TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica
Campus Tecnológico Central Cartago
Escuela De Ingeniería En Computación
Bases de datos I
I-Semestre 2023
Jueves 27 de abril
Nombre: Leonardo Céspedes
Número de Carné: 2022080602
Nombre: Frankmin Feng
Número de Carné: 2022089248

Preliminar #3 - Caso #3

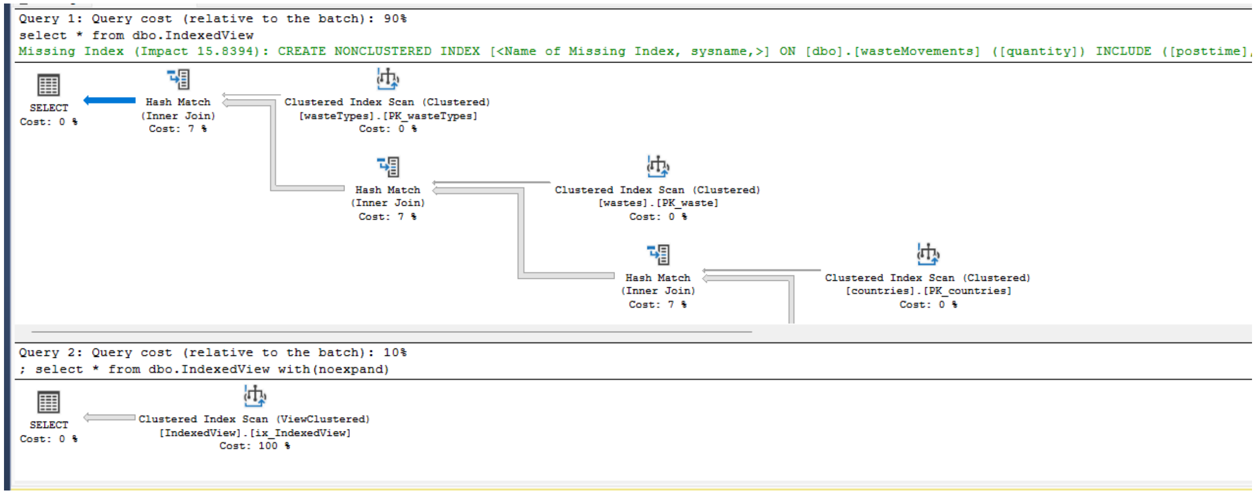
1. Basado en su diseño, si existe una consulta que requiera al menos 4 joins, cuál opción sería más eficiente: encapsular el query en una vista dinámica o en una vista indexada. Si hay diferencia encontrar una justificación teórica que justifique el hallazgo. la cantidad de datos deben ser lo suficiente para encontrar diferencias

Consideramos que sería más apropiado implementar una **vista indexada** por las siguientes razones:

1. Suele tener un mayor rendimiento, ya que la tabla para almacenar los queries no debe ser creada en el momento que se ejecuta.
2. La base de datos de este caso tiene mucha información al tratarse de un sistema internacional. La gran complejidad de los queries y la alta densidad de datos hace que una vista indexada sea más adecuada para mantener un rendimiento adecuado.
3. Se puede usar una vista indexada para precalcular los resultados de la consulta y almacenarlos en la base de datos, lo que puede acelerar significativamente el tiempo de ejecución de la consulta.

View:

```
CREATE VIEW dbo.IndexedView WITH SCHEMABINDING AS
SELECT dbo.wasteMovements.wasteMovementId, dbo.wasteMovements.posttime, dbo.wasteMovements.quantity,
dbo.containers.containerName, dbo.wastes.wasteName, dbo.wasteTypes.typeName, dbo.producers.producerName,
dbo.countries.countryName
FROM dbo.wasteMovements
INNER JOIN dbo.wastes ON dbo.wasteMovements.wasteId = dbo.wastes.wasteId
INNER JOIN dbo.wasteTypes ON dbo.wastes.wasteType = dbo.wasteTypes.wasteTypeId
INNER JOIN dbo.addresses ON dbo.wasteMovements.addressId = dbo.addresses.addressId
INNER JOIN dbo.countries ON dbo.addresses.countryId = dbo.countries.countryId
INNER JOIN dbo.containers ON dbo.wasteMovements.containerId = dbo.containers.containerId
INNER JOIN dbo.containerTypes ON dbo.containers.containerTypeId = dbo.containerTypes.containerTypeId
INNER JOIN dbo.producersXmovements ON dbo.wasteMovements.wasteMovementId =
dbo.producersXmovements.wasteMovementId
INNER JOIN dbo.producers ON dbo.producersXmovements.producerId = dbo.producers.producerId
WHERE quantity > 400;
```



```
Query 1: Query cost (relative to the batch): 90%
select * from dbo.IndexedView
Missing Index (Impact 15.8394): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sysname,>] ON [dbo].[wasteMovements] ([quantity]) INCLUDE ([posttime],
```

