Data Administration in Information Systems

2nd semester

Project Report

Teachers

Diogo Ribeiro Ferreira
Daniel Gonçalves
Rui Lopes Baeta

Students

Miguel Baltazar, 99280
Pedro Rodrigues, 99300

Shift - Group

AOBD3L05 - G44

QUESTION 1

⇛ **SQL Code:**

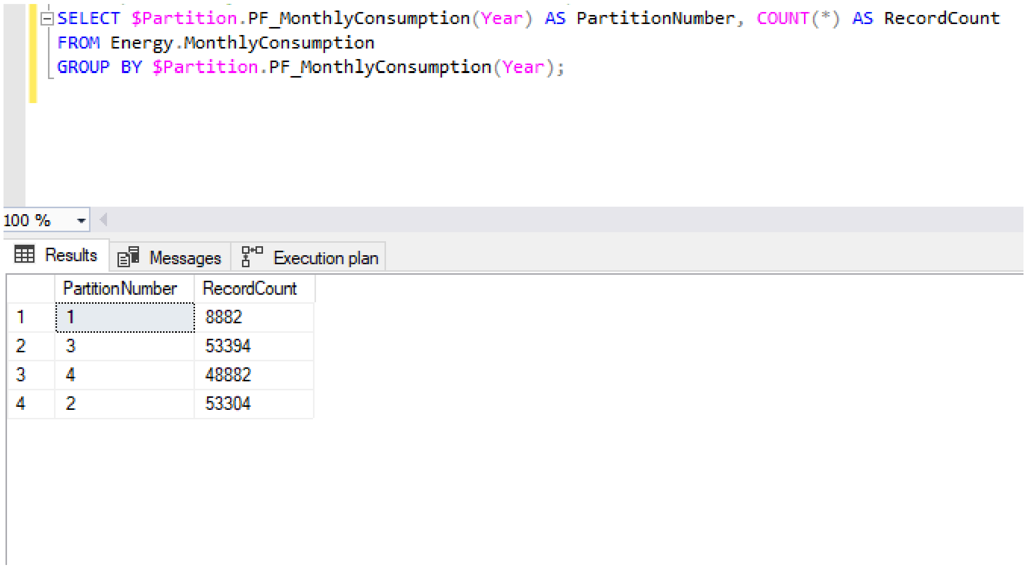
```
USE ProjectDB;
-- Create filegroups if they don't exist
IF NOT EXISTS (
    SELECT * FROM sys.filegroups
    WHERE name = 'Archive2020')
BEGIN
    ALTER DATABASE ProjectDB ADD FILEGROUP [Archive2020];
END
IF NOT EXISTS (
    SELECT * FROM sys.filegroups
    WHERE name = 'Archive2021')
BEGIN
    ALTER DATABASE ProjectDB ADD FILEGROUP [Archive2021];
END
IF NOT EXISTS (
    SELECT * FROM sys.filegroups
    WHERE name = 'Archive2022')
BEGIN
    ALTER DATABASE ProjectDB ADD FILEGROUP [Archive2022];
END
IF NOT EXISTS (
    SELECT * FROM sys.filegroups
    WHERE name = 'Archive2023')
BEGIN
    ALTER DATABASE ProjectDB ADD FILEGROUP [Archive2023];
END

ALTER DATABASE ProjectDB ADD FILE ( NAME = 'Archive2020_Data3', FILENAME =
'C:\Temp\ProjectDB_Archive2020_Data3.ndf', SIZE = 20MB, MAXSIZE =
UNLIMITED, FILEGROWTH = 10MB ) TO FILEGROUP [Archive2020];
ALTER DATABASE ProjectDB ADD FILE ( NAME = 'Archive2021_Data3', FILENAME =
'C:\Temp\ProjectDB_Archive2021_Data3.ndf', SIZE = 20MB, MAXSIZE =
UNLIMITED, FILEGROWTH = 10MB ) TO FILEGROUP [Archive2021];
ALTER DATABASE ProjectDB ADD FILE ( NAME = 'Archive2022_Data3', FILENAME =
'C:\Temp\ProjectDB_Archive2022_Data3.ndf', SIZE = 20MB, MAXSIZE =
UNLIMITED, FILEGROWTH = 10MB ) TO FILEGROUP [Archive2022];
```

```
ALTER DATABASE ProjectDB ADD FILE ( NAME = 'Archive2023_Data3', FILENAME =
'C:\Temp\ProjectDB_Archive2023_Data3.ndf', SIZE = 20MB, MAXSIZE =
UNLIMITED, FILEGROWTH = 10MB ) TO FILEGROUP [Archive2023];
IF NOT EXISTS (
   SELECT * FROM sys.partition_schemes
   WHERE function_id = OBJECT_ID('PF_MonthlyConsumption') )
BEGIN
   IF EXISTS (
      SELECT * FROM sys.partition_functions
      WHERE name = 'PF_MonthlyConsumption')
   BEGIN
      DROP PARTITION FUNCTION PF_MonthlyConsumption;
   END

   CREATE PARTITION FUNCTION PF_MonthlyConsumption (char(4)) AS RANGE LEFT
FOR
   VALUES (2020, 2021, 2022, 2023);

CREATE PARTITION SCHEME PS_MonthlyConsumption AS PARTITION
PF_MonthlyConsumption TO ([Primary], [Archive2020], [Archive2021],
[Archive2022], [Archive2023]);
END

IF NOT EXISTS (
   SELECT * FROM sys.indexes
   WHERE name = 'PK_MonthlyConsumption')
BEGIN
   ALTER TABLE Energy.MonthlyConsumption ADD CONSTRAINT
PK_MonthlyConsumption PRIMARY KEY CLUSTERED (Year)
   ON PS_MonthlyConsumption(Year);
END
```

⇒ Query to show the number of records that each partition contains:

```
SELECT
   $ Partition.PF_MonthlyConsumption(Year) AS PartitionNumber,
   COUNT(*) AS RecordCount
FROM
   Energy.MonthlyConsumption
GROUP BY
   $ Partition.PF_MonthlyConsumption(Year);
```

⇛ Query’s results:



```sql
SELECT $Partition.PF_MonthlyConsumption(Year) AS PartitionNumber, COUNT(*) AS RecordCount
FROM Energy.MonthlyConsumption
GROUP BY $Partition.PF_MonthlyConsumption(Year);
```

```
   PartitionNumber  RecordCount
1  1                8882
2  3                53394
3  4                48882
4  2                53304
```

Year	2020	2021	2022	2023
Partition Number	1	2	3	4
Record Count	**8882**	**53304**	**53394**	**48882**

QUESTION 2

⇛ SQL Code:

```sql
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy]
FROM [Energy].MonthlyConsumption]
WHERE [Municipality] = ‘Lisboa’ AND [Month] = ‘06’
GROUP BY [Parish], [Year]
ORDER BY [Parish], [Year]
```

Identify Potential Indexes:

Columns used in the WHERE clause: [Municipality], [Month]

Columns used in the GROUP BY clause: [Parish], [Year]

Columns used in the ORDER BY clause: [Parish], [Year]

Covered Index

The identified columns are **potential indexes** because they are used in key operations within the query:

- Columns used in the **WHERE** clause: Indexing columns used in the WHERE clause can improve query performance by allowing the database engine to quickly locate relevant rows based on the specified conditions. In this case, [Municipality] and [Month] are used in the WHERE clause for filtering the data.

- Columns used in the **GROUP BY** clause: When grouping data, the database engine needs to efficiently organize and aggregate rows based on the specified grouping columns. Creating an index on the columns used in the GROUP BY clause ([Parish], [Year]) can help optimize the grouping operation.

- Columns used in the **ORDER BY** clause are the same as the **GROUP BY**. Indexing columns used in the ORDER BY clause can improve sorting performance.

- A **covering index** is an index that includes all the columns needed to satisfy a query, allowing the database engine to retrieve the required data directly from the index without accessing the actual table data.
  The columns used in the WHERE clause are [Municipality] and [Month], and the columns used in the SELECT, GROUP BY, and ORDER BY clauses are [Parish], [Year], and [ActiveEnergy]. Therefore, to create a covering index for this query, all these columns should be included in the index.

⇒ SQL Code to create indexes:

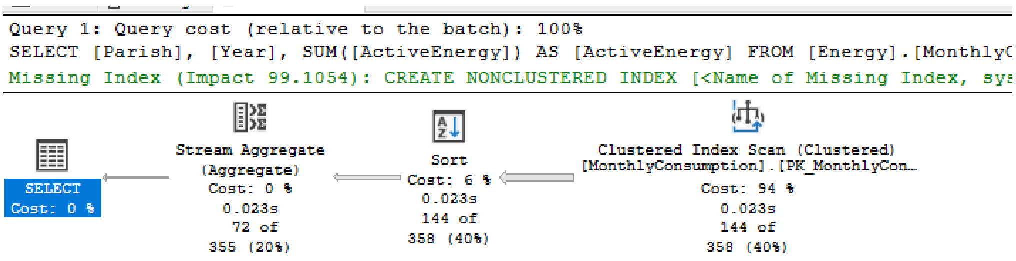
```
CREATE INDEX idx_municipality_month
ON [Energy].[MonthlyConsumption] ([Municipality], [Month]);

CREATE INDEX idx_parish_year
ON [Energy].[MonthlyConsumption] ([Parish], [Year]);

CREATE INDEX idx_covering_index
ON [Energy].[MonthlyConsumption] ([Municipality], [Month], [Parish],
[Year], [ActiveEnergy]);
```

⇒ SQL Code to delete indexes:

```
DROP INDEX idx_municipality_month ON [Energy].[MonthlyConsumption];

DROP INDEX idx_parish_year ON [Energy].[MonthlyConsumption];

DROP INDEX idx_covering_index ON [Energy].[MonthlyConsumption];
```

⇒ Executions Plans and Estimated Subtree Costs:

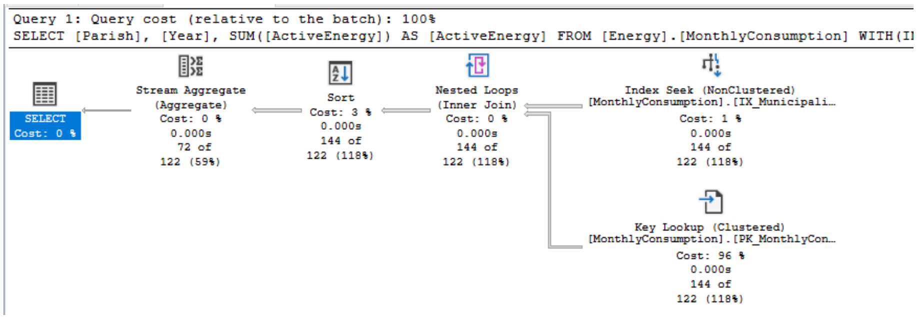
- **No Index**



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy] FROM [Energy].[MonthlyC
Missing Index (Impact 99.1054): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sys
```

Estimated Subtree Cost: *2.60689*

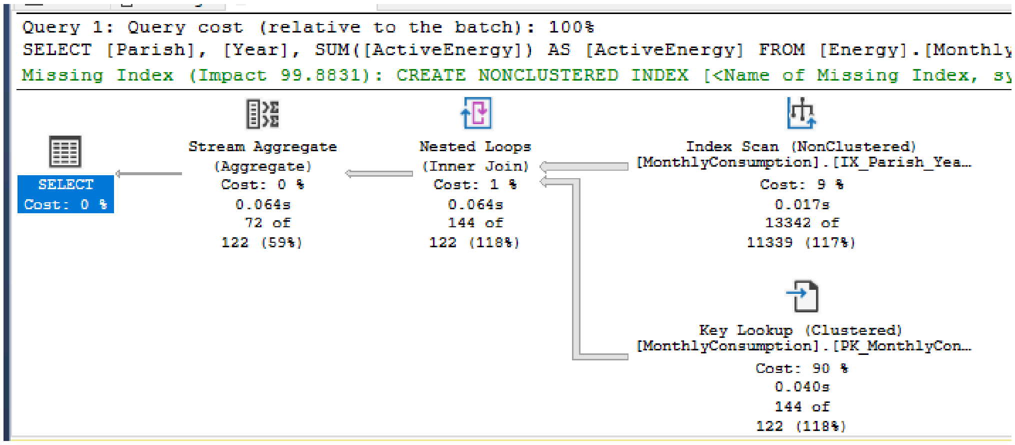
- **IX_Municipality_Month**



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy] FROM [Energy].[MonthlyConsumption] WITH(I
```

Estimated Subtree Cost: 0.4082

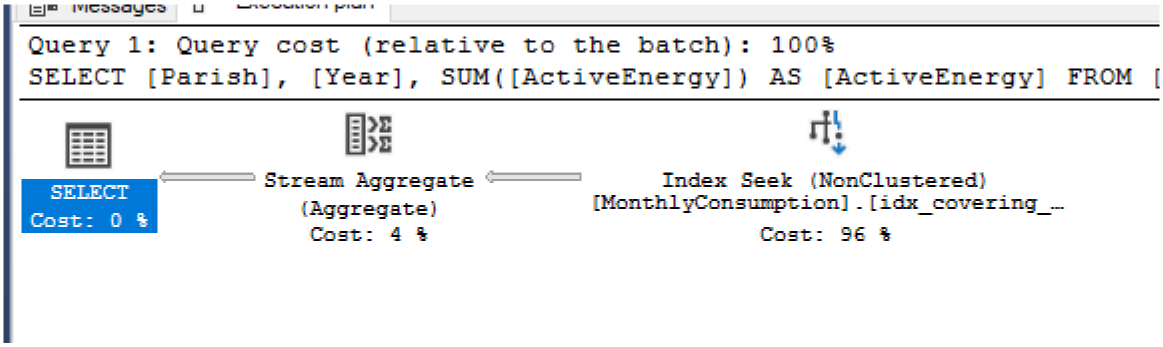
- **IX_Parish_Year**



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy] FROM [Energy].[Monthly
Missing Index (Impact 99.8831): CREATE NONCLUSTERED INDEX [<Name of Missing Index, sy
```

Estimated Subtree Cost: *12.2901*

- **idx_covering_index**



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy] FROM [
```

Estimated Subtree Cost: *0.00369*

Index	**Estimated Subtree Cost**
No index	2.60689
IX_Municipality_Month	0.4082
IX_Parish_Year	12.2901
idx_covering_index	0.00369

The **estimated subtree cost** provided by the query optimizer indicates the expected cost of executing the query subtree (execution plan) associated with each index. Lower costs generally imply better performance. Let's analyze the observed costs:

1. **No index**: With no index present, the query essentially requires a full table scan, where every row in the table must be examined to satisfy the filtering and aggregation conditions. This can be resource-intensive and slow, resulting in a relatively high estimated subtree cost of **2.60689**.

2. **IX_Municipality_Month**: Creating an index on **[Municipality]** and **[Month]** allows the query optimizer to quickly locate the rows that match the specified filter conditions. Since the **WHERE** clause filters by **[Municipality]** and **[Month]**, this index significantly reduces the number of rows that need to be examined, resulting in a much lower estimated subtree cost of **0.4082** compared to the case with no index. However, this index does not cover the columns used in the **SELECT** or **GROUP BY** clauses, so additional lookups might be required to fetch those columns.

3. **IX_Parish_Year**: Creating an index on **[Parish]** and **[Year]** may not significantly improve performance in this specific query because **sorting** and **grouping** operations can still be performed efficiently without an index, especially since the number of **distinct** values in **Parish** and **Year** is not very high. Therefore, although the index might still be utilized, it may not be as effective in reducing the number of rows that need to be

examined. As a result, the estimated subtree cost is relatively high at **12.2901**, indicating poorer performance compared to the case with no index.

4. **idx_covering_index**: The covering index on **[Municipality]**, **[Month]**, **[Parish]**, **[Year]**, and **[ActiveEnergy]** covers all the columns used in the query, including those in the **SELECT**, **GROUP BY**, and **ORDER BY** clauses. As a result, the query optimizer can fully satisfy the query requirements by accessing only the index, without needing to perform additional lookups or access the actual table data. This leads to the lowest estimated subtree cost of **0.00369**, indicating the best performance among the tested scenarios.

In summary, the observed differences in estimated subtree costs show how important is creating indexes that align with the query's filter conditions and column usage.

QUESTION 3

⇛ SQL Code:

```sql
USE ProjectDB;

SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District],
[Energy].[Municipality], [Energy].[Parish], [Energy].[ActiveEnergy],
[Contracts].[NumberContracts], [Energy].[ActiveEnergy] /
[Contracts].[NumberContracts] AS EnergyPerContract
FROM
	( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS [ActiveEnergy]
	FROM [Energy].[MonthlyConsumption]
	GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Energy],
	( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([NumberContracts]) AS [NumberContracts]
	FROM [Energy].[ActiveContracts]
	GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Contracts]
WHERE [Energy].[DistrictMunicipalityParishCode] =
 [Contracts].[DistrictMunicipalityParishCode]
ORDER BY  [Energy].[District], [Energy].[Municipality], [Energy].[Parish]
```

⇛ Join Algorithms

- Nested Loops
- Merge Join
- Hash Match

⇛ Group-by Algorithms

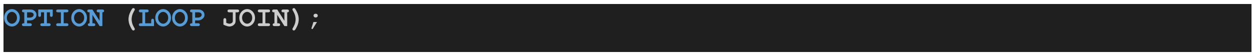
- Stream Aggregate
- Hash Aggregate

The following pairs of one group-by and one join algorithm will be tested:

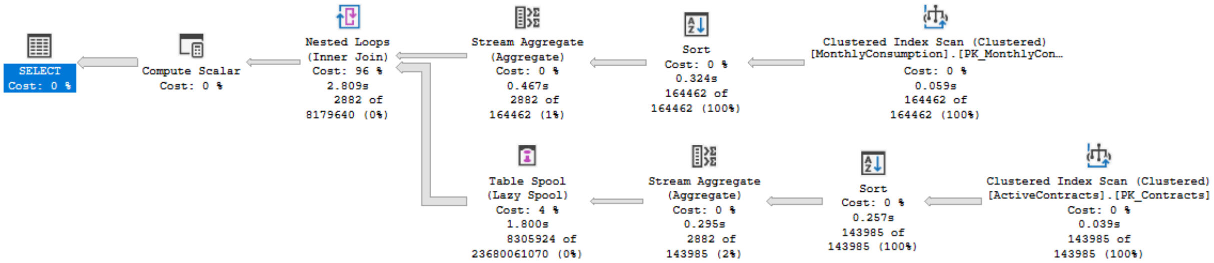
- ➔ 1.1 Nested Loops (Inner Join) + Stream Aggregate
- ➔ 1.2 Nested Loops (Inner Join) + Hash Match (Aggregate)
- ➔ 2.1 Merge Join (Inner Join) + Stream Aggregate
- ➔ 2.2 Merge Join (Inner Join) + Hash Match (Aggregate)
- ➔ 3.1 Hash Match (Inner Join) + Stream Aggregate
- ➔ 3.2 Hash Match (Inner Join) + Hash Match (Aggregate)

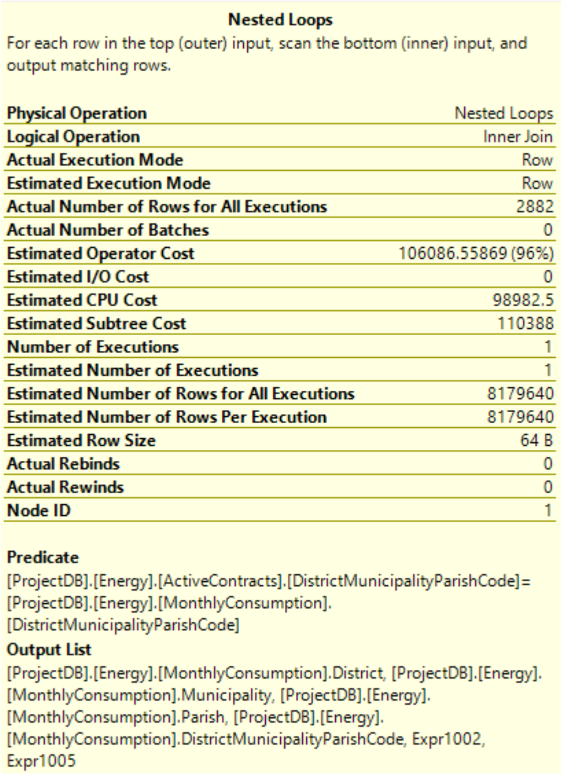
⇛ 1.1 Nested Loops (Inner Join) + Stream Aggregate

Please add this line to SQL Code:



```
OPTION (LOOP JOIN);
```





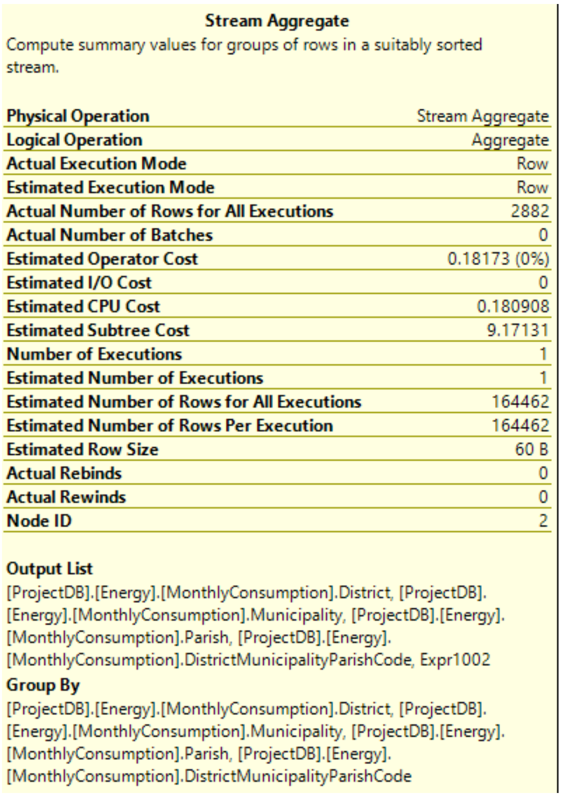
Nested Loops

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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	106086.55869 (96%)
Estimated I/O Cost	0
Estimated CPU Cost	98982.5
Estimated Subtree Cost	110388
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	8179640
Estimated Number of Rows Per Execution	8179640
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	1

Predicate
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005



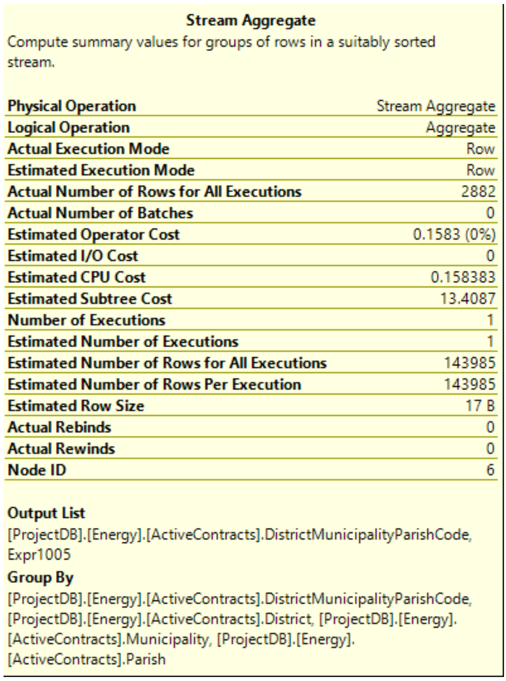
Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	0.18173 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.180908
Estimated Subtree Cost	9.17131
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	164462
Estimated Number of Rows Per Execution	164462
Estimated Row Size	60 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002

Group By
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode



Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	0.1583 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.158383
Estimated Subtree Cost	13.4087
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	143985
Estimated Number of Rows Per Execution	143985
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	6

Output List
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005

Group By
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[ActiveContracts].District, [ProjectDB].[Energy].[ActiveContracts].Municipality, [ProjectDB].[Energy].[ActiveContracts].Parish

⇛ 1.2 Nested Loops (Inner Join) + Hash Match (Aggregate)

Before running SQL Code, create indexes:

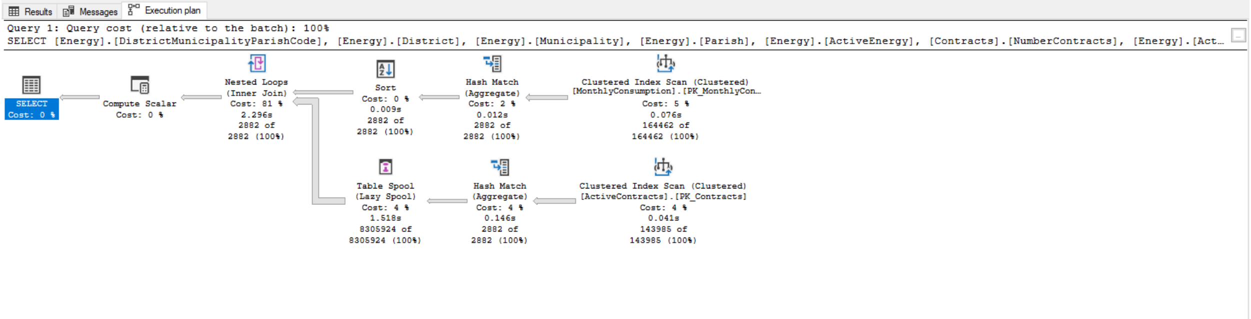
```
CREATE INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption
(DistrictMunicipalityParishCode, District, Municipality, Parish);
CREATE INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts
(DistrictMunicipalityParishCode, District, Municipality, Parish);
```

Please add this line to SQL Code:

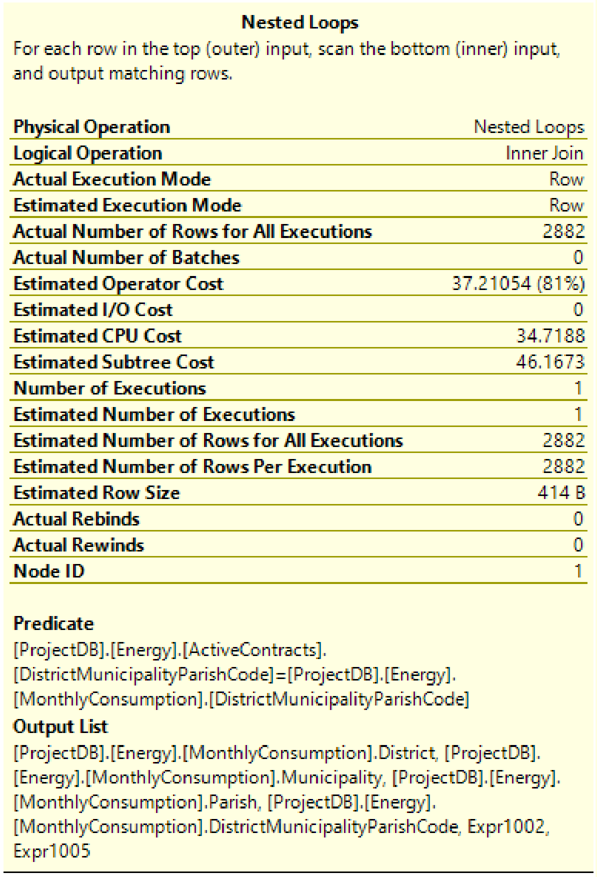
```
OPTION (LOOP JOIN);
```

After running SQL Code, delete indexes:

```
DROP INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption;
DROP INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts;
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality], [Energy].[Parish], [Energy].[ActiveEnergy], [Contracts].[NumberContracts], [Energy].[Act…
```



Nested Loops

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

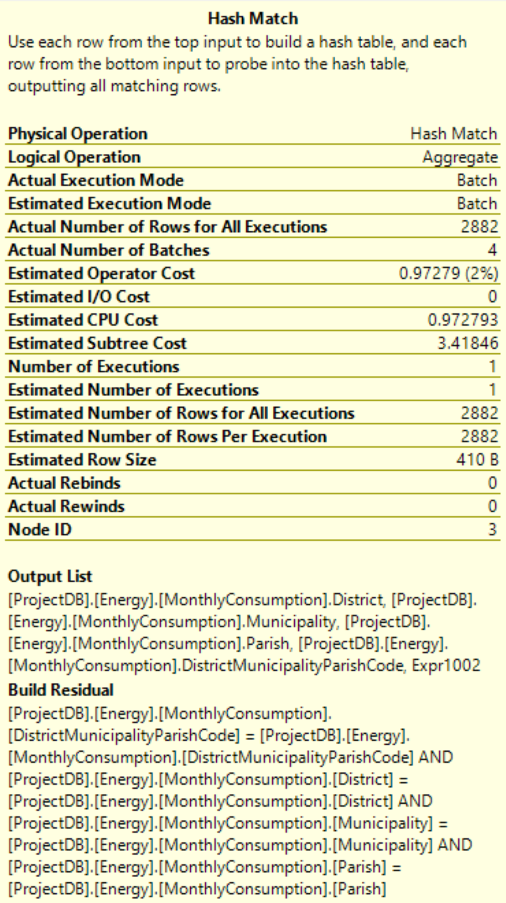
Physical Operation	Nested Loops
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	37.21054 (81%)
Estimated I/O Cost	0
Estimated CPU Cost	34.7188
Estimated Subtree Cost	46.1673
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	414 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	1

Predicate

[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

Output List

[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005



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.

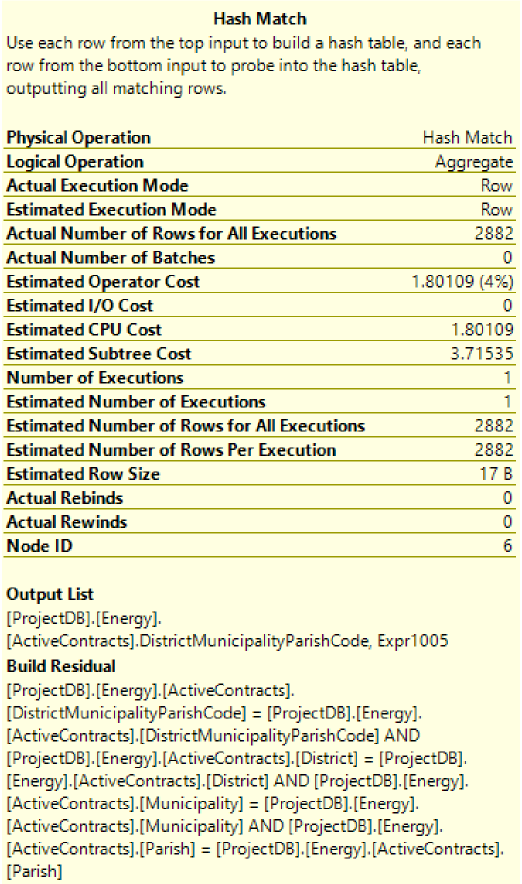
Physical Operation	Hash Match
Logical Operation	Aggregate
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.97279 (2%)
Estimated I/O Cost	0
Estimated CPU Cost	0.972793
Estimated Subtree Cost	3.41846
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	410 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	3

Output List

[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002

Build Residual

[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[MonthlyConsumption].[District] = [ProjectDB].[Energy].[MonthlyConsumption].[District] AND [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] = [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] AND [ProjectDB].[Energy].[MonthlyConsumption].[Parish] = [ProjectDB].[Energy].[MonthlyConsumption].[Parish]



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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	1.80109 (4%)
Estimated I/O Cost	0
Estimated CPU Cost	1.80109
Estimated Subtree Cost	3.71535
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	6

Output List

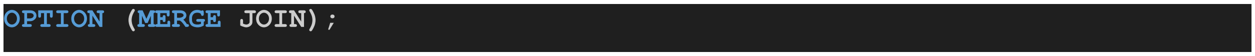
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005

Build Residual

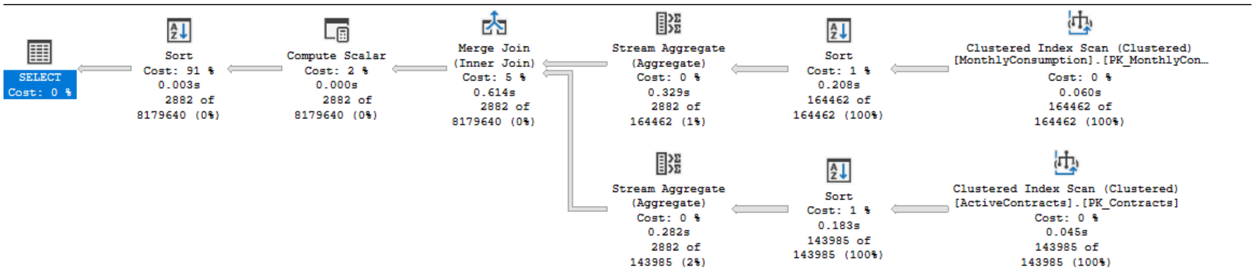
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[ActiveContracts].[District] = [ProjectDB].[Energy].[ActiveContracts].[District] AND [ProjectDB].[Energy].[ActiveContracts].[Municipality] = [ProjectDB].[Energy].[ActiveContracts].[Municipality] AND [ProjectDB].[Energy].[ActiveContracts].[Parish] = [ProjectDB].[Energy].[ActiveContracts].[Parish]

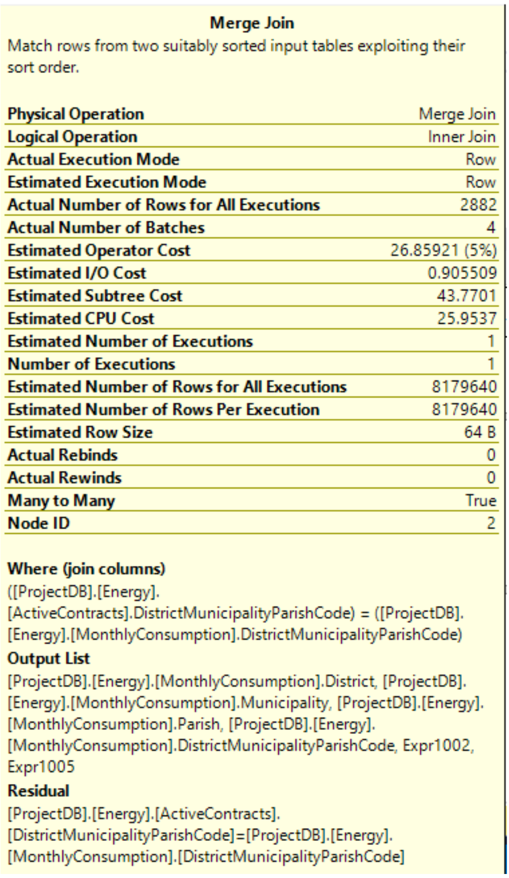
⇛ 2.1 Merge Join (Inner Join) + Stream Aggregate

Please add this line to SQL Code:



```sql
OPTION (MERGE JOIN);
```



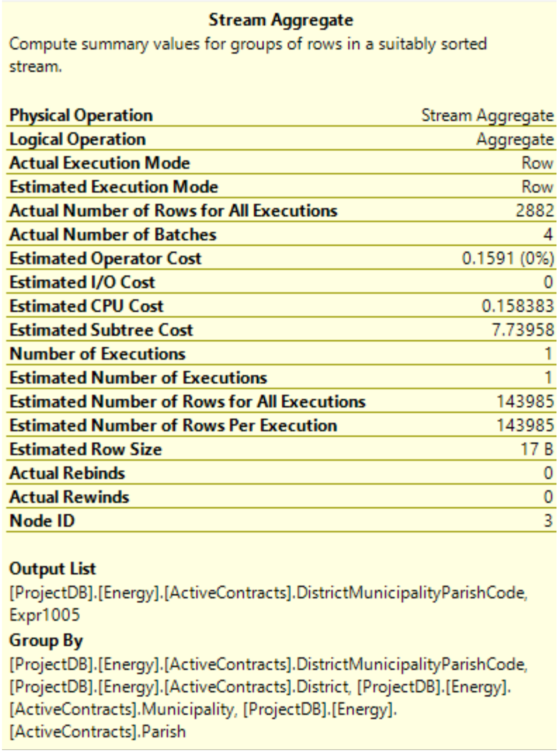


Merge Join

Match rows from two suitably sorted input tables exploiting their sort order.

Physical Operation	Merge Join
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	26.85921 (5%)
Estimated I/O Cost	0.905509
Estimated Subtree Cost	43.7701
Estimated CPU Cost	25.9537
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	8179640
Estimated Number of Rows Per Execution	8179640
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Many to Many	True
Node ID	2

Where (join columns)
([ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode) = ([ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode)
Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005
Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

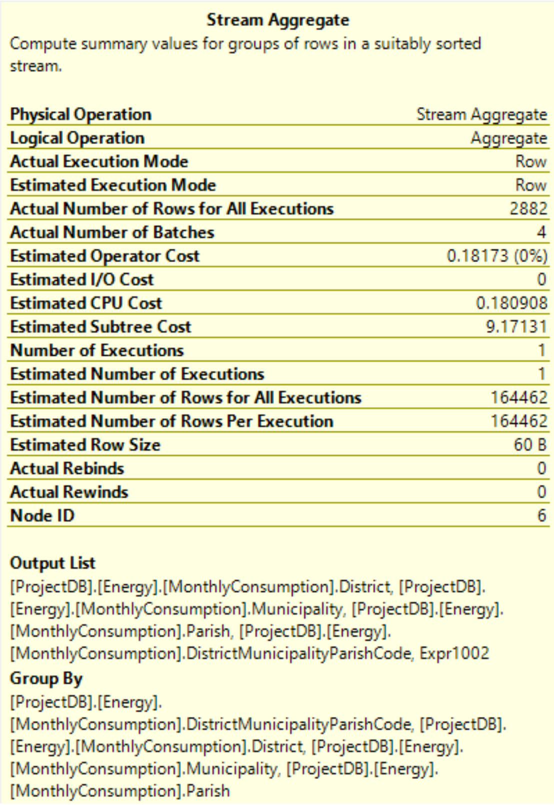


Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.1591 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.158383
Estimated Subtree Cost	7.73958
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	143985
Estimated Number of Rows Per Execution	143985
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	3

Output List
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005
Group By
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[ActiveContracts].District, [ProjectDB].[Energy].[ActiveContracts].Municipality, [ProjectDB].[Energy].[ActiveContracts].Parish



Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.18173 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.180908
Estimated Subtree Cost	9.17131
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	164462
Estimated Number of Rows Per Execution	164462
Estimated Row Size	60 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	6

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002
Group By
[ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish

⇒ 2.2 Merge Join (Inner Join) + Hash Match (Aggregate)

Before running SQL Code, create indexes:

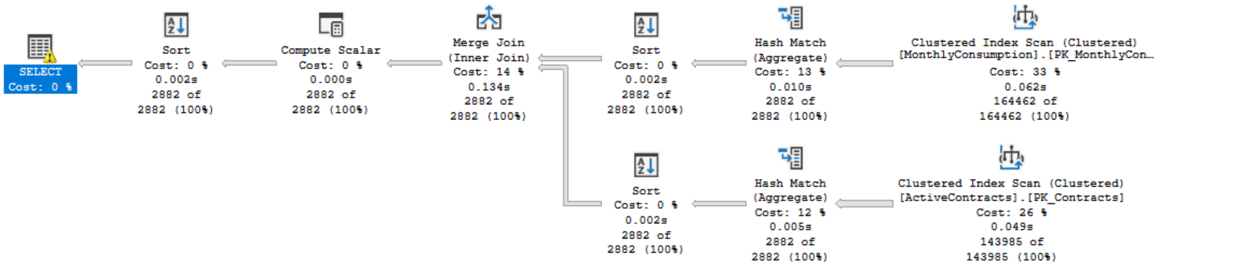
```
CREATE INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption
(DistrictMunicipalityParishCode, District, Municipality, Parish);
CREATE INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts
(DistrictMunicipalityParishCode, District, Municipality, Parish);
```

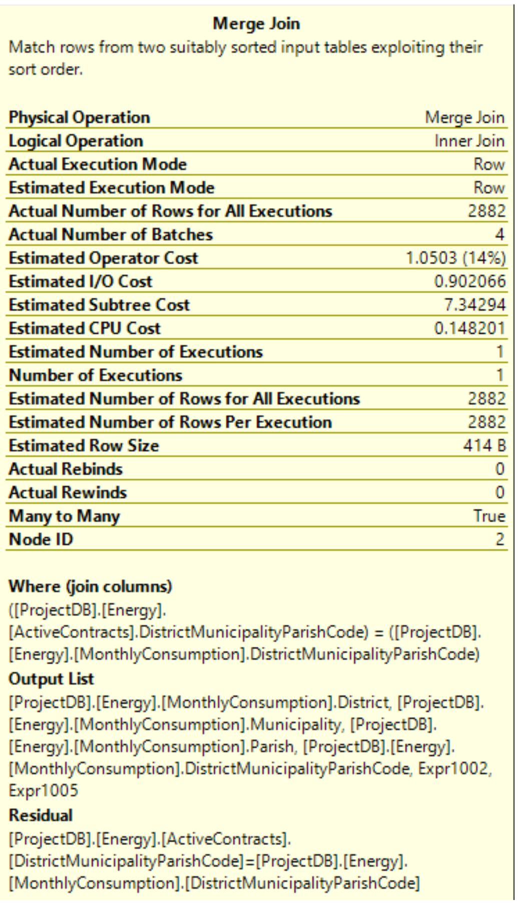
Please add this line to SQL Code:

```
OPTION (MERGE JOIN);
```

After running SQL Code, delete indexes:

```
DROP INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption;
DROP INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts;
```



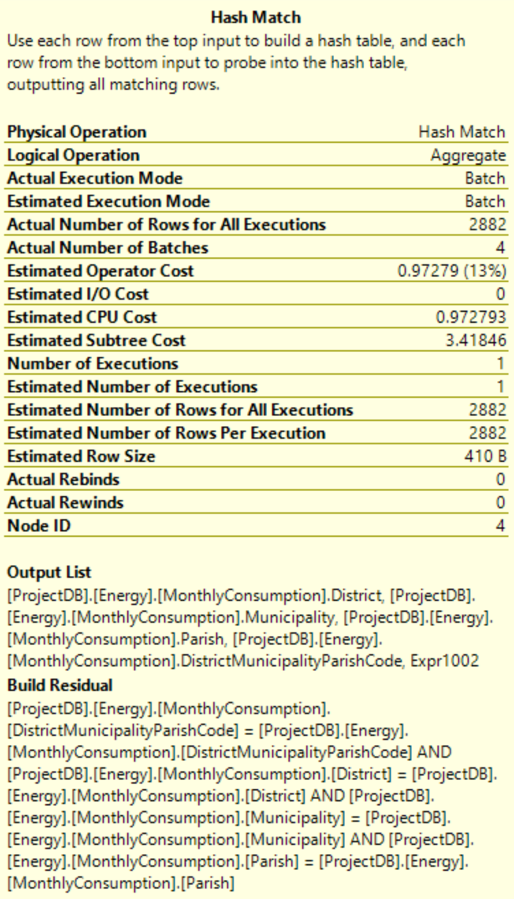


Merge Join

Match rows from two suitably sorted input tables exploiting their sort order.

Physical Operation	Merge Join
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	1.0503 (14%)
Estimated I/O Cost	0.902066
Estimated Subtree Cost	7.34294
Estimated CPU Cost	0.148201
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	414 B
Actual Rebinds	0
Actual Rewinds	0
Many to Many	True
Node ID	2

Where (join columns)
([ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode) = ([ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode)
Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005
Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

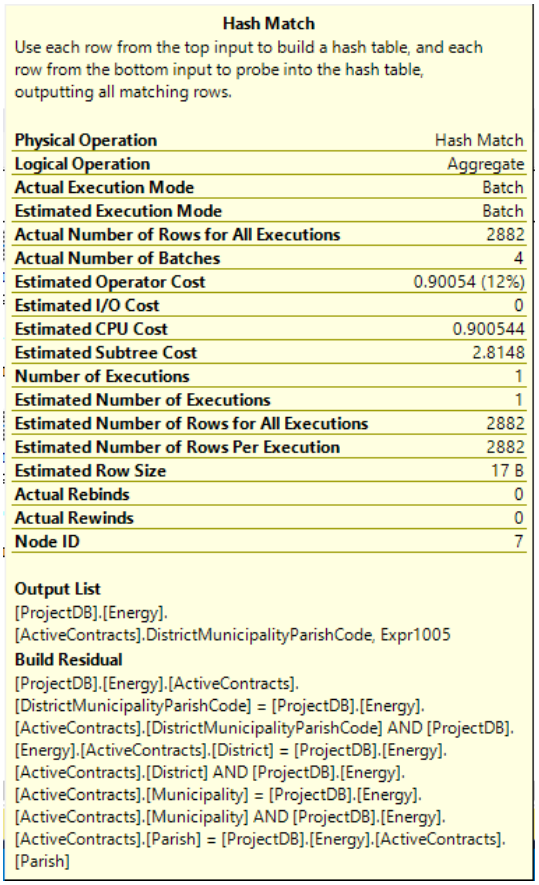


Hash Match

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
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.97279 (13%)
Estimated I/O Cost	0
Estimated CPU Cost	0.972793
Estimated Subtree Cost	3.41846
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	410 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	4

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002
Build Residual
[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[MonthlyConsumption].[District] = [ProjectDB].[Energy].[MonthlyConsumption].[District] AND [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] = [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] AND [ProjectDB].[Energy].[MonthlyConsumption].[Parish] = [ProjectDB].[Energy].[MonthlyConsumption].[Parish]



Hash Match

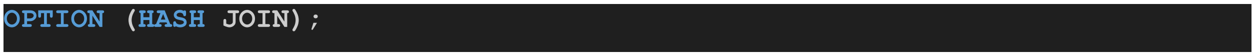
Use each row from the top input to build a hash table, and each row from the bottom input to probe into the hash table, outputting all matching rows.

Physical Operation	Hash Match
Logical Operation	Aggregate
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.90054 (12%)
Estimated I/O Cost	0
Estimated CPU Cost	0.900544
Estimated Subtree Cost	2.8148
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	7

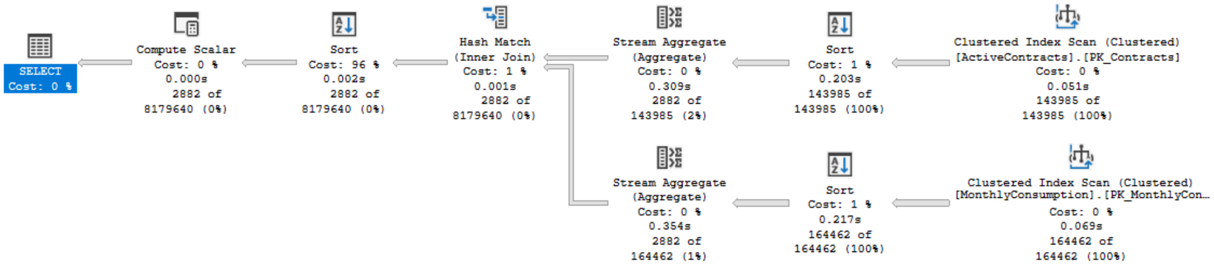
Output List
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005
Build Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[ActiveContracts].[District] = [ProjectDB].[Energy].[ActiveContracts].[District] AND [ProjectDB].[Energy].[ActiveContracts].[Municipality] = [ProjectDB].[Energy].[ActiveContracts].[Municipality] AND [ProjectDB].[Energy].[ActiveContracts].[Parish] = [ProjectDB].[Energy].[ActiveContracts].[Parish]

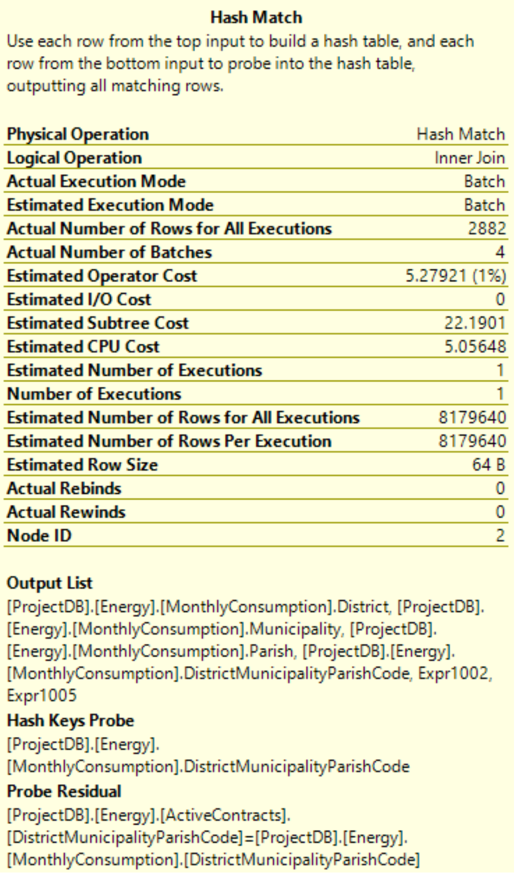
⇛ 3.1 Hash Match (Inner Join) + Stream Aggregate

Please add this line to SQL Code:



```
OPTION (HASH JOIN);
```



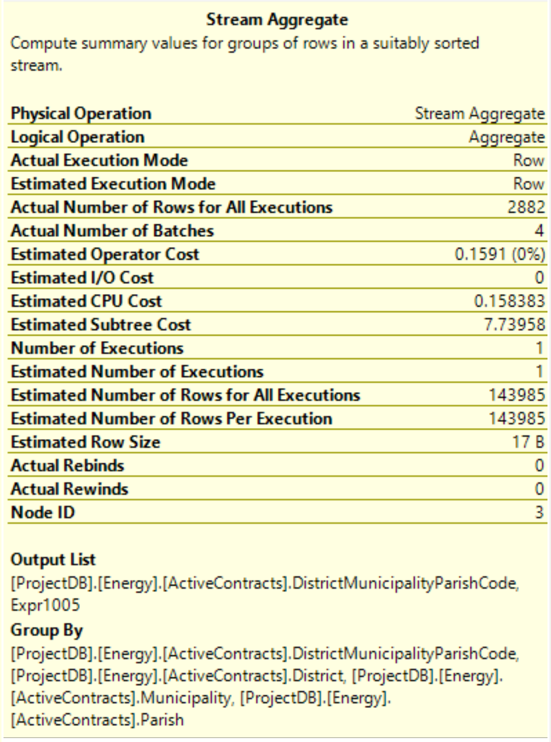


Hash Match

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	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	5.27921 (1%)
Estimated I/O Cost	0
Estimated Subtree Cost	22.1901
Estimated CPU Cost	5.05648
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	8179640
Estimated Number of Rows Per Execution	8179640
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005
Hash Keys Probe
[ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode
Probe Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

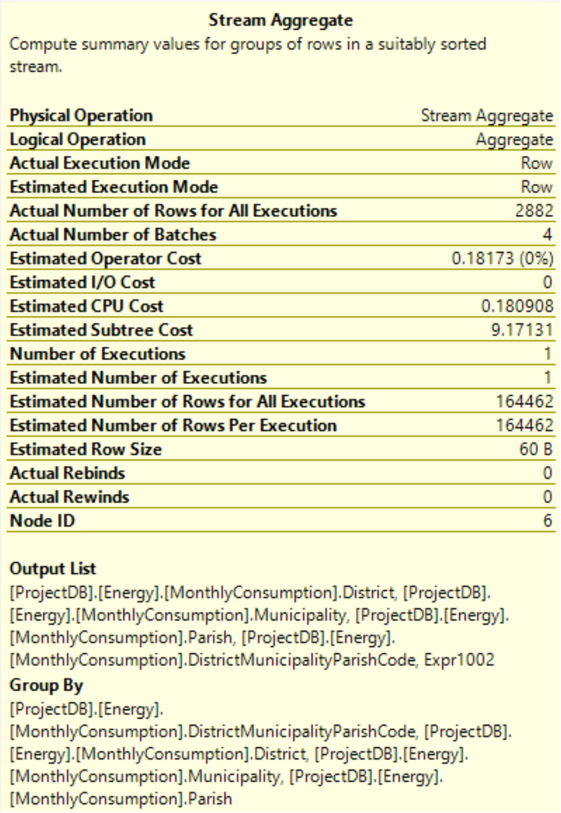


Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.1591 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.158383
Estimated Subtree Cost	7.73958
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	143985
Estimated Number of Rows Per Execution	143985
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	3

Output List
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005
Group By
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[ActiveContracts].District, [ProjectDB].[Energy].[ActiveContracts].Municipality, [ProjectDB].[Energy].[ActiveContracts].Parish



Stream Aggregate

Compute summary values for groups of rows in a suitably sorted stream.

Physical Operation	Stream Aggregate
Logical Operation	Aggregate
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.18173 (0%)
Estimated I/O Cost	0
Estimated CPU Cost	0.180908
Estimated Subtree Cost	9.17131
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	164462
Estimated Number of Rows Per Execution	164462
Estimated Row Size	60 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	6

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002
Group By
[ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish

⇛ 3.2 Hash Match (Inner Join) + Hash Match (Aggregate)

Before running SQL Code, create indexes:

