INSTITUTO POLITECNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERIA Y TECNOLGIAS AVANZADAS - IPN

MATERIA

Bases de Datos Distribuidas

PROFESOR

Carlos De La Cruz Sosa

ALUMNOS

Fernández Guerrero Keb Sebastián

Ramírez Orozco Juan Carlos

Sánchez Herrera Armando Eduardo

TEMA

Planes de Ejecución

Grupo 3TM3 Equipo 06 Practica No. 02

Fecha de asignación:	22/05/25 08:30 hrs
Fecha de entrega:	29/05/25 08:30 hrs

Introducción

En esta práctica se abordará el análisis, diseño y optimización de consultas SQL en un entorno realista utilizando bases de datos relacionales. Como punto de partida, se trabajará con la base de datos AdventureWorks, un modelo ampliamente utilizado que simula un entorno empresarial con operaciones de ventas, producción y gestión de clientes.

El primer objetivo será crear una base de datos llamada practicaPE, y replicar dentro de ella un subconjunto de tablas relevantes desde AdventureWorks. Este subconjunto incluye tablas clave del dominio de ventas, productos, territorios, y personas, permitiendo realizar consultas complejas de análisis de negocio.

Posteriormente, se desarrollarán consultas específicas que exploren patrones como:

- Identificación de productos más vendidos por categoría,
- Clientes más activos por territorio,
- Ordenes similares a una específica según su composición de productos.

Cada consulta será evaluada en términos de rendimiento, utilizando planes de ejecución para analizar cómo el motor de base de datos procesa cada instrucción SQL. A partir de estos análisis, se propondrán índices adecuados para optimizar el acceso a los datos.

Luego, se comparará el rendimiento entre la base original (AdventureWorks) y la nueva (practicaPE), identificando diferencias relevantes. Adicionalmente, se repetirá un análisis similar sobre una base de datos adicional (covidHistorico), aplicando los mismos principios de optimización y comparando resultados entre equipos.

Desarrollo

1. **Crear una base de datos con el nombre: practicaPE, esto con la idea de experimentar con tablas que no contengan llaves ni índices.**

```
create database practicaPE
```

2. **Copiar a la base de datos practicaPE las siguientes tablas de la base de datos AdventureWorks**

```
select * into SalesOrderHeader
from AdventureWorks.Sales.SalesOrderHeader

select * into SalesOrderDetail
from AdventureWorks.Sales.SalesOrderDetail

select * into Customer
from AdventureWorks.Sales.Customer

select * into SalesTerritory
from AdventureWorks.Sales.SalesTerritory

select * into Product
from AdventureWorks.Production.Product

select * into ProductCategory
from AdventureWorks.Production.ProductCategory

select * into ProductSubcategory
from AdventureWorks.Production.ProductSubcategory

select BusinessEntityID, FirstName, LastName into Person
from AdventureWorks.Person.Person
```

3. Codificar las siguientes consultas

Consulta A	Listar el producto más vendido de cada una de las categorías registradas en la base de datos.
Requisitos	- ID del producto más vendido (Por identificador) - ID de la categoría del producto (Por identificador) - Las ventas totales del producto
Responsable	Juan Carlos Ramirez Orozco
Catálogos	- SalesOrderDetail - Product - ProductSubcategory
Comentarios	- Sumamos todos los OrderQty por productos con mismo ID, esto con la intención de obtener el total vendidos por cada producto sin ser solo la orden. - Ahora agrupamos por categoría y despejamos el máximo de cada una. Esto con la idea de obtener los valores máximos asociado a cada categoría. - Repetimos lo anterior la agrupación cambiara por el ID del producto. - Hacemos una igualación para obtener una tabla final que nos permita visualizar los datos requeridos.

	ProductCategoryID	ProductID	Total
1	1	782	2977
2	2	738	1581
3	3	712	8311
4	4	870	6815

Ilustración 1 Resultado de la consulta A

Consulta B	Listar el nombre de los clientes con más ordenes por cada uno de los territorios registrados en la base de datos.
Requisitos	- ID del territorio - Primero nombre del cliente - Apellido del cliente - Compras realizadas en total
Responsable	Juan Carlos Ramirez Orozco
Catálogos	- SalesOrderHeader - Customer - Person
Comentarios	- Hacemos una cuenta de las veces que aparecen cada cliente en determinado territorio, con ayuda de row_number y partition hacemos que la información quede en pequeñas búsquedas. - Ranqueamos los valores que nos da para escoger a uno solo de todos los valores obtenidos. Es un estilo de top 1 pero con la idea de no escoger solo la fila de hasta arriba.

	TerritoryID	FirstName	LastName	Compras_realizadas
1	1	François	Ferrier	12
2	2	Michael	Allen	12
3	3	Kim	Abercrombie	12
4	4	Frances	Adams	12
5	5	Pamela	Cox	12
6	6	Dalton	Perez	28
7	7	April	Shan	25
8	8	David	Brink	8
9	9	Scot	Bent	4
10	10	Christop...	Beck	8

Ilustración 2 Resultado de la consulta B

Consulta C	Listar los datos generales de las ordenes que tengan al menos los mismos productos de la orden con salesorderid = 43676.
Requisitos	- El ID de los tickets que cumplan con tener al menos los mismos productos de la orden 43676
Responsable	Carlos De La Cruz Sosa
Catálogos	- SalesOrderDetail
Comentarios	- Se emplea una consulta por división relacional. - Por un lado, separamos los productos que tiene la orden 43676 para poder consultarlo con cada orden. Al no tener alguno de los artículos mencionados se descarta. - De ese descarte comparamos los datos sobrantes con la tabla nuevamente para dar una presentación de los datos.

