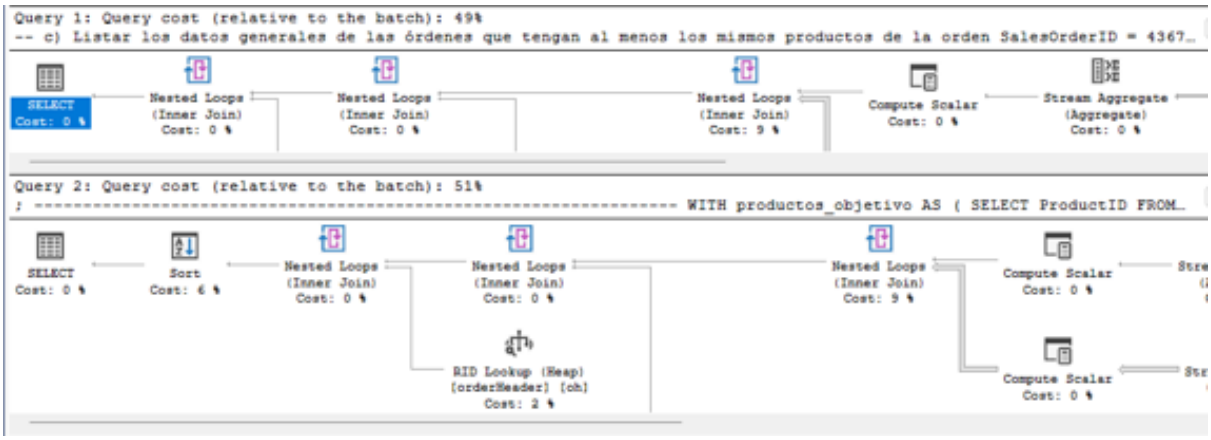
5.5. Consulta 2.1 con indices vs Consulta 2.2 con indices vs Consulta 2.3 con indices



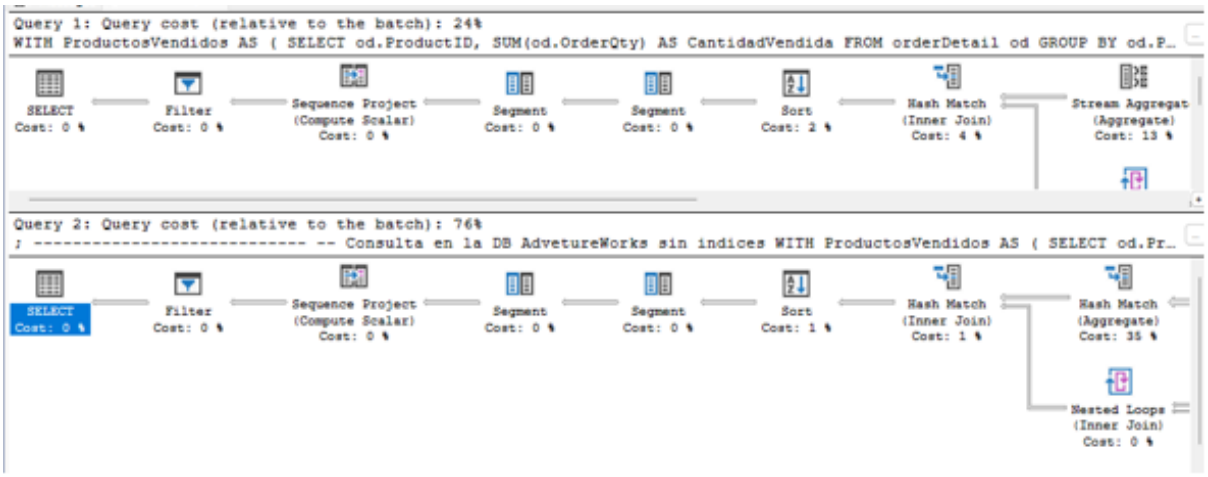
5.6. Consulta 3.1 vs Consulta 3.2 sin indices