SELECT
Cost: 0 %
Hash Match
(Inner Join)
Cost: 7 %
Clustered Index Scan (Clustered)
[wasteTypes].[PK_wasteTypes]
Cost: 0 %
Hash Match
(Inner Join)
Cost: 7 %
Clustered Index Scan (Clustered)
[wastes].[PK_waste]
Cost: 0 %
Hash Match
(Inner Join)
Cost: 7 %
Clustered Index Scan (Clustered)
[countries].[PK_countries]
Cost: 0 %

```
Query 2: Query cost (relative to the batch): 10%
; select * from dbo.IndexedView with(noexpand)
```

SELECT
Cost: 0 %
Clustered Index Scan (ViewClustered)
[IndexedView].[ix_IndexedView]
Cost: 100 %

Statistics:

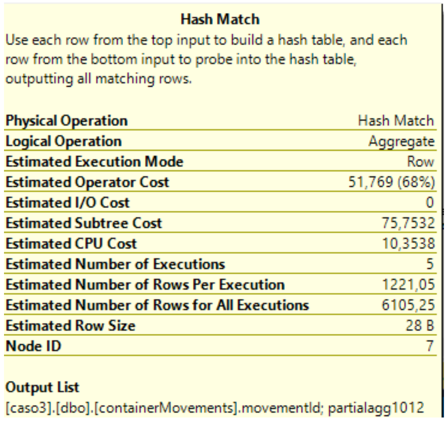
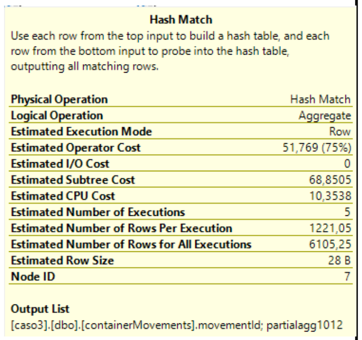
Sin clustered	(144102 rows affected) Table 'Workfile'. Scan count 0, logical reads 0, physical reads 0, page server reads

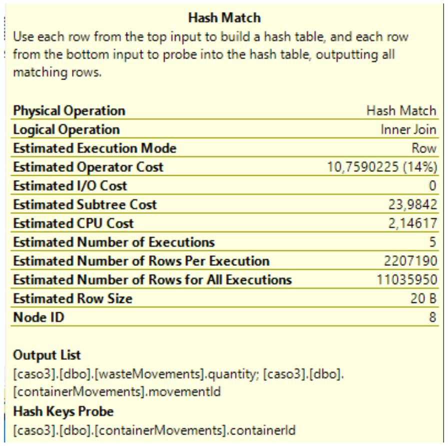
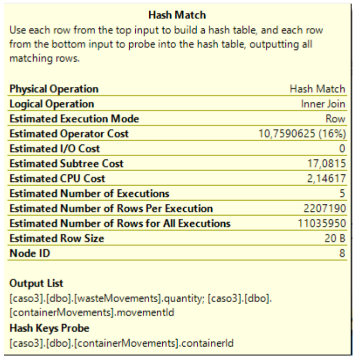
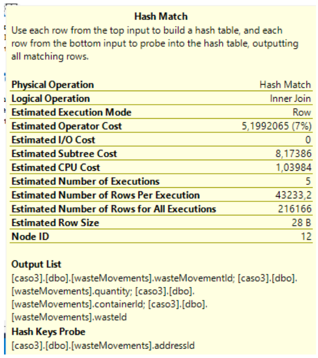
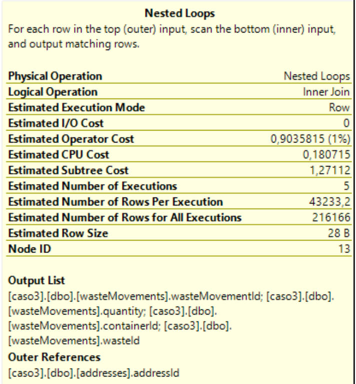
index	0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'producersXmovements'. Scan count 1, logical reads 564, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'wasteMovements'. Scan count 1, logical reads 2412, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'addresses'. Scan count 1, logical reads 3, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'producers'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'containers'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'countries'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server