	Salesorderid
1	43676
2	43891
3	43894
4	43900
5	44075
6	44285
7	44523
8	44549
9	44567
10	44779
11	44783
12	44792
13	45785
14	45796
15	46039
16	46052
17	46332

Ilustración 3 Resultado de la consulta C

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.**

Los siguientes planes de ejecución son de la consulta A:

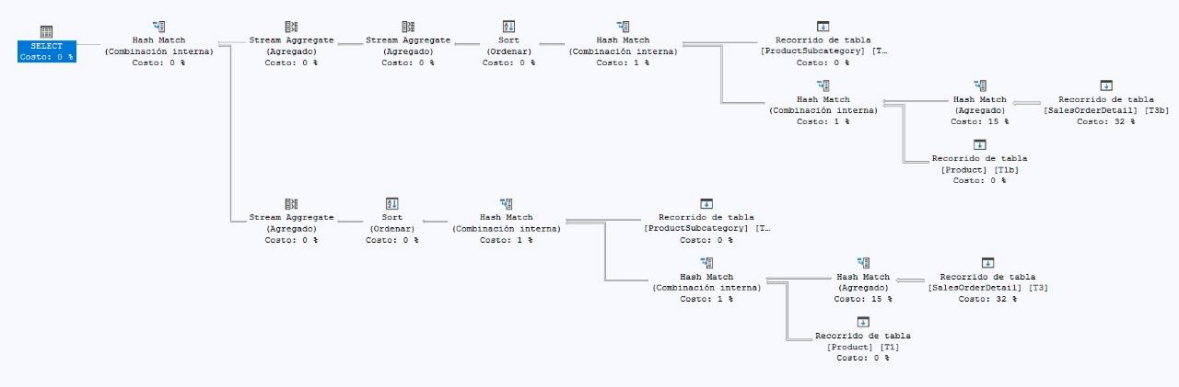

Ilustración 4 Plan de ejecución de la consulta A sin índices

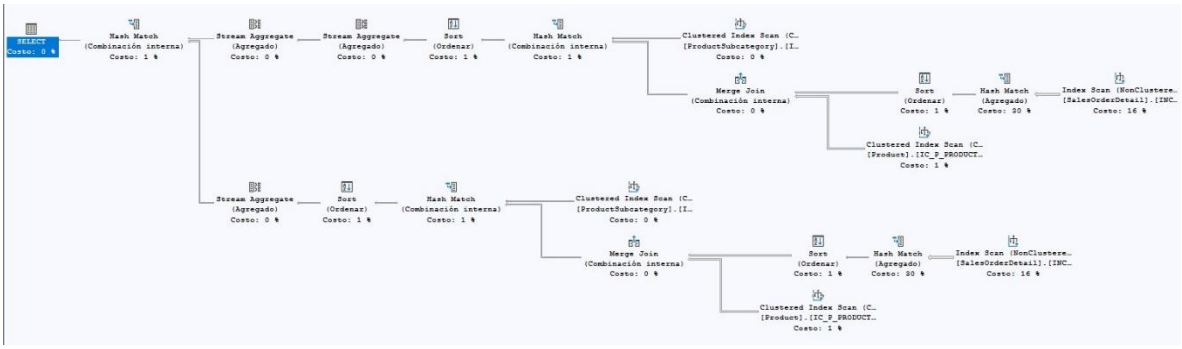

Ilustración 5 Plan de ejecución de la consulta A con índices

Propuesta de índices para la mejora en las consultas:

```
create clustered index IC_P_PRODUCTID on Product (ProductID)

create clustered index IC_PSC_PRODUCTSUBCATEGORY on ProductSubcategory
(ProductSubcategoryID)

create nonclustered index INC_SOD_OQTY_PID on SalesOrderDetail (OrderQty)
include (ProductID)
```

Como cada tabla no contiene un orden especificado empecé por revisar cual podría ser una columna que ayudé tanto en su ordenamiento como en la búsqueda requerida en la consulta.

Al escoger dicha columna se encontraron mejores resultados al momento de hacer el plan de ejecución.

Al inicio se pueden ver porcentajes del 32% en los recorridos de tablas sin índices, lo cual hace que la mayor carga de la ejecución la lleve a cabo en un ordenamiento innecesario con búsquedas repetitivas.

Cuando se usan los índices se puede notar una reducción de la mitad siendo 16%, porque ahora el rendimiento es más estable, y los recursos de dicha consulta se reparte en objetivos más específicos y estables. Esto provoca que una consulta compleja se convierta en un juego de niños para el gestor. Comparando las en ejecución a la par, obtenemos una mejora sustancias del triple de rapidez en cuestión de recursos. Lo que implica que los índices escogidos de esa manera ayudan mucho al gestor a no hacer búsquedas innecesarias con descartes que siempre serán los mismos en cada búsqueda.