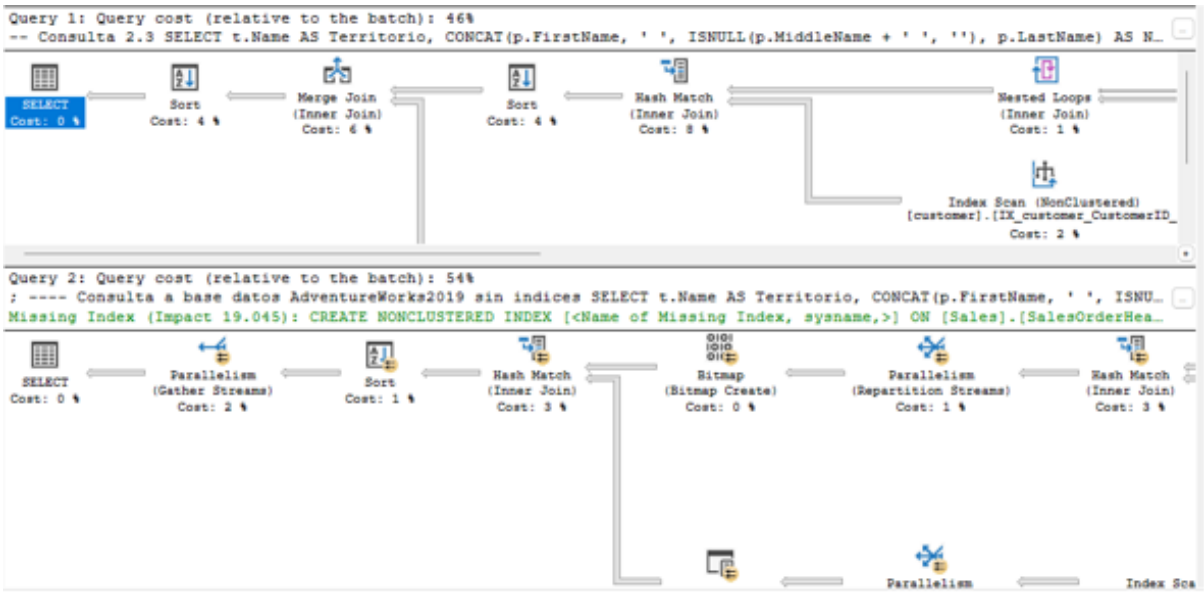
5.7. Consulta 3.1 vs Consulta 3.2 con indices



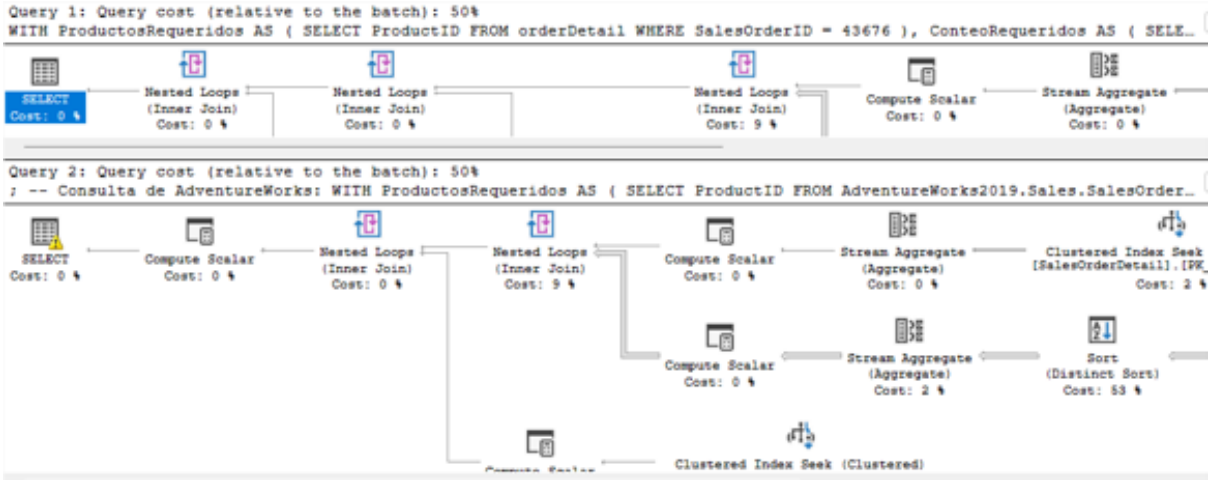
6. Consulta “a” con índices en base de datos “Planes de Ejecución” vs consulta sin índices en la base de datos AdventureWorks2019