	read-ahead reads 0. Table 'wastes'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0. Table 'wasteTypes'. Scan count 1, logical reads 2, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.
Con clustered index	(144102 rows affected) Table 'IndexedView'. Scan count 1, logical reads 1479, physical reads 0, page server reads 0, read-ahead reads 0, page server read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob page server reads 0, lob read-ahead reads 0, lob page server read-ahead reads 0.

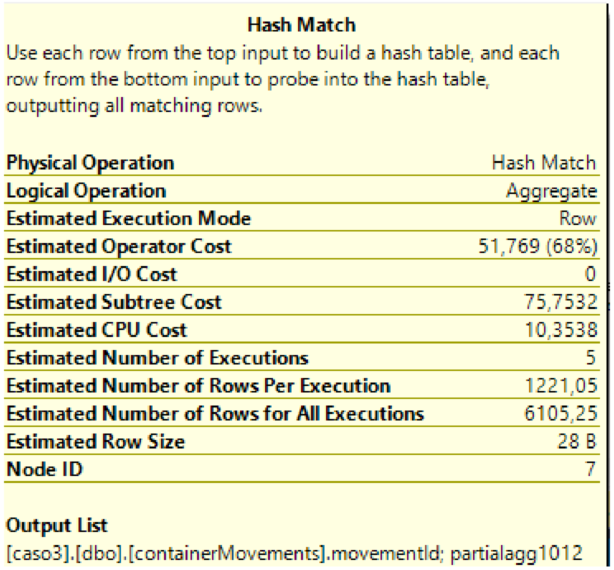
Se puede observar que sin el clustered index, se tuvieron que hacer alrededor de 3000 logical reads, mientras que en el clustered index solo se hicieron 1479.

```
SELECT
    p.producerName AS producerName,
    COUNT(DISTINCT cm.movementId) AS containerMoventCount,
    SUM(wm.quantity) AS totalWasteAmount
FROM
    producers p
LEFT JOIN
    producersXmovements pm ON p.producerId = pm.producerId
LEFT JOIN
    wasteMovements wm ON pm.wasteMovementId = wm.wasteMovementId
LEFT JOIN
    wastes w ON wm.wasteId = w.wasteId
LEFT JOIN
    containerMovements cm ON wm.containerId = cm.containerId
LEFT JOIN
    addresses ad ON wm.addressId = ad.addressId
WHERE
    ad.countryId = 1 AND
    w.wasteType = 2 AND
    cm.postime BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY
    p.producerName
HAVING
    COUNT(DISTINCT cm.movementId) > 10
ORDER BY
    totalWasteAmount DESC
FOR JSON AUTO;
```

Unidad de workload	Explicación	Norma	Optimizado
 **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** Aggregate **Estimated Execution Mode** Row **Estimated Operator Cost** 51,769 (68%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 75,7532 **Estimated CPU Cost** 10,3538 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 1221,05 **Estimated Number of Rows for All Executions** 6105,25 **Estimated Row Size** 28 B **Node ID** 7 **Output List** [caso3].[dbo].[containerMovements].movementId; partialagg1012	Cuando se ejecuta la consulta, el hash mash se usaría para unir las tablas de producers, producerXmovement, waste, wasteMovements, containerMovments y address en sus respectivos Joins. Una vez que se unen los datos, las filas resultantes se agruparían por las columnas ProducerId y ProducerName y las funciones agregadas COUNT y SUM se usarían para calcular los valores decontainerMovementCount y totalWasteAmount.	Crear un índice en las columnas provenientes a FK de las tablas de mayor tamaño utilizados en la consulta Esto ayudará al motor de la base de datos a clasificar de manera eficiente el conjunto de resultados sin tener que realizar una exploración completa de la tabla o una clasificación temporal.	 **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** Aggregate **Estimated Execution Mode** Row **Estimated Operator Cost** 51,769 (75%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 68,8505 **Estimated CPU Cost** 10,3538 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 1221,05 **Estimated Number of Rows for All Executions** 6105,25 **Estimated Row Size** 28 B **Node ID** 7 **Output List** [caso3].[dbo].[containerMovements].movementId; partialagg1012