Los siguientes planes ejecución son de la consulta B:

Ilustración 6 Plan de ejecución de la consulta B sin índices

Ilustración 7 Plan de ejecución de la consulta B con índices

Propuesta de índices para la mejora en las consultas:

```
create clustered index IC_C_CUSTOMERID on Customer (CustomerID)

create clustered index IC_SOH_SalesOrderID on SalesOrderHeader (SalesOrderID)

create nonclustered index INC_SOH_TerritoryID on SalesOrderHeader (TerritoryID)

create clustered index IC_P_BusinessEntityID on Person (BusinessEntityID)

create nonclustered index IC_P_FIRSTNAME_LAST on Person (FirstName) include
(LastName)
```

Se puede notar un aumento significativo en la cantidad de índices, mi propuesta de índices se debe a que, como la lista pide, se necesitan separar los territorios y hacer búsquedas relacionadas a otras tablas que por la forma de estas no están completamente relacionadas entre ellas.

La solución mas efectiva puede llegar a ser en dividir las tablas grandes de territorios lo cual no lo vuelve mas eficiente y en teoría es hacer trampa. Seria convertir una consulta grande en “n” mini consultas. Pero ocupando eso como partida entonces cree un índice de búsqueda que relacione rápidamente a las personas con su numero de orden la cual relaciona directamente a su orden de venta.

El por que no se nota una mejora significativa en los porcentajes es por dos cosas.

- Poca cantidad de entradas en algunas tablas lo cual no simboliza un gran problema al momento de estar desordenados o no.

- El hecho de que se deben relacionar 4 tablas diferentes que cada una contiene información que complemente la búsqueda de una en otra.

Debido a lo anterior el colocar índices no ayudaría en gran cosa en tablas tan pequeñas. Lo que se me ocurre es tener un histograma mayor o ya sea una vista completa que este a su vez tenga un índice especifico a él. Pero todo esto se sale de la idea principal de solo mejorar la misma consulta añadiendo índices específicos. Mas adelante con índices más especializados como los que contiene la base de datos adventureWorks podemos comparar si hay una mejora sustancial.

Los siguientes planes ejecución son de la consulta C:

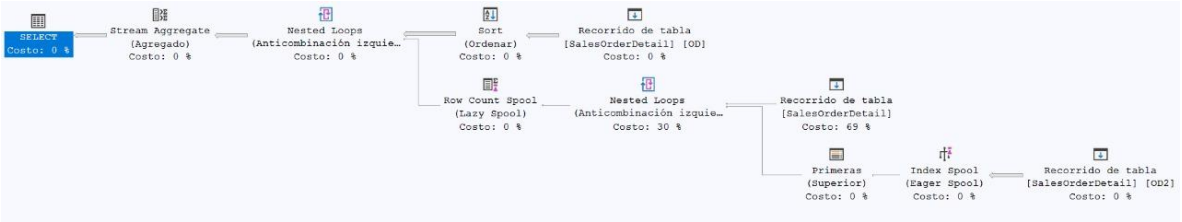

Ilustración 8 Plan de ejecución de la consulta C sin índices

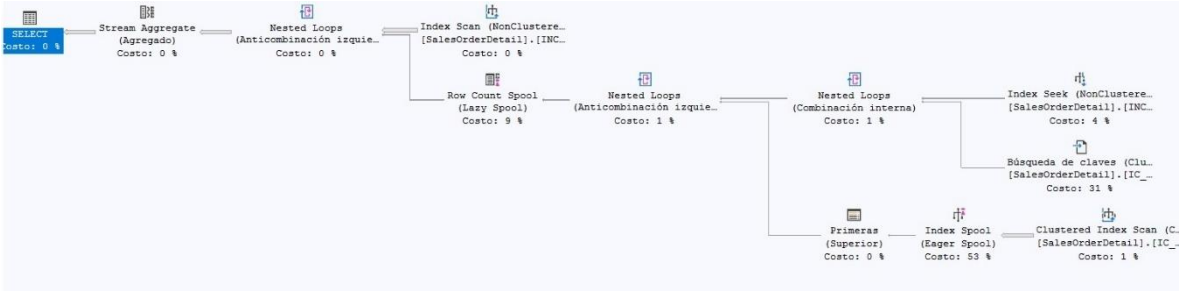

Ilustración 9 Plan de ejecución de la consulta C con índices

Propuesta de índices para la mejora en las consultas:

```
create clustered index IC_SOD_SalesOrderDetailID on SalesOrderDetail
(SalesOrderDetailID)

create nonclustered index INC_SOD_SalesOrderID on SalesOrderDetail
(SalesOrderID)
```

En esta nueva implantación de índices si hubo un cambio radical. El índice clúster que se agrego fue muy efectivo en esta búsqueda (no por nada al momento de ejecutarse conllevo mucho tiempo en ajustar los datos al nuevo índice) y con el apoyo del no clúster complementan lo que anteriormente solo hacia una consulta por si sola, se nota mucho como un proceso que ocupaba casi el 70% de los recursos de la consulta ahora se repartió en diferente parte de esta. El problema si bien se soluciono en una medida. Empeoro otra parte haciendo que el costo aumentara al 53%.

La pregunta que me surgió es por que paso esto. ¿Acaso eso mismo pasaba antes, pero la carga de la otra era superior? Tal parece que es así, al compara las dos consultas se puede ver una diferencia bastante clara, siendo que sin índices la carga de recursos se va a 90% y con

índices a 10%, por lo que, si al parecer ese proceso siempre fue muy cargado, pero no era nada en comparación al ordenamiento conjunto que tenia que hacer ante. Por lo que si bien sigue siendo un costo muy alto. La comparación demuestra que la ejecución mejoro demasiado.