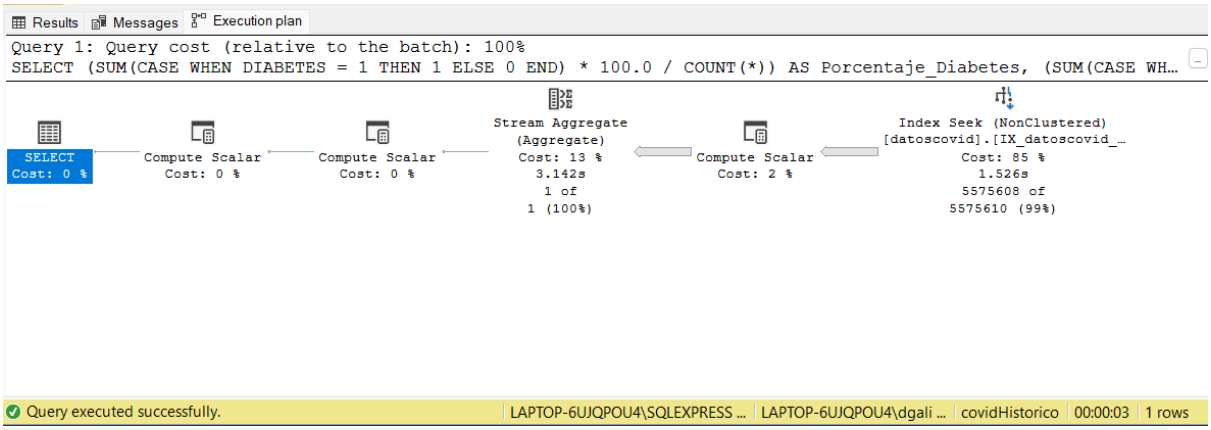
7. **Consulta "b" con índices en base de datos "Planes de Ejecución" vs consulta sin índices en la base de datos AdventureWorks2019**

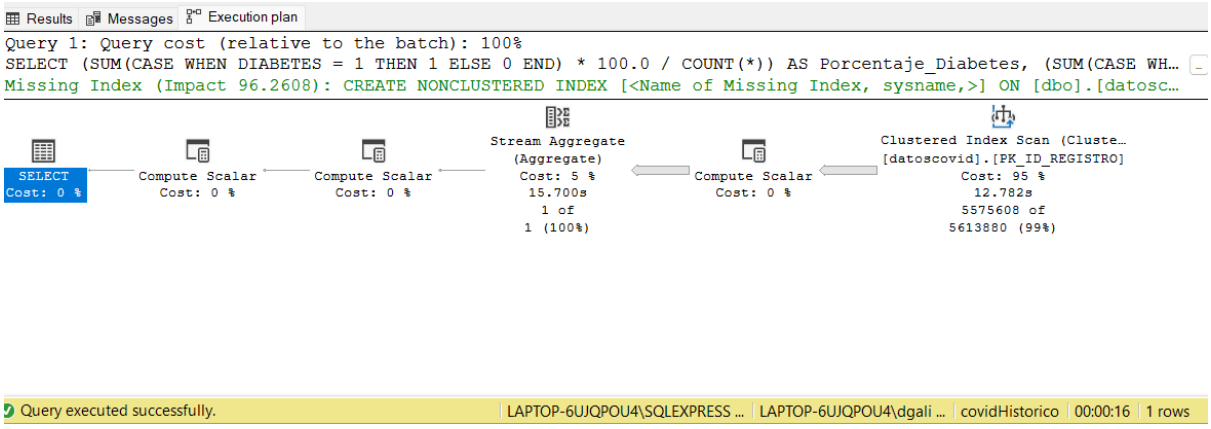


8. **Consulta "c" con índices en base de datos "Planes de Ejecución" vs consulta sin índices en la base de datos AdventureWorks2019**

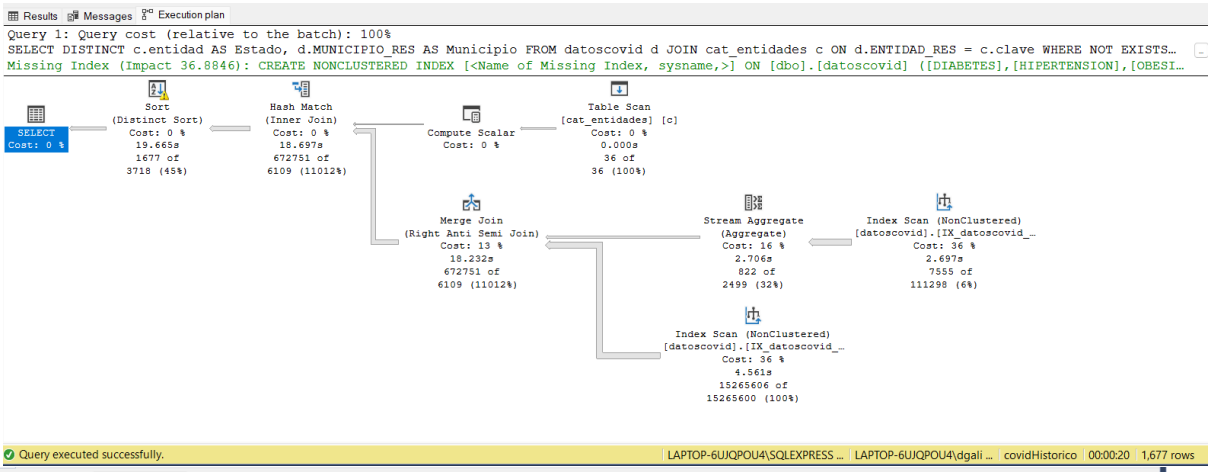


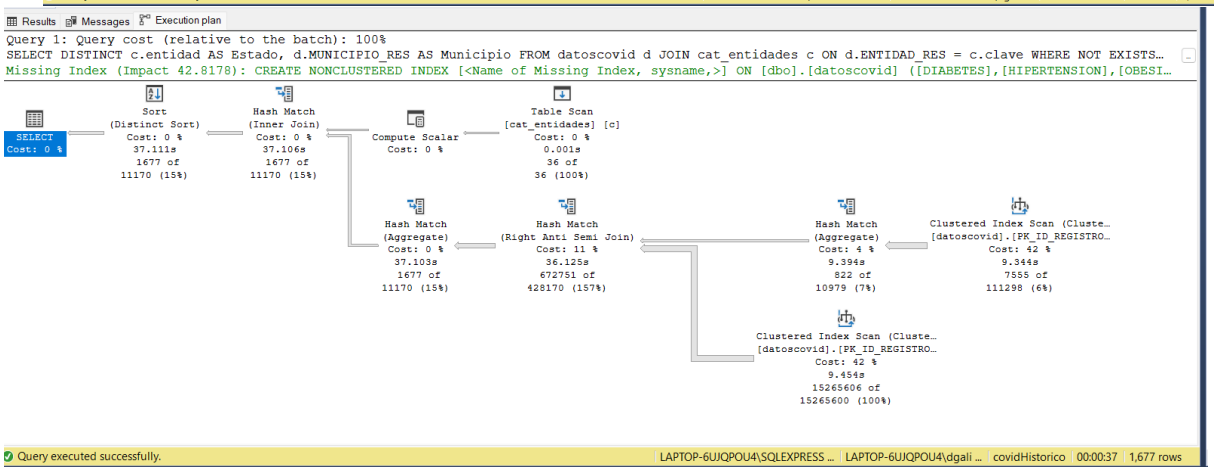
9. Consulta “3” con índices en base de datos “Planes de Ejecución” vs consulta sin índices en la base de datos covidHistorico



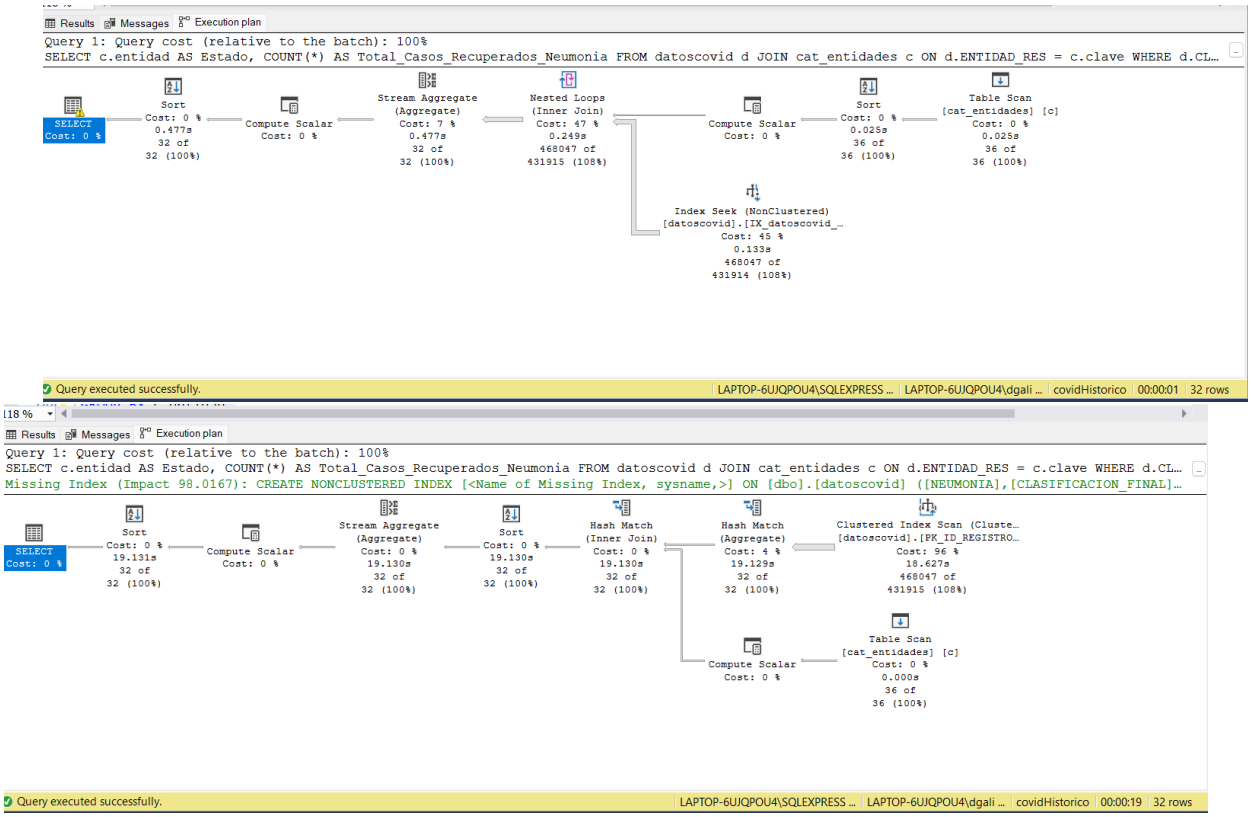


10. Consulta “4” con índices en base de datos “Planes de Ejecución” vs consulta sin índices en la base de datos covidHistorico

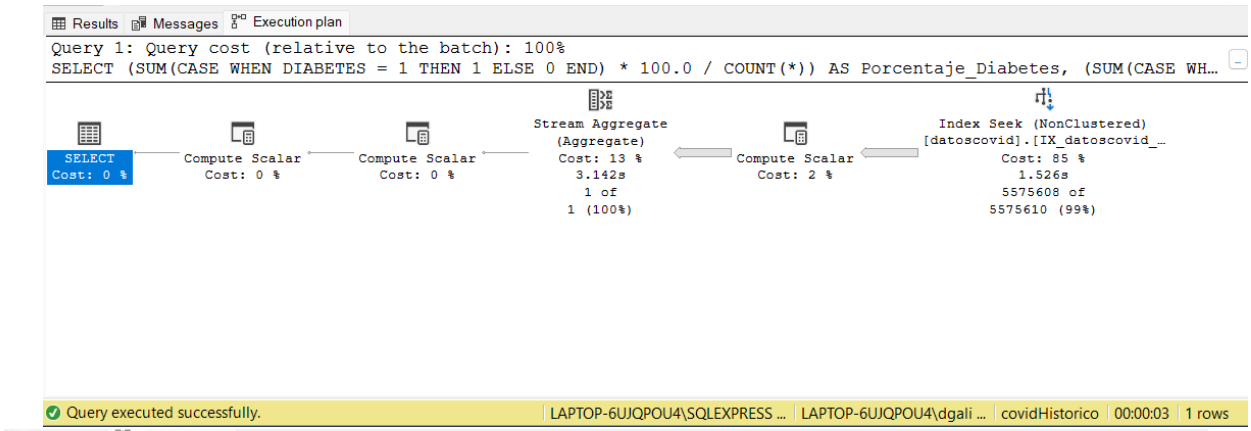


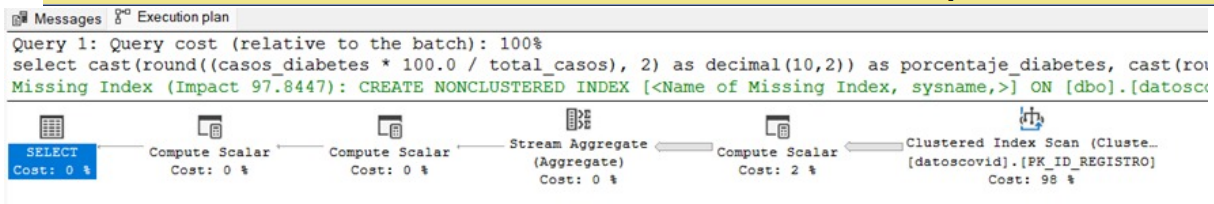


11. Consulta “5” con índices en base de datos “Planes de Ejecución” vs consulta sin índices en la base de datos covidHistorico



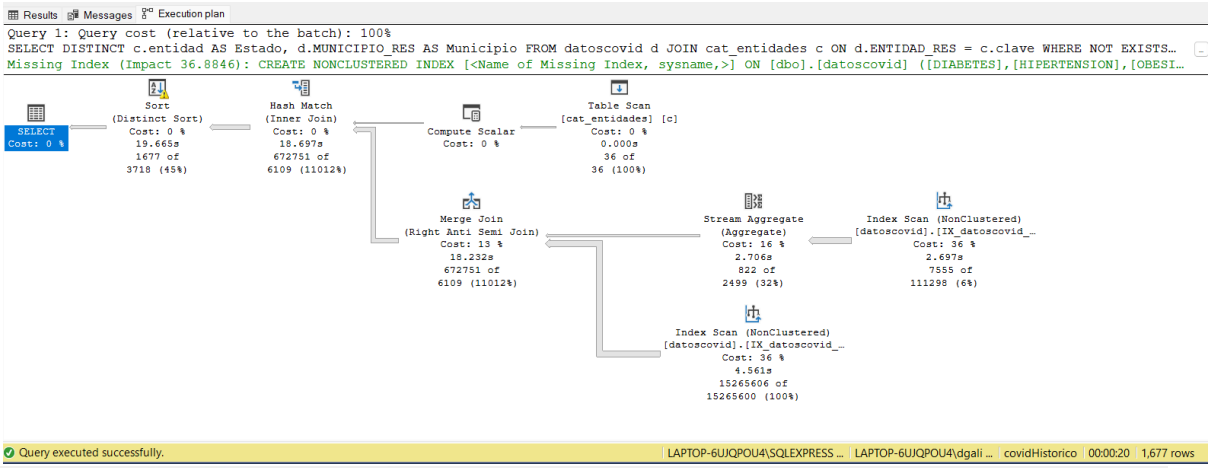
12. Comparacion consulta 3. ”Plan de ejecucion propio vs plan de ejecucion externo”

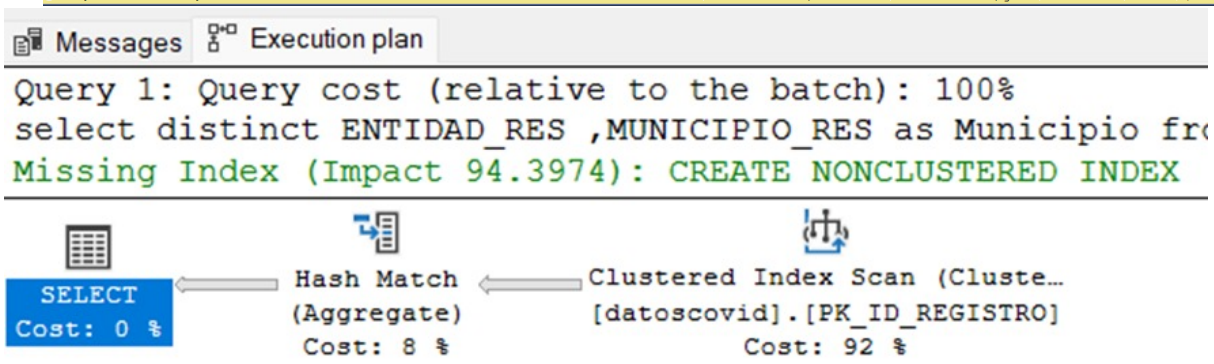




La consulta exerna, es decir, la segunda imagen no uso indices, por lo que el rendimiento se ve afectado a causa de que este metodo hace un escaneo de fila por fila de la tabla. Es por eso que no existio una herramienta que ayudara a filtrar los registros por CLASIFICACION FINAL. Como consecuencia la ejecucion no pudo saltar datos irrelevantes.

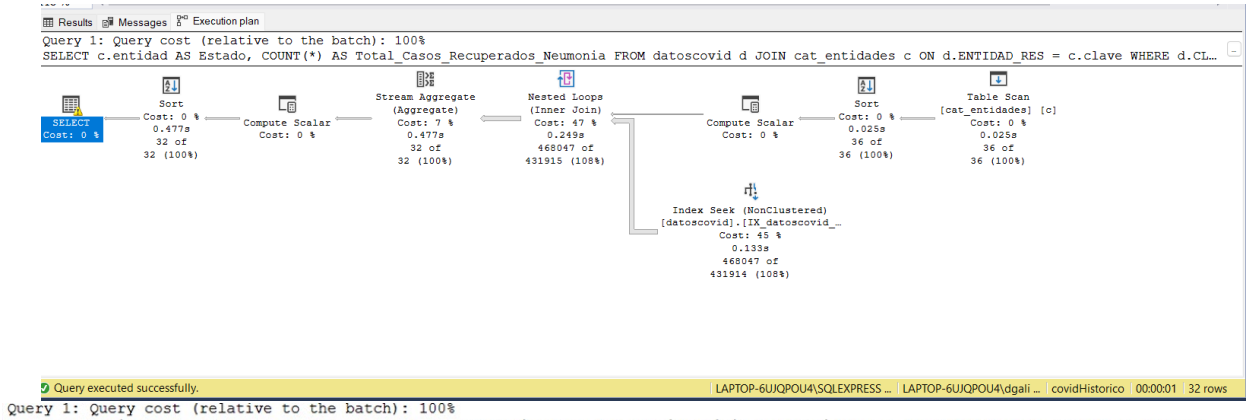
13. Comparacion consulta 4. "Plan de ejecucion propio vs plan de ejecucion externo"

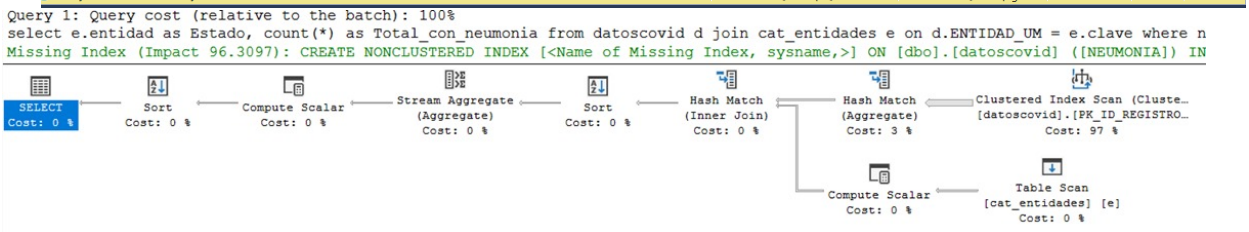




La consulta externa Usa el hash match dentro del DISTINC es un procesamiento muy tardado que duplica filas y hace mas costosa y lenta la busqueda.

14. Comparacion consulta 5. "Plan de ejecucion propio vs plan de ejecucion externo"





El cuello de botella que se produce dentro de la consulta externa es debido a INNER JOIN que une los resultados con $\text{cat}_entidadesloscualessondatosnofiltrados.Ademasdequeordenaunagrancantidaddedatosintermedios.$

15. Conclusiones

Daniel Galicia Cobaxin: Depues de analizar los planes de ejecucion y el impacto de los indices en las tres consultas, observe que el diseño adecuado de indices es crucial para el rendimiento de las consultas. En el caso de la consulta número 3 sobre porcentaje de morbilidades, la creación del índice especifico en CLASIFICACIONFINAL incluyendo las columnas de morbilidades demostro ser muy efectivo. Este indice permitió al motor de base de datos realizar una búsqueda dirigida que redujo drasticamente el tiempo de ejecucion, evitando el escaneo completo de la tabla y procesando unicamente los registros relevantes. El resultado fue un plan de ejecucion eficiente donde la mayor parte del costo se concentró en operaciones productivas como en operaciones de datos ya filtrados.
En la consulta número 5 sobre casos de neumonía, donde el Indice creado no fue utilizado efectivamente. A pesar de incluir las columnas relevantes, el orden de las columnas en el índice no coincidiO con la lógica de filtrado de la consulta, que priorizaba NEUMONIA y CLASIFICACIONFINAL. Esto provoco que el motor ignorara el índice y recurriera a un escaneo completo de la tabla, consumiendo el 97 por ciento del costo de ejecución. El propio motor sugirió un índice alternativo con mayor impacto.
Las columnas usadas en filtros WHERE deben ser las primeras en el índice, seguidas de las de agrupación u ordenamiento, mientras que las columnas solo en el SELECT pueden incluirse mediante INCLUDE.

Pérez López Leonardo: Esta práctica fue interesante ya que no sólo fue hacer consultas, sino que no enfocamos más alla de eso, en el rendimiento y optimizacicón de estas, mediante un análisis de los planes de ejecución y definición de índices.
En las tres consultas me encontré con varios desafios:
La consulta (a) implicaba agrupar y clasificar productos por ventas dentro de sus categorías, lo cual forzaba operaciones de agregación complejas y múltiples uniones con tablas jerárquicas como Product, ProductSubcategory y ProductCategory.
La consulta (b) requería identificar al cliente con más órdenes por territorio, lo que implicaba operaciones de agregación y particionado por SalesTerritoryID, junto con una correcta identificación de clientes a través de relaciones entre Customer, SalesOrderHeader y Person.
La consulta (c) fue la más lógica y relacionalmente compleja: identificar órdenes que compartan todos los productos con una orden de referencia. Este tipo de comparación entre conjuntos es costosa, tanto así que en clase implementamos una solución con división relacional que, a mí parecer es la mejor solución, siempre y cuando se sepa aplicar bien.

Uno de los principales descubrimientos del ejercicio fue la enorme diferencia en el costo de ejecución que se puede lograr al definir índices adecuados.
Esto confirma que los índices no solo aceleran las consultas, sino que también permiten al optimizador del SQL Server elegir mejores planes de ejecución (por ejemplo, usar Index Seek en lugar de Scan, evitar Sort costosos, o permitir Nested Loop Join en lugar de Hash Match).
Otro punto clave fue aprender a leer e interpretar los planes de ejecución gráficos:

- Identificar cuándo una operación de Hash Match está justificada y cuándo no.
- Reconocer el beneficio de un Nested Loop cuando hay una cardinalidad baja en una de las tablas.
- Distinguir entre Index Seek (bueno) e Index Scan (potencialmente costoso).
- Evaluar correctamente los porcentajes de Query Cost en consultas comparativas.

Este análisis mostró que no basta con que una consulta "funcione"; hay que verificar si está usando bien los recursos del sistema. SQL Server puede ejecutar una consulta de muchas formas, pero solo con índices adecuados y una lógica clara en la consulta se alcanzan planes óptimos. Que fue algo que tambié costó un poco de trabajo, ya que la realización de consultas es algo que debemos de poner más en prática para facilitar este tipo de ejercicios.