 **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 **Estimated Execution Mode** Row **Estimated Operator Cost** 10,7590225 (14%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 23,9842 **Estimated CPU Cost** 2,14617 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 2207190 **Estimated Number of Rows for All Executions** 11035950 **Estimated Row Size** 20 B **Node ID** 8 **Output List** [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[containerMovements].movementId **Hash Keys Probe** [caso3].[dbo].[containerMovements].containerId	El engine esta utilizando de Hash Key Probe la llave de containerId, recorre cada wasteMovement y le hace hash al wasteMovementId para así juntar las tablas.	Crear un índice en las columnas provenientes a FK de las tablas de mayor tamaño utilizados en la consulta Esto ayudará al motor de la base de datos a clasificar de manera eficiente el conjunto de resultados sin tener que realizar una exploración completa de la tabla o una clasificación temporal.	 **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 **Estimated Execution Mode** Row **Estimated Operator Cost** 10,7590625 (16%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 17,0815 **Estimated CPU Cost** 2,14617 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 2207190 **Estimated Number of Rows for All Executions** 11035950 **Estimated Row Size** 20 B **Node ID** 8 **Output List** [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[containerMovements].movementId **Hash Keys Probe** [caso3].[dbo].[containerMovements].containerId
 **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 **Estimated Execution Mode** Row **Estimated Operator Cost** 5,1992065 (7%) **Estimated I/O Cost** 0 **Estimated Subtree Cost** 8,17386 **Estimated CPU Cost** 1,03984 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 43233,2 **Estimated Number of Rows for All Executions** 216166 **Estimated Row Size** 28 B **Node ID** 12 **Output List** [caso3].[dbo].[wasteMovements].wasteMovementId; [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[wasteMovements].containerId; [caso3].[dbo].[wasteMovements].wasteId **Hash Keys Probe** [caso3].[dbo].[wasteMovements].addressId	El engine esta utilizando de Hash Key Probe la FK de addressId recorre cada wasteMovement para así obtener el quantity, el containerId y el wasteId.	Crear un índice en las columnas provenientes a FK de las tablas de mayor tamaño utilizados en la consulta Esto ayudará al motor de la base de datos a clasificar de manera eficiente el conjunto de resultados sin tener que realizar una exploración completa de la tabla o una clasificación temporal.	 **Nested Loops** For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows. **Physical Operation** Nested Loops **Logical Operation** Inner Join **Estimated Execution Mode** Row **Estimated I/O Cost** 0 **Estimated Operator Cost** 0,9035815 (1%) **Estimated CPU Cost** 0,180715 **Estimated Subtree Cost** 1,27112 **Estimated Number of Executions** 5 **Estimated Number of Rows Per Execution** 43233,2 **Estimated Number of Rows for All Executions** 216166 **Estimated Row Size** 28 B **Node ID** 13 **Output List** [caso3].[dbo].[wasteMovements].wasteMovementId; [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[wasteMovements].containerId; [caso3].[dbo].[wasteMovements].wasteId **Outer References** [caso3].[dbo].[addresses].addressId

```
—-------------------------------------------------------
CREATE NONCLUSTERED INDEX [ixWasteMovment]
ON [dbo].[wasteMovements] ([addressId])
INCLUDE ([quantity],[containerId],[wasteId])
```



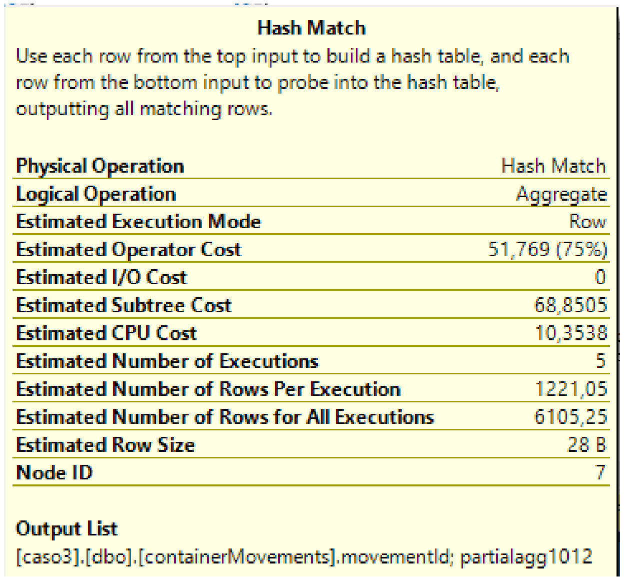
Hash Match

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	Aggregate
Estimated Execution Mode	Row
Estimated Operator Cost	51,769 (68%)
Estimated I/O Cost	0
Estimated Subtree Cost	75,7532
Estimated CPU Cost	10,3538
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	1221,05
Estimated Number of Rows for All Executions	6105,25
Estimated Row Size	28 B
Node ID	7

Output List

[caso3].[dbo].[containerMovements].movementId; partialagg1012



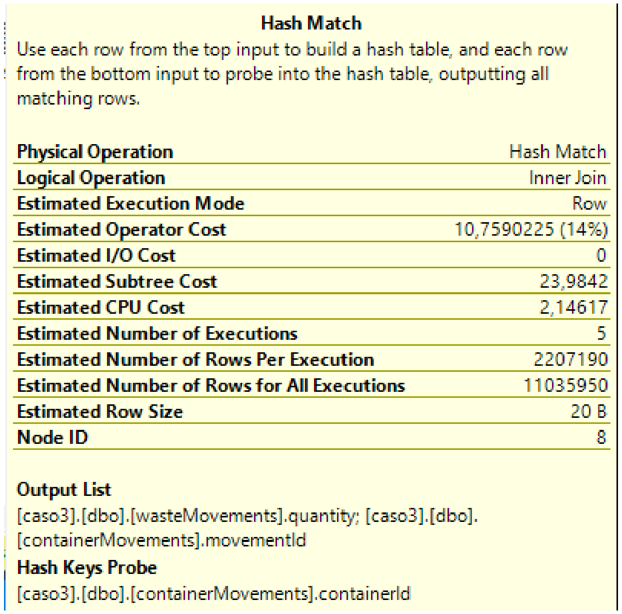
Hash Match

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	Aggregate
Estimated Execution Mode	Row
Estimated Operator Cost	51,769 (75%)
Estimated I/O Cost	0
Estimated Subtree Cost	68,8505
Estimated CPU Cost	10,3538
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	1221,05
Estimated Number of Rows for All Executions	6105,25
Estimated Row Size	28 B
Node ID	7

Output List

[caso3].[dbo].[containerMovements].movementId; partialagg1012



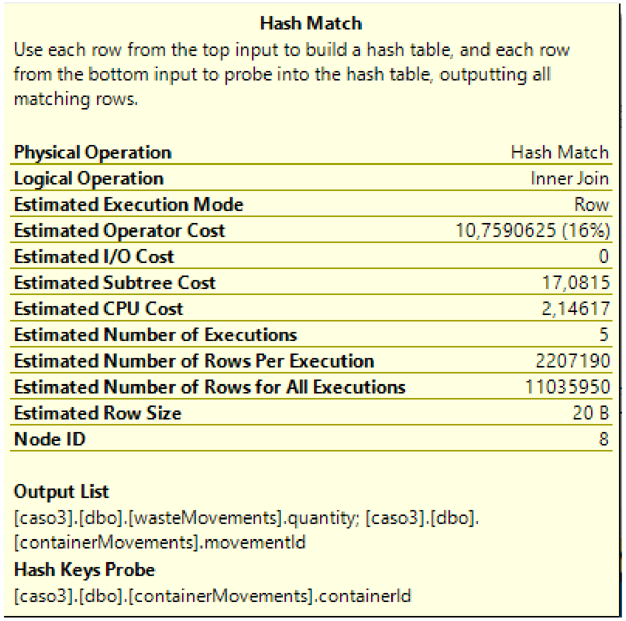
Hash Match

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
Estimated Execution Mode	Row
Estimated Operator Cost	10,7590225 (14%)
Estimated I/O Cost	0
Estimated Subtree Cost	23,9842
Estimated CPU Cost	2,14617
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	2207190
Estimated Number of Rows for All Executions	11035950
Estimated Row Size	20 B
Node ID	8

Output List
[caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[containerMovements].movementId

Hash Keys Probe
[caso3].[dbo].[containerMovements].containerId



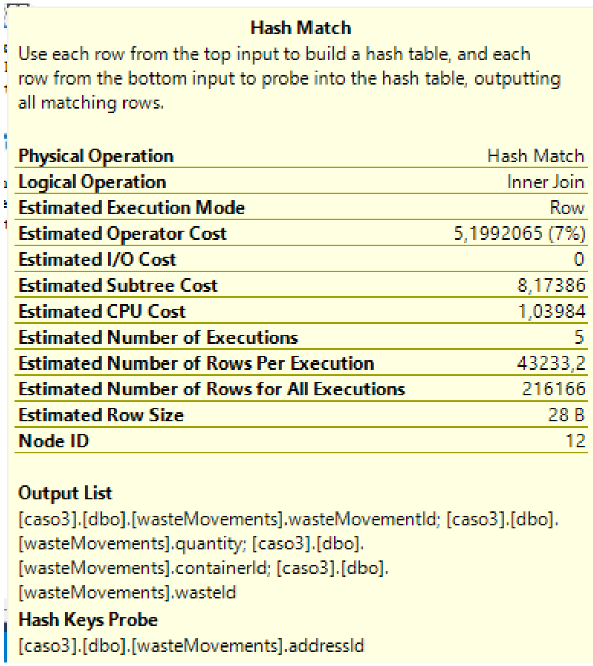
Hash Match

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
Estimated Execution Mode	Row
Estimated Operator Cost	10,7590625 (16%)
Estimated I/O Cost	0
Estimated Subtree Cost	17,0815
Estimated CPU Cost	2,14617
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	2207190
Estimated Number of Rows for All Executions	11035950
Estimated Row Size	20 B
Node ID	8

Output List
[caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[containerMovements].movementId

Hash Keys Probe
[caso3].[dbo].[containerMovements].containerId



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.

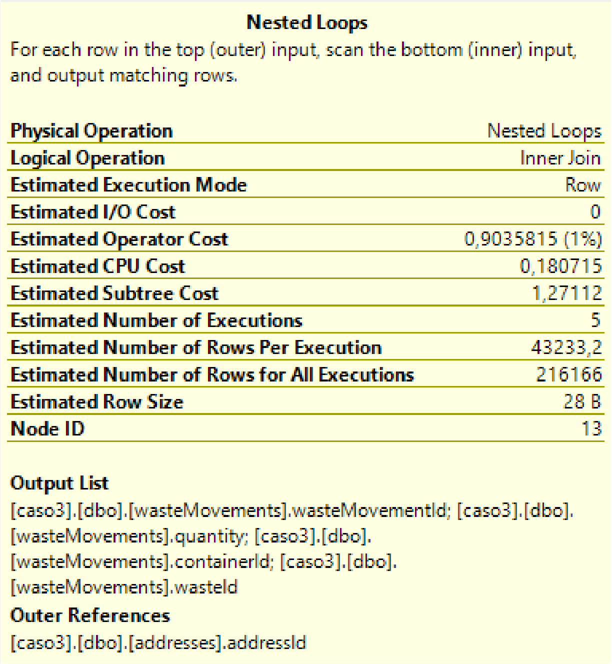
Physical Operation	Hash Match
Logical Operation	Inner Join
Estimated Execution Mode	Row
Estimated Operator Cost	5,1992065 (7%)
Estimated I/O Cost	0
Estimated Subtree Cost	8,17386
Estimated CPU Cost	1,03984
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	43233,2
Estimated Number of Rows for All Executions	216166
Estimated Row Size	28 B
Node ID	12

Output List

[caso3].[dbo].[wasteMovements].wasteMovementId; [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[wasteMovements].containerId; [caso3].[dbo].[wasteMovements].wasteId

Hash Keys Probe

[caso3].[dbo].[wasteMovements].addressId



Nested Loops

For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.

Physical Operation	Nested Loops
Logical Operation	Inner Join
Estimated Execution Mode	Row
Estimated I/O Cost	0
Estimated Operator Cost	0,9035815 (1%)
Estimated CPU Cost	0,180715
Estimated Subtree Cost	1,27112
Estimated Number of Executions	5
Estimated Number of Rows Per Execution	43233,2
Estimated Number of Rows for All Executions	216166
Estimated Row Size	28 B
Node ID	13

Output List

[caso3].[dbo].[wasteMovements].wasteMovementId; [caso3].[dbo].[wasteMovements].quantity; [caso3].[dbo].[wasteMovements].containerId; [caso3].[dbo].[wasteMovements].wasteId

Outer References

[caso3].[dbo].[addresses].addressId