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.**

Comparación de AdventureWorks con mis índices en consulta A:

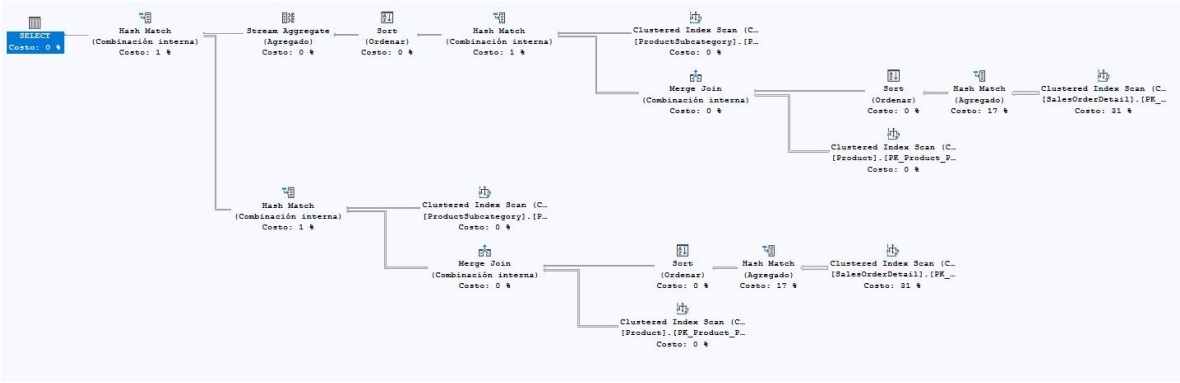

Ilustración 10 Plan de ejecución de adventureWorks sobre la consulta A

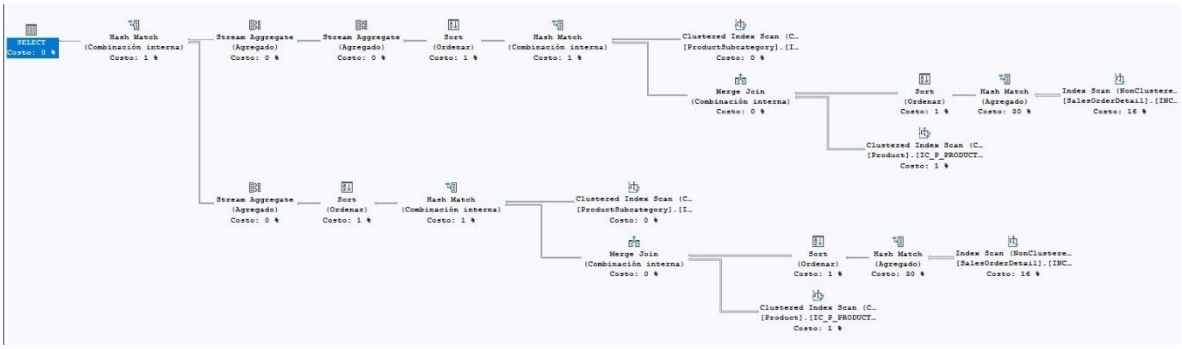

Ilustración 11 Plan de ejecución de mis índices sobre la consulta A

Es muy curioso como pasa lo contrario en cada proceso de búsqueda en tablas. Mientras que mis índices cargan más el Hash Match que provocara esto un rendimiento superior a la hora de la ejecución. La base de datos adventureWorks lo enfoca al revés, siento casi el mismo porcentaje que el mío, pero invertido. Al momento de revisar los recursos utilizados en ambas da una diferencia sustancial tal que:

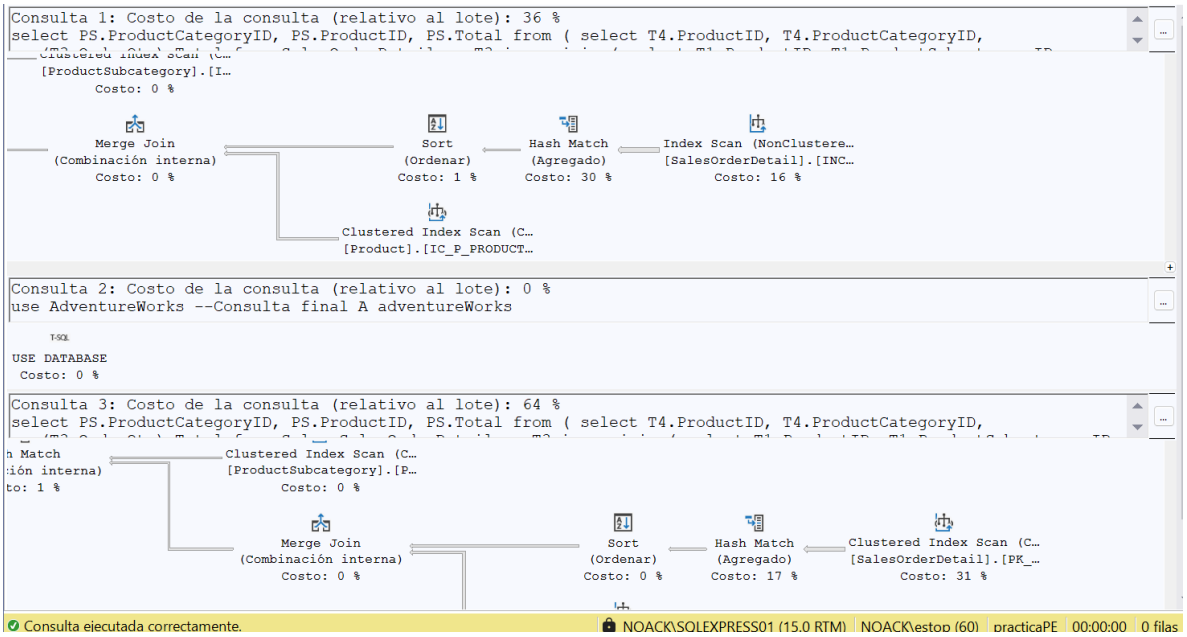

Ilustración 12 costeo de consultas (mis índices y adventureWorks respectivamente)

