Estudiantes:

Dominic José Casares Aguirre c.2022085016

Alonso Navarro Carrillo c.2022236435

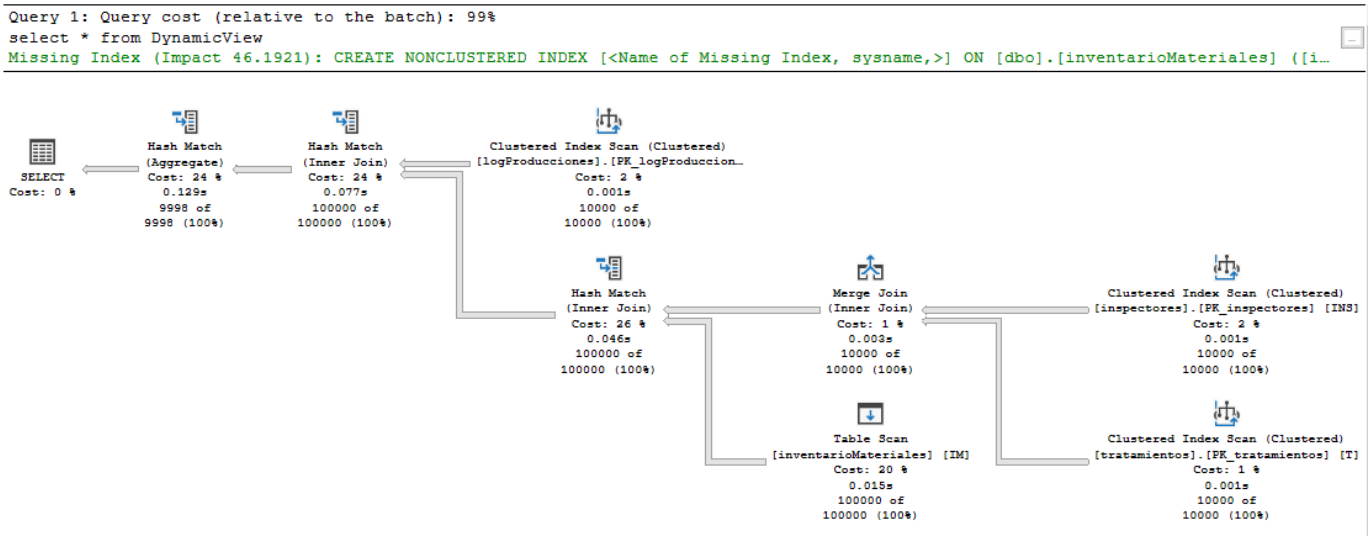
Justificación Teórica sobre Diferencia de Views

Query:

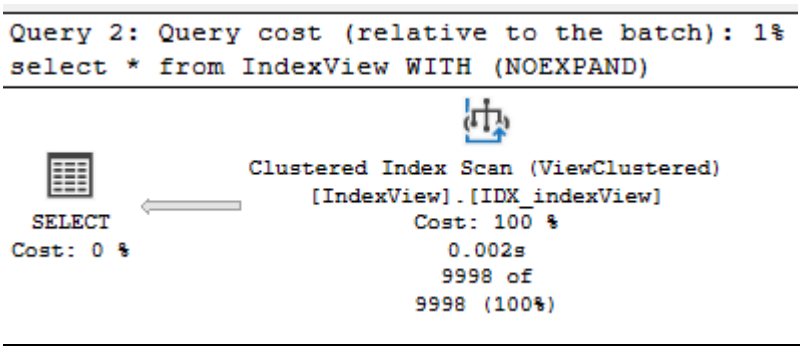
```
-- View Dinamica
IF OBJECT_ID('dbo.DynamicView','view') IS NOT NULL
	DROP VIEW dbo.DynamicView;
GO
CREATE VIEW dbo.DynamicView AS
SELECT INS.nombre,count_big(*) AS conteo
FROM dbo.inventarioMateriales IM
	INNER JOIN dbo.inspectores INS on INS.inspectorId = IM.inspectorId
	INNER JOIN dbo.logProducciones P on P.produccionId = IM.produccionId
	INNER JOIN dbo.materiales M on M.materialId= IM.materialId
	INNER JOIN dbo.tratamientos T on T.tratamientoId = IM.inspectorId
GROUP BY INS.nombre
GO

-- View Indexada
IF OBJECT_ID('dbo.IndexView','view') IS NOT NULL
	DROP VIEW dbo.IndexView;
GO
CREATE VIEW dbo.IndexView WITH SCHEMABINDING AS
SELECT INS.nombre,count_big(*) AS conteo
FROM dbo.inventarioMateriales IM
	INNER JOIN dbo.inspectores INS on INS.inspectorId = IM.inspectorId
	INNER JOIN dbo.logProducciones P on P.produccionId = IM.produccionId
	INNER JOIN dbo.materiales M on M.materialId= IM.materialId
	INNER JOIN dbo.tratamientos T on T.tratamientoId = IM.inspectorId
GROUP BY INS.nombre
GO
GO
CREATE UNIQUE CLUSTERED INDEX IDX_indexView ON IndexView(nombre) ;
GO
```

Planes de Ejecución:

View Dinámica



View Indexada



Análisis y Justificación Teórica:

Desplegando un total aproximado de 10000 registros en cada view, mediante un análisis de la view dinámica y la indexada se puede apreciar que el Query 1(View dinámica) tiene un costo de 99%, por otro lado, el Query 2(View indexada) tiene un costo de 1%, esto se debe a que la view dinámica realiza un recorrido secuencial en donde debe recorrer toda la tabla para poder localizar un registro, en cambio en la view indexada se trabajan índices por lo que el acceso a los registros es directo y disminuye en gran magnitud el costo de ejecución.

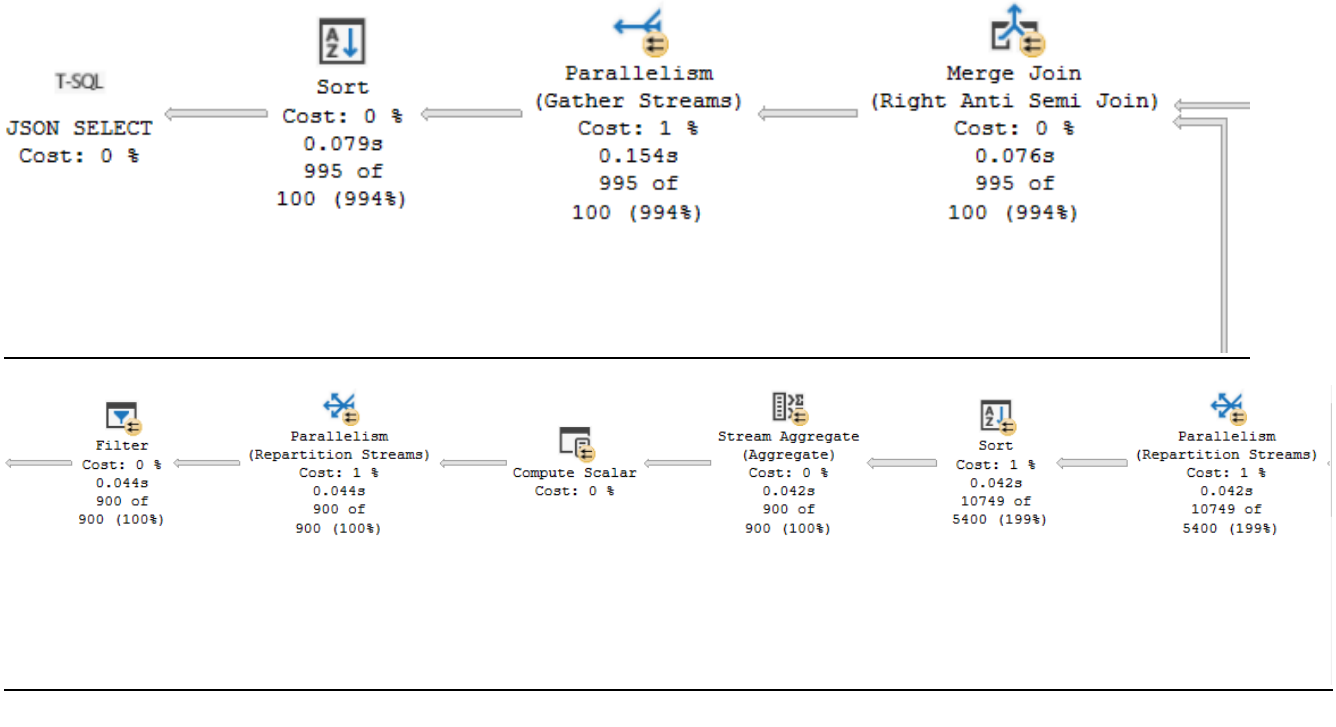
Estrategias de Optimización para Consulta Compleja

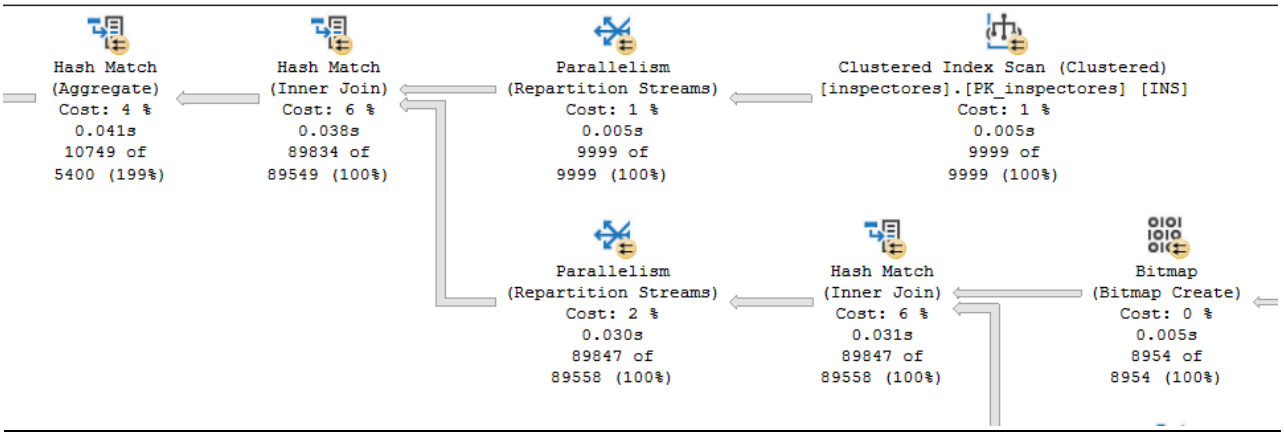
Query:

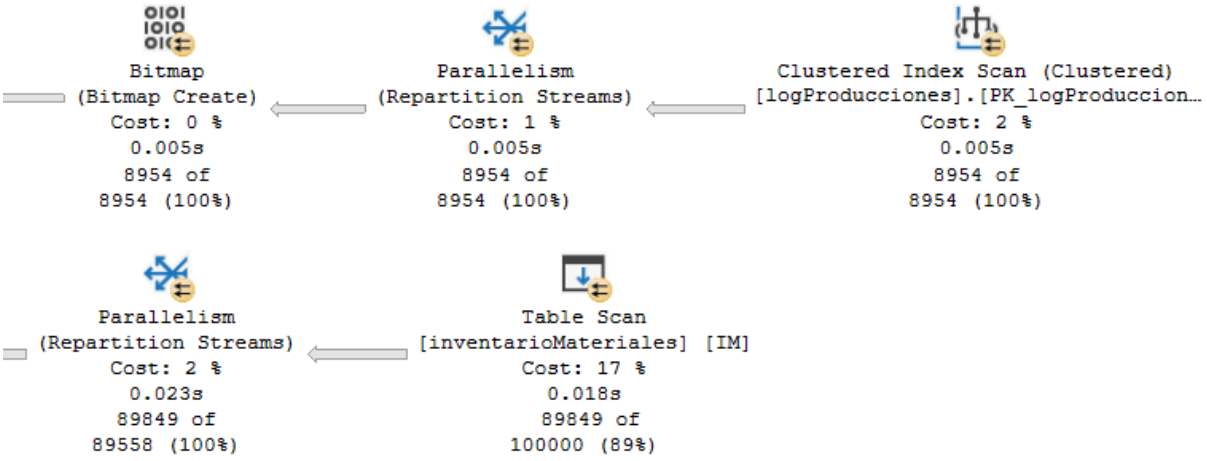
```
GO
SELECT
	-- Agregate Functions
	SUM(P.cantidad) AS SumaCantidad,
	STDEV(INS.inspectorId) AS InspectorCount
FROM dbo.inventarioMateriales IM
	LEFT JOIN dbo.inspectores INS on INS.inspectorId = IM.inspectorId-- Left Join
	INNER JOIN dbo.logProducciones P on P.produccionId = IM.produccionId
```

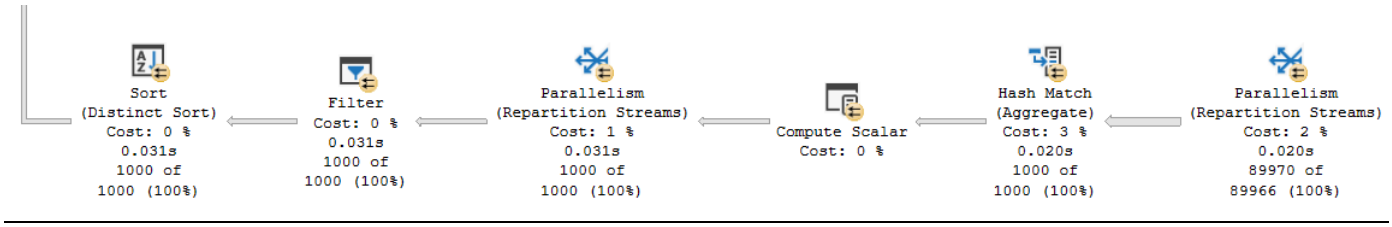
```
        INNER JOIN dbo.materiales M on M.materialId= IM.materialId
WHERE
        INS.inspectorId > 1000 AND                              -- Where Primary Field
        uniMedida = 'kg'                                        -- Igualdad
GROUP BY P.cantidad                                             -- Group By
HAVING SUM(P.cantidad) > 5

EXCEPT                                                          -- EXCEPT

SELECT
        -- Agregate Functions
        SUM(P.cantidad) AS SumaCantidad,
        STDEV(INS.inspectorId) AS InspectorCount
FROM dbo.inventarioMateriales IM
        LEFT JOIN dbo.inspectores INS on INS.inspectorId = IM.inspectorId-- Left Join
        INNER JOIN dbo.logProducciones P on P.produccionId = IM.produccionId
WHERE
        P.cantidad > 100 AND                                    -- Where Non Primary Field
        INS.nombre != 'NombreIns:1'                             -- Desigualdad
GROUP BY P.cantidad                                             -- Group By
HAVING SUM(P.cantidad) > 5

ORDER BY SUM(P.cantidad) DESC                                   -- Sort
FOR JSON PATH                                                   -- For JSON
GO
```

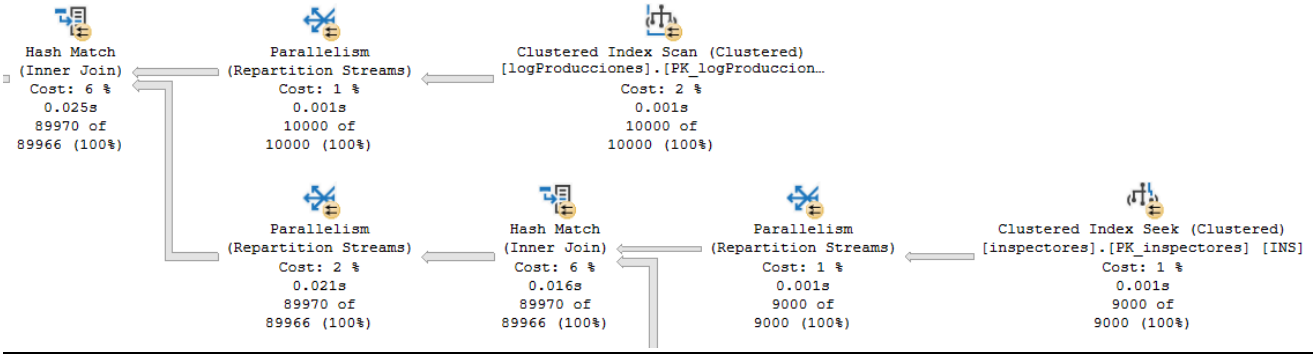
Execution Plan (Sin optimización):

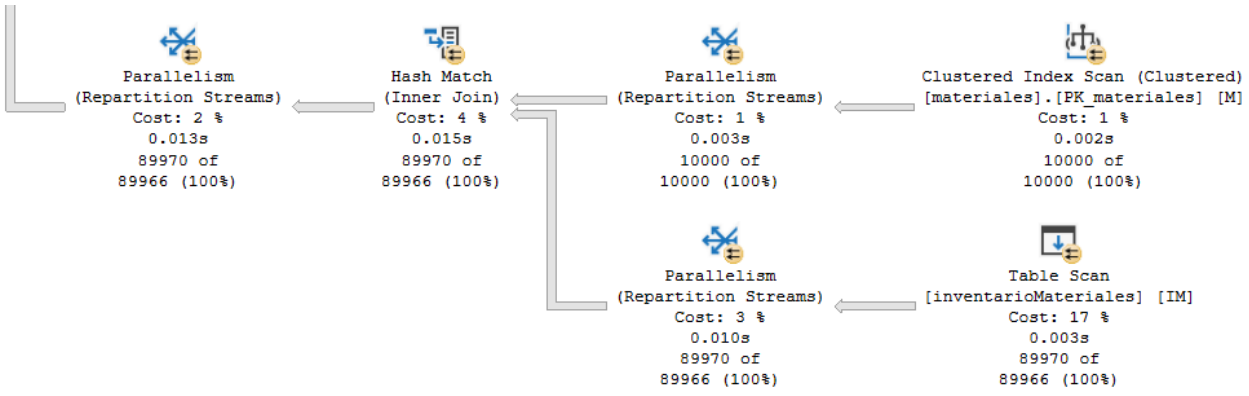


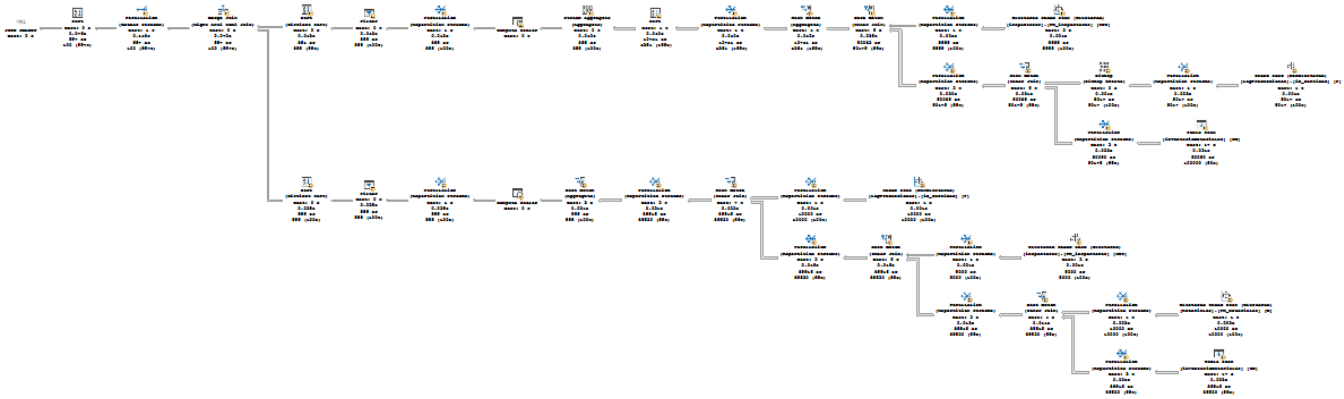

Hash Match
(Aggregate)
Cost: 4 %
0.041s
10749 of
5400 (199%)
Hash Match
(Inner Join)
Cost: 6 %
0.038s
89834 of
89549 (100%)
Parallelism
(Repartition Streams)
Cost: 1 %
0.005s
9999 of
9999 (100%)
Clustered Index Scan (Clustered)
[inspectores].[PK_inspectores] [INS]
Cost: 1 %
0.005s
9999 of
9999 (100%)
Parallelism
(Repartition Streams)
Cost: 2 %
0.030s
89847 of
89558 (100%)
Hash Match
(Inner Join)
Cost: 6 %
0.031s
89847 of
89558 (100%)
Bitmap
(Bitmap Create)
Cost: 0 %
0.005s
8954 of
8954 (100%)


Bitmap
(Bitmap Create)
Cost: 0 %
0.005s
8954 of
8954 (100%)
Parallelism
(Repartition Streams)
Cost: 1 %
0.005s
8954 of
8954 (100%)
Clustered Index Scan (Clustered)
[logProducciones].[PK_logProduccion…
Cost: 2 %
0.005s
8954 of
8954 (100%)
Parallelism
(Repartition Streams)
Cost: 2 %
0.023s
89849 of
89558 (100%)
Table Scan
[inventarioMateriales] [IM]
Cost: 17 %
0.018s
89849 of
100000 (89%)


Sort
(Distinct Sort)
Cost: 0 %
0.031s
1000 of
1000 (100%)
Filter
Cost: 0 %
0.031s
1000 of
1000 (100%)
Parallelism
(Repartition Streams)
Cost: 1 %
0.031s
1000 of
1000 (100%)
Compute Scalar
Cost: 0 %
Hash Match
(Aggregate)
Cost: 3 %
0.020s
1000 of
1000 (100%)
Parallelism
(Repartition Streams)
Cost: 2 %
0.020s
89970 of
89966 (100%)


Hash Match
(Inner Join)
Cost: 6 %
0.025s
89970 of
89966 (100%)
Parallelism
(Repartition Streams)
Cost: 1 %
0.001s
10000 of
10000 (100%)
Clustered Index Scan (Clustered)
[logProducciones].[PK_logProduccion…
Cost: 2 %
0.001s
10000 of
10000 (100%)
Parallelism
(Repartition Streams)
Cost: 2 %
0.021s
89970 of
89966 (100%)
Hash Match
(Inner Join)
Cost: 6 %
0.016s
89970 of
89966 (100%)
Parallelism
(Repartition Streams)
Cost: 1 %
0.001s
9000 of
9000 (100%)
Clustered Index Seek (Clustered)
[inspectores].[PK_inspectores] [INS]
Cost: 1 %
0.001s
9000 of
9000 (100%)





Puntos Críticos y Optimización

Workload:

Table Scan

Scan rows from a table.

Physical Operation	Table Scan
Logical Operation	Table Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	100000
Actual Number of Rows for All Executions	90073
Actual Number of Batches	0
Estimated I/O Cost	0,652833
Estimated Operator Cost	0,680353 (15%)
Estimated Subtree Cost	0,680353
Estimated CPU Cost	0,0275196
Estimated Number of Executions	1
Number of Executions	8
Estimated Number of Rows for All Executions	90066,1
Estimated Number of Rows Per Execution	90066,1
Estimated Number of Rows to be Read	100000
Estimated Row Size	27 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	44

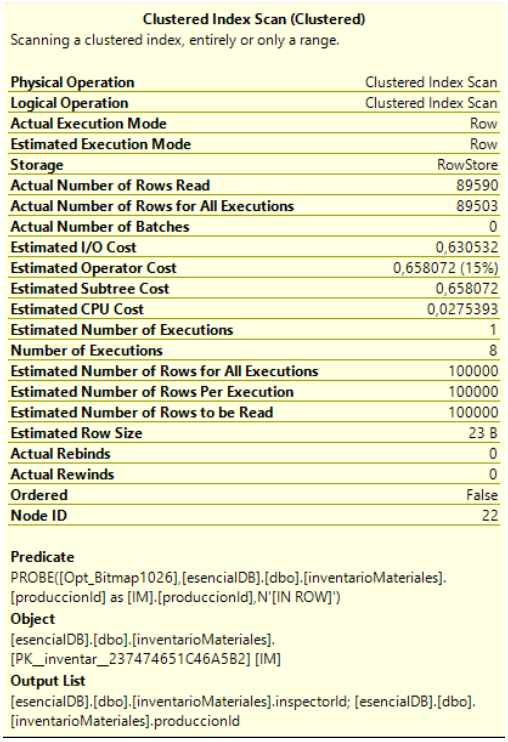
Predicate
[esencialDB].[dbo].[inventarioMateriales].[inspectorId] as [IM].[inspectorId]>(1000)
Object
[esencialDB].[dbo].[inventarioMateriales] [IM]
Output List
[esencialDB].[dbo].[inventarioMateriales].materialId; [esencialDB].[dbo].[inventarioMateriales].inspectorId; [esencialDB].[dbo].[inventarioMateriales].produccionId

Explicación:

El engine realiza un recorrido secuencial de la tabla inventarioMateriales para poder obtener todos los registros en donde su inspectorId cumple la condicional establecida, devolviendo el materialId, inspectorId y produccionId dichos registros.

Norma:

1. Utilizar un primary key que identifique los registros de la tabla inventarioMateriales y así tener un acceso directo para comparar y así disminuir el costo y tiempo ejecución.



Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	89590
Actual Number of Rows for All Executions	89503
Actual Number of Batches	0
Estimated I/O Cost	0,630532
Estimated Operator Cost	0,658072 (15%)
Estimated Subtree Cost	0,658072
Estimated CPU Cost	0,0275393
Estimated Number of Executions	1
Number of Executions	8
Estimated Number of Rows for All Executions	100000
Estimated Number of Rows Per Execution	100000
Estimated Number of Rows to be Read	100000
Estimated Row Size	23 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	22

Predicate
PROBE([Opt_Bitmap1026],[esencialDB].[dbo].[inventarioMateriales].[produccionId] as [IM].[produccionId],N'[IN ROW]')

Object
[esencialDB].[dbo].[inventarioMateriales].[PK__inventar__237474651C46A5B2] [IM]

Output List
[esencialDB].[dbo].[inventarioMateriales].inspectorId; [esencialDB].[dbo].[inventarioMateriales].produccionId

2. Agregar un clustered index para las produccionId de la tabla inventarioMateriales y así disminuir el porcentaje del costo de ejecución.

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	100000
Actual Number of Rows for All Executions	90073
Actual Number of Batches	0
Estimated I/O Cost	0,607569
Estimated Operator Cost	0,635109 (14%)
Estimated Subtree Cost	0,635109
Estimated CPU Cost	0,0275393
Estimated Number of Executions	1
Number of Executions	8
Estimated Number of Rows for All Executions	90066,1
Estimated Number of Rows Per Execution	90066,1
Estimated Number of Rows to be Read	100000
Estimated Row Size	27 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	44

Predicate

[esencialDB].[dbo].[inventarioMateriales].[inspectorId] as [IM].[inspectorId]>(1000)

Object

[esencialDB].[dbo].[inventarioMateriales].[ix_produccionId] [IM]

Output List

[esencialDB].[dbo].[inventarioMateriales].materialId; [esencialDB].[dbo].[inventarioMateriales].inspectorId; [esencialDB].[dbo].[inventarioMateriales].produccionId

3. Agregar un non clustered index para el inspectorId y produccionId de la tablaInventarioMateriales y así disminuir notablemente el costo.

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Actual Execution Mode	Row
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows Read	89549
Actual Number of Rows for All Executions	89501
Actual Number of Batches	0
Estimated I/O Cost	0,274977
Estimated Operator Cost	0,302516 (7%)
Estimated Subtree Cost	0,302516
Estimated CPU Cost	0,0275393
Estimated Number of Executions	1
Number of Executions	8
Estimated Number of Rows for All Executions	100000
Estimated Number of Rows Per Execution	100000
Estimated Number of Rows to be Read	100000
Estimated Row Size	23 B
Actual Rebinds	0
Actual Rewinds	0
Ordered	False
Node ID	21

Predicate

PROBE([Opt_Bitmap1028],[esencialDB].[dbo].[inventarioMateriales].[produccionId] as [IM].[produccionId],N'[IN ROW]')

Object

[esencialDB].[dbo].[inventarioMateriales].[IX_inventarioMateriales_inspectorId] [IM]

Output List

[esencialDB].[dbo].[inventarioMateriales].inspectorId; [esencialDB].[dbo].[inventarioMateriales].produccionId

Workload:

Hash Match

Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical Operation	Hash Match
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	90073
Actual Number of Batches	0
Estimated I/O Cost	0
Estimated Operator Cost	0,3184293 (7%)
Estimated CPU Cost	0,318434
Estimated Subtree Cost	2,11032
Estimated Number of Executions	1
Number of Executions	8
Estimated Number of Rows for All Executions	90066,1
Estimated Number of Rows Per Execution	90066,1
Estimated Row Size	23 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	32

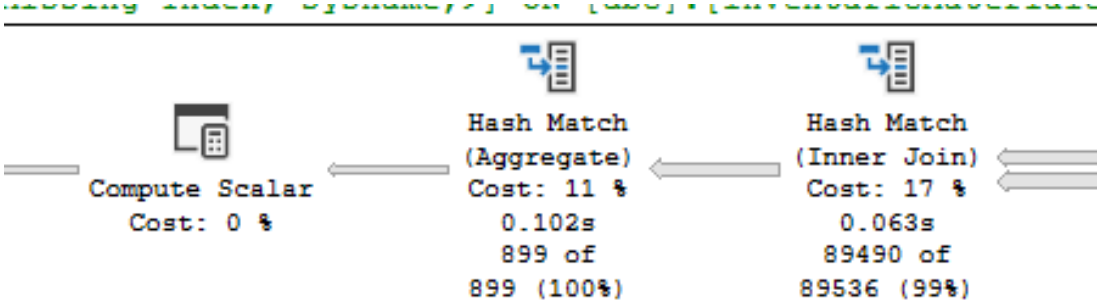
Output List
[esencialDB].[dbo].[inspectores].inspectorId; [esencialDB].[dbo].[logProducciones].cantidad
Hash Keys Probe
[esencialDB].[dbo].[inventarioMateriales].produccionId
Probe Residual
[esencialDB].[dbo].[logProducciones].[produccionId] as [P].[produccionId]=[esencialDB].[dbo].[inventarioMateriales].[produccionId] as [IM].[produccionId]

Explicación:

El motor compara el produccionId de la tabla inventarioMateriales con los produccionId de la tabla logProducciones para sacar el FK de inspectorId y la columna de cantidad respectiva, este hash match se realiza en los joins que realiza el query.

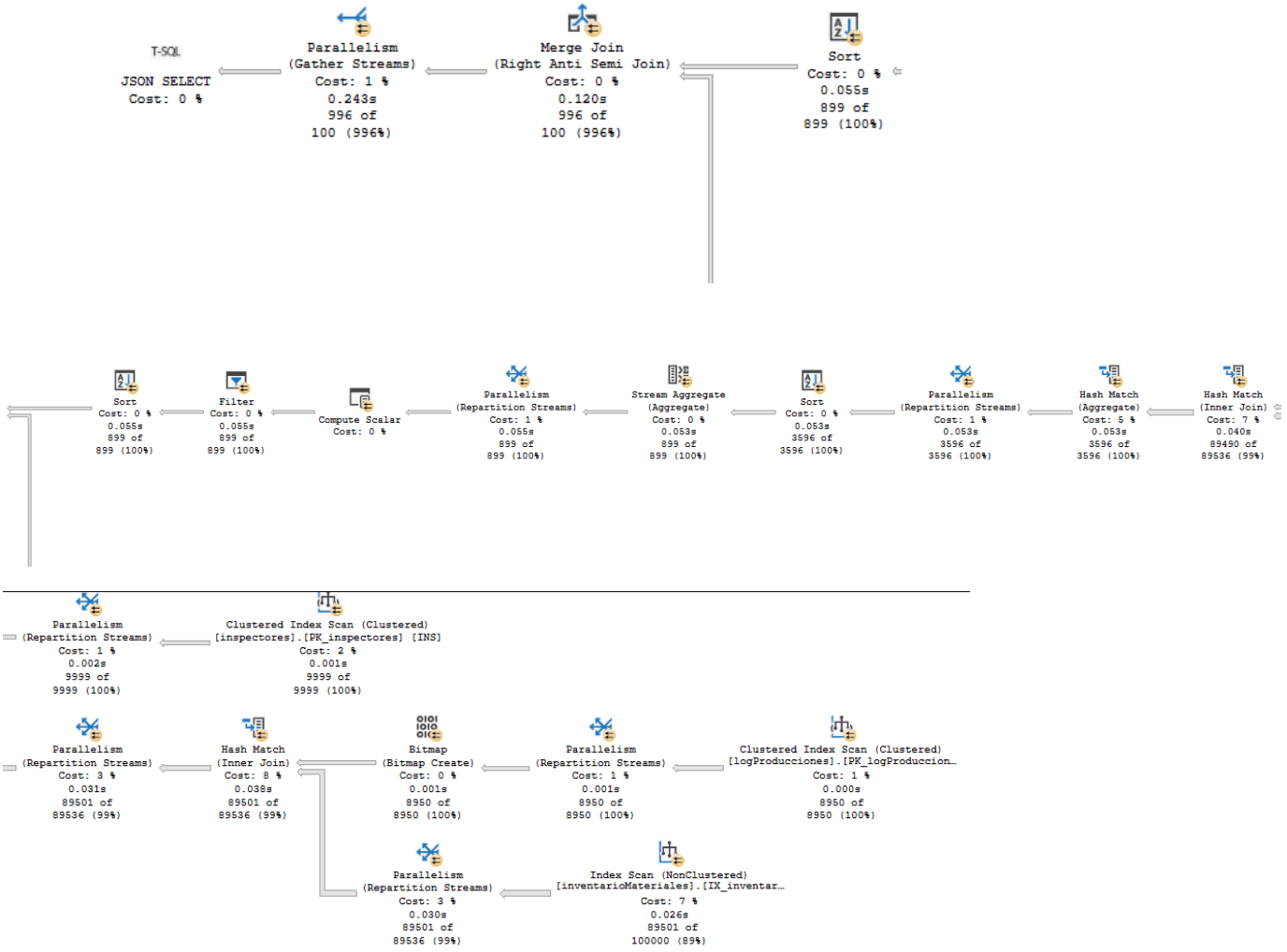
Norma:

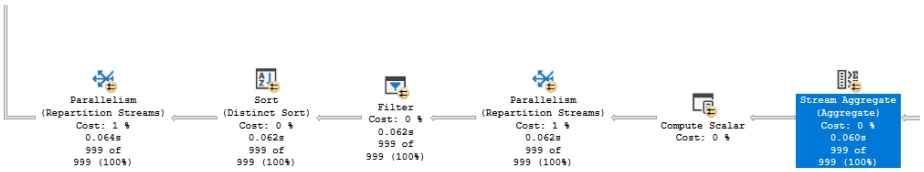
1. Los joins de las tablas con gran cantidad de registros se deben poner al final y se debe priorizar el filtrado previo de los registros para trabajar con una cantidad limitada de filas, en este caso el inner join de logProducciones debe ir de ultimo dentro del select debido a que es la tabla más grande, en caso contrario ocasiona lo siguiente:

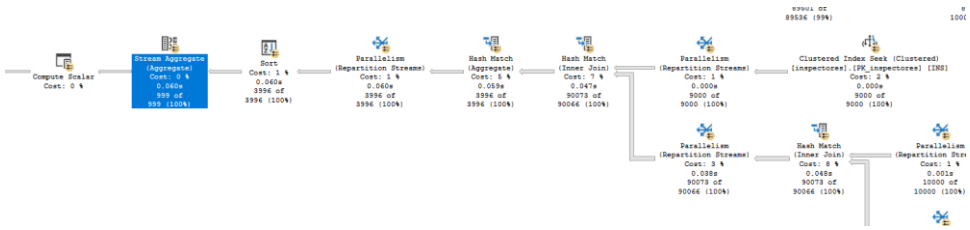


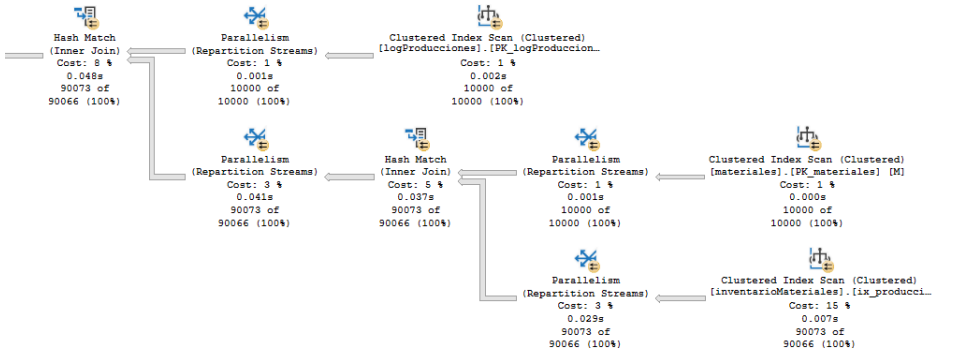
Se concluye que la manera más eficiente para trabajar en la consulta es la presentada en workload.

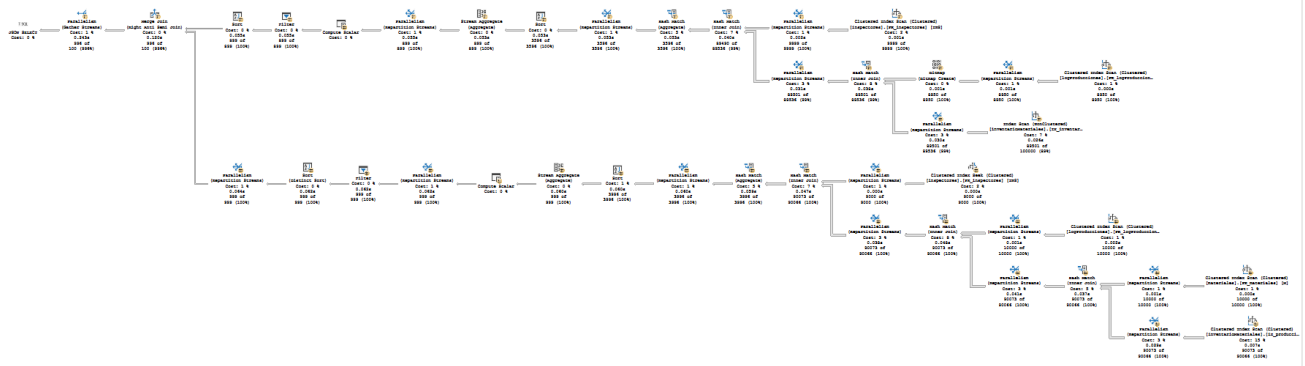
Execution Plan (Optimizado):









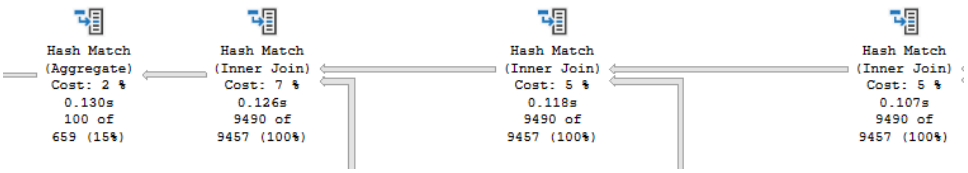


CTE’S

Query CTE:

```sql
USE [esencialDB]
GO
SET STATISTICS TIME ON
GO

WITH cte1 AS (
    SELECT
        IM.produccionId,
        P.cantidad,
        INS.inspectorId,
        M.uniMedida
    FROM dbo.inventarioMateriales IM
    LEFT JOIN dbo.inspectores INS ON INS.inspectorId = IM.inspectorId
    LEFT JOIN dbo.materiales M ON M.materialId = IM.materialId
    INNER JOIN dbo.logProducciones P ON P.produccionId = IM.produccionId
    WHERE
        INS.inspectorId > 1000 AND
        M.uniMedida = 'kg'
),
cte2 AS (
    SELECT
        IM.produccionId,
        P.cantidad,
        INS.inspectorId
    FROM dbo.inventarioMateriales IM
    LEFT JOIN dbo.inspectores INS ON INS.inspectorId = IM.inspectorId
    INNER JOIN dbo.logProducciones P ON P.produccionId = IM.produccionId
    WHERE
        P.cantidad > 100 AND
        INS.nombre != 'NombreIns:1'
)
SELECT
    -- Agregate Functions
    SUM(cte1.cantidad) AS SumaCantidad,
    STDEV(cte1.inspectorId) AS InspectorCount
FROM cte1
WHERE cte1.produccionId NOT IN (SELECT produccionId FROM cte2)
GROUP BY cte1.cantidad
HAVING SUM(cte1.cantidad) > 5
ORDER BY SUM(cte1.cantidad) DESC
FOR JSON PATH
```

Explicación rendimiento CTE:



El rendimiento del query es optimizado al dividir los selects del except en dos CTE's que permiten una ejecución más simple de los hash match's disminuyendo significativamente los costos de ejecución, se tiene una diferencia en los tiempos de ejecución del query.