Aquí se da el dicho que más no significa mejor. Dado a que la consulta realizada en adventureWorks es tres veces peor que usando mis índices. Lo que da como pauta que el no depender mucho de un index scan tan cargado ayuda que el procesamiento sea mejor y los recursos, aunque igualmente repartidos no sean tan pesados para el procesador del servidor en cuestión.

Comparación de AdventureWorks con mis índices en consulta B:

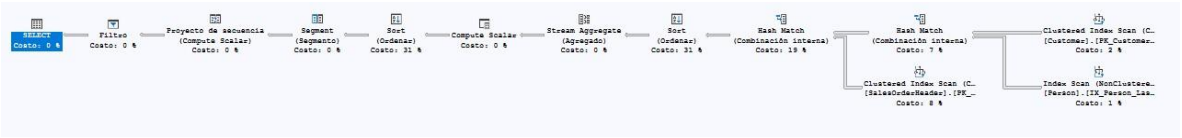

Ilustración 13 Plan de ejecución de adventureWorks sobre la consulta B

Ilustración 14 Plan de ejecución de mis índices sobre la consulta B

Esta consulta es la que se ve en principio mas sencilla al momento de ejecución, pero el uso de tablas separadas con poca relación entre ellas provoca lo que me imagino genera el mismo uso de recursos en ambos planes. Como recordatorio esta consulta es la que más índices tuve que generar para poder llevar a cabo la “mejora”, recordando que esta fue casi nula. Ahora comprobamos que de hecho es exactamente lo mismo tener índices, como no tenerlos, así como tener llaves y varios índices predefinidos (como en adventureWorks).

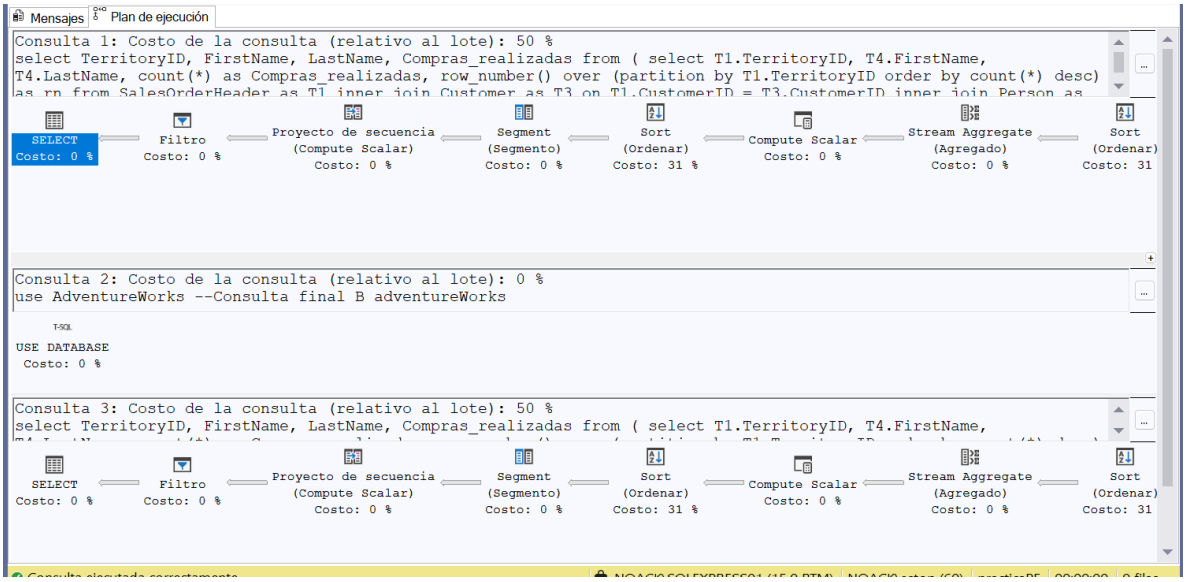

Ilustración 15 costeo de consultas (mis índices y adventureWorks respectivamente)

Se puede ver mas claramente como no mejora en lo más mínimo. Si hay unos pequeños cambios en el costo de CPU y del subárbol, pero esto es tan poco que realmente no representa una mejora sustancial. Supongo que aquí empieza el punto de mejora de consulta. Como dije anteriormente lo mejor seria crear vistas que enlacen las tablas y que estas a su vez tengan un índice más característico, obviamente estamos hablando también de mejorar el histograma del gestor en cada uso, pero sería algo mucho más técnico que algo relacionado al plan de ejecución mejorado por índices así que se hizo lo mejor en esta consulta.

Comparación de AdventureWorks con mis índices en consulta C:

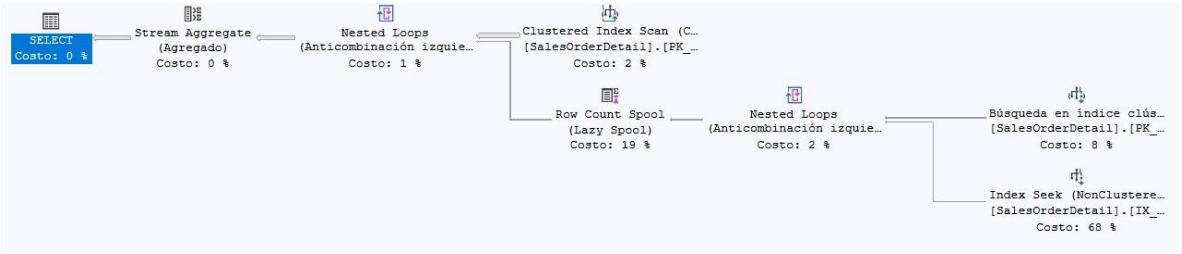

Ilustración 16 Plan de ejecución de adventureWorks sobre la consulta C

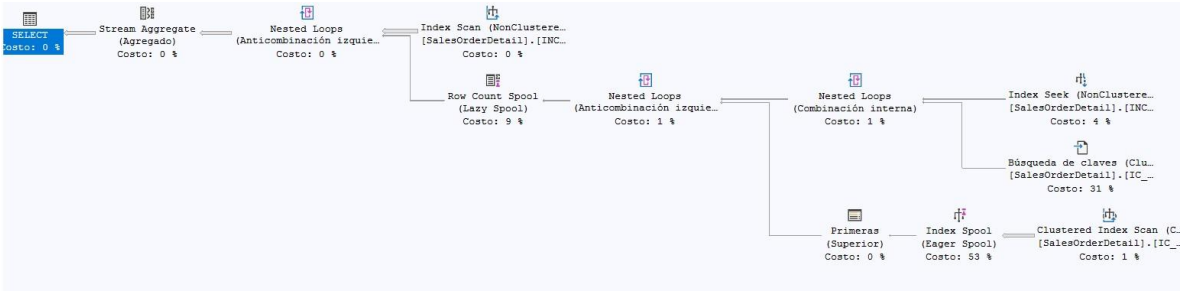

Ilustración 17 Plan de ejecución de mis índices sobre la consulta C

En este caso compartimos porcentajes muy parecidos, pero con uso de recursos completamente diferentes. Hice varias pruebas con índices, pero claro escogí estos por que daban resultados que según yo eran mejores. Vaya error.

Aunque en apariencia se nota mucho lo parecido de los porcentajes esto no son realmente buenos.

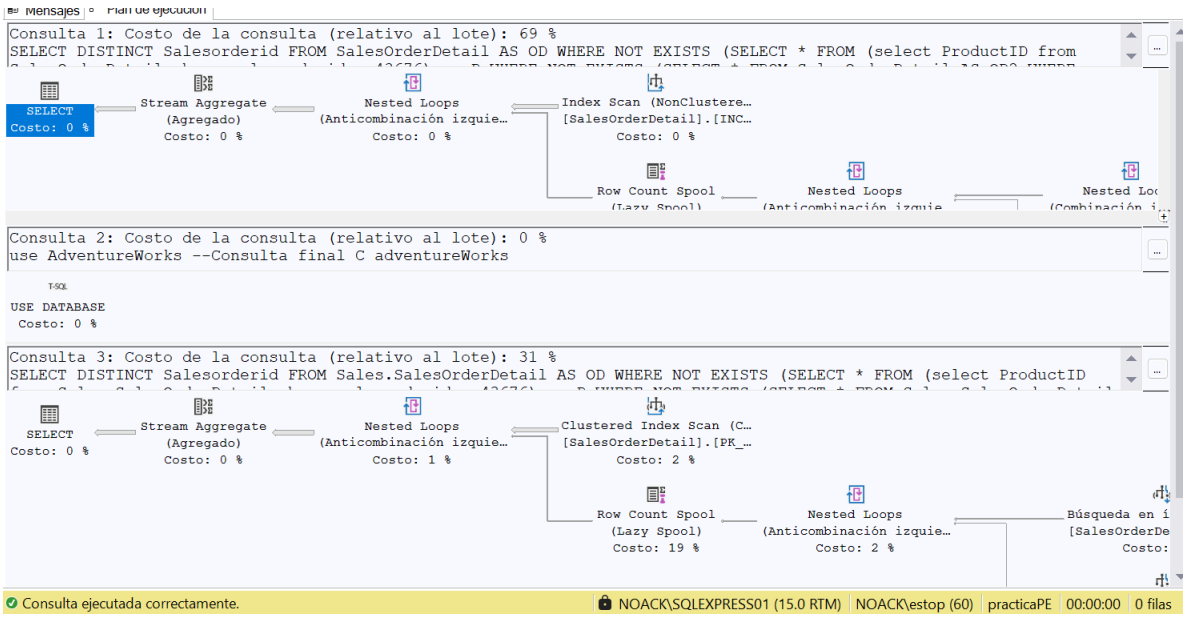

Ilustración 18 costeo de consultas (mis índices y adventureWorks respectivamente)

En este punto aprendí que el costeo en una búsqueda en índices no clúster y clúster si representan un costo en los recursos significativos. Al ser en el mío un costo mayor en el clúster empeora mi costo de consulta debido a que en teoría, al estar ordenados debe buscarlo más rigurosamente, a que tener un no clúster, pero orientado a un orden en apariencia más específico. Es chistoso como este pequeño cambio hace que pierda contra la base de datos adventureWorks.

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

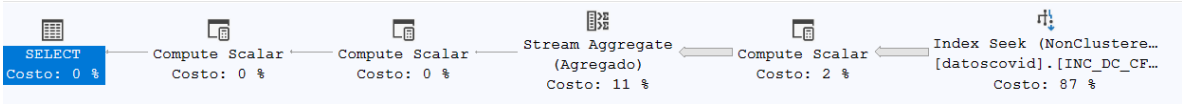

Ilustración 19 Plan de ejecución de la consulta 3 con índices

```
CREATE NONCLUSTERED INDEX INC_DC_CF_3 ON datoscovid (CLASIFICACION_FINAL)
INCLUDE ([DIABETES],[HIPERTENSION],[OBESIDAD])

create clustered index IC_DC_ID_REGSTRO on datoscovid (ID_REGISTRO)

create nonclustered index INC_DC_CLASIFICACION on datoscovid
(CLASIFICACION_FINAL)
```

El primer índice es el bueno, creo que ya empiezo a mejorar en esto. Por otro lado, este índice provoca que la computadora no tenga que ordenar ni seccionar los datos de la tabla. Por lo que a la hora de contarlos y promediarlos se hace relativamente rápido y sencillo. Porque ahora el gestor no busca todas las apariciones de tal padecimiento. Sea cualquiera de las búsquedas ahora solo busca donde empieza las apariciones de diabetes y donde termina para empezar con el siguiente padecimiento en cuestión.

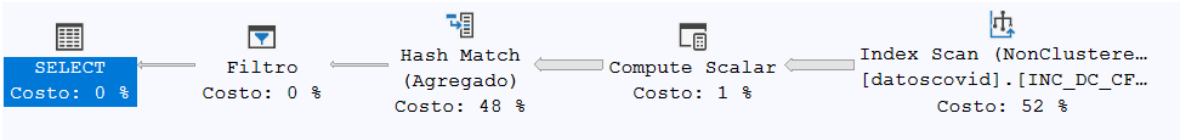

Ilustración 20 Plan de ejecución de la consulta 4 con índices

```
CREATE NONCLUSTERED INDEX INC_DC_CF_5 ON datoscovid ([CLASIFICACION_FINAL])
INCLUDE ([MUNICIPIO_RES],[DIABETES],[HIPERTENSION],[OBESIDAD],[TABAQUISMO])
```

Con ese sencillo índice logre crear una consulta mucho mas eficiente. A tal grado que el table scan era de aproximadamente del 90%, pero esto logra distribuir los recursos. Esto es por lo mismo al punto anterior, el gestor ya no debe hacer búsquedas minuciosas sino solo buscar donde inicia y donde termina la búsqueda.

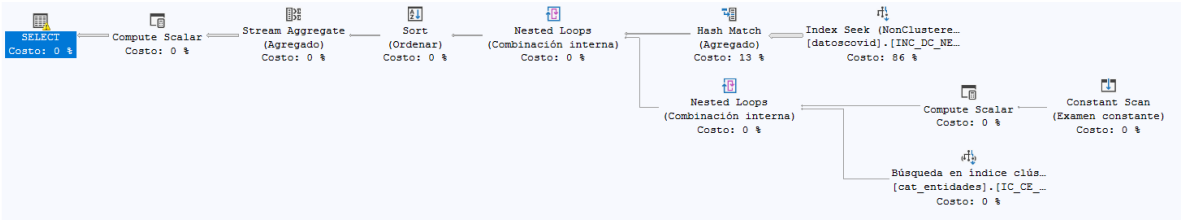

Ilustración 21 Plan de ejecución de la consulta 5 con índices

```
CREATE NONCLUSTERED INDEX INC_DC_NEU_CF_2 ON datoscovid
([NEUMONIA],[CLASIFICACION_FINAL]) INCLUDE ([ENTIDAD_RES],[FECHA_DEF])

create clustered index IC_CE_ENTIDAD on cat_entidades (clave)
```

Aquí se nota mucho la carga que tiene que hacer el índice que cree. Pero al ser no clúster no gasta tantos recursos y logra hacer una ejecución más limpia y rápida que en otras opciones que fue haciendo. Siento esta la que mejor desempeño demostró al final de la carga.

7. **Comparar los planes de ejecución del punto 6 con los planes de ejecución de otro equipo (Es el equipo 4).**

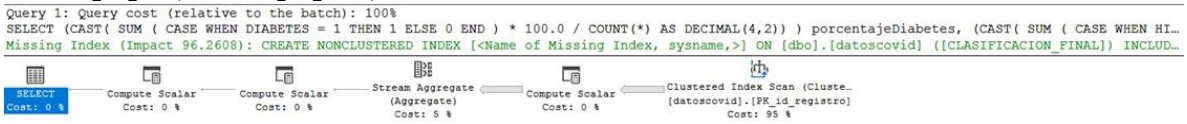

Ilustración 22 consulta 3 del equipo 4

Si lo presentamos contra la solución que proponemos puedo decir que es mucho más efectiva mi índice. Anteriormente vimos la diferencia entre usar búsquedas entre clúster y no clúster, siendo que el clúster con mayor porcentaje representa un mayor uso de recursos. Ahora con esos datos mi solución no solo usa una sin clúster, sino que su porcentaje es mucho menor. Si se compara una contra la otra podría apostar que es aproximadamente 5 veces mejor la opción que propongo con mis índices.

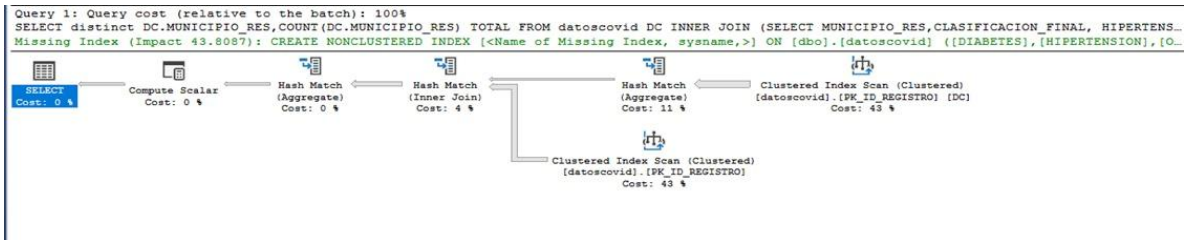

Ilustración 23 consulta 4 del equipo 4

Esta solución de los compañeros es mucho peor en lo que cabe. Usa dos clústeres con porcentaje muy alto. Siendo que casi todos los recursos solo se dedican a deshacer el orden dado por sus índices. En mi opción si esta un poco cargado el index scan pero al ser no clúster el recurso es ligero lo que ayuda a mejorar bastante el flujo de datos, eso sin mencionar que aquí hace uso de dos scan que me imagino que no se ve en la nuestras por el uso de with en fin cuestión de diseño. Por mas creo que seria un doble o triple mejor una con respecto de la otra.

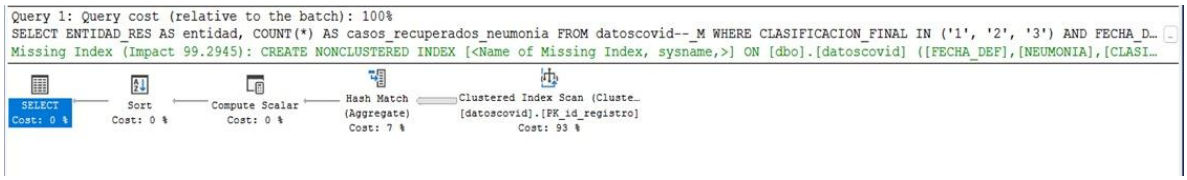

Ilustración 24 consulta 5 del equipo 4

Aunque en este plan se ve mas simple. Se nota una carga excesiva en el index scan, pero un hash match poco cargado. Como anteriormente comprobé lo que mejora y por mucho es distribuir el porcentaje del index scan. El problema es constante con este equipo porque ordenan una tabla la cual empeora su rendimiento en vez de ayudarlo. Supongo que no terminaron de entender la practica o simplemente no experimentaron lo suficiente.

8. Conclusiones por equipo argumentando la selección de índices.

Se ha demostrado de manera práctica la importancia del análisis de planes de ejecución en SQL Server como herramienta fundamental para la optimización del rendimiento de consultas. A través de la comparación entre las bases de datos AdventureWorks y practicaPE (usando el covidHistorico para compararse entre equipos del grupo), así como mediante la creación de índices personalizados, se observaron mejoras significativas en el uso eficiente de los recursos del sistema y en los tiempos de respuesta.

Uno de los hallazgos más relevantes fue el impacto del tipo de escaneo de índices en el rendimiento de las consultas. Por ejemplo:

- Index Scan (Clustered) representó entre 60% a 75% del costo estimado en algunas consultas sin optimización, ya que, al no haber filtros selectivos o índices adecuados, el motor de base de datos opta por recorrer toda la tabla.
- En contraste, tras la implementación de índices no clustered focalizados en las columnas más utilizadas en filtros (WHERE) y condiciones de unión (JOIN), el costo se redujo significativamente, y el plan pasó a usar Index Seek (NonClustered) con un costo estimado entre 10% y 25%, demostrando una mejora del 50% o más en eficiencia.

Además, se analizaron los diferentes métodos de combinación utilizados por el optimizador para unir tablas:

- Hash Match fue empleado frecuentemente en consultas sin índices adecuados. Aunque útil para grandes volúmenes de datos, su costo estimado osciló entre 30% y 50%, principalmente por su necesidad de realizar operaciones de hash temporales en memoria.
- Nested Loops fue preferido en consultas con índices adecuados y tamaños de datos pequeños a moderados, mostrando un costo de ejecución mucho menor, alrededor del 10% al 20%. Fue particularmente eficiente cuando se trabajó con tablas sin claves per bien indexadas.
- Merge Join se utilizó en situaciones donde ambas tablas estaban ordenadas y adecuadamente indexadas. Su desempeño fue balanceado, con un costo estimado entre 20% y 30%, y mostró ventajas cuando se trataba de consultas que requerían grandes volúmenes de combinación ordenada.

Esta práctica evidencia que la correcta distribución de procesos de ejecución, junto con la aplicación estratégica de índices, no solo reduce significativamente el costo de ejecución de las consultas, sino que también mejora la escalabilidad y la capacidad de respuesta del sistema. Ahora en mas cada vez que realice bases de datos (con la esperanza de hacerlo de manera distribuida) hare planes de ejecución tanto eficientes como enfocados a consultas rápidas y certeras. Hay un gran camino por recorrer, pero es muy divertido estar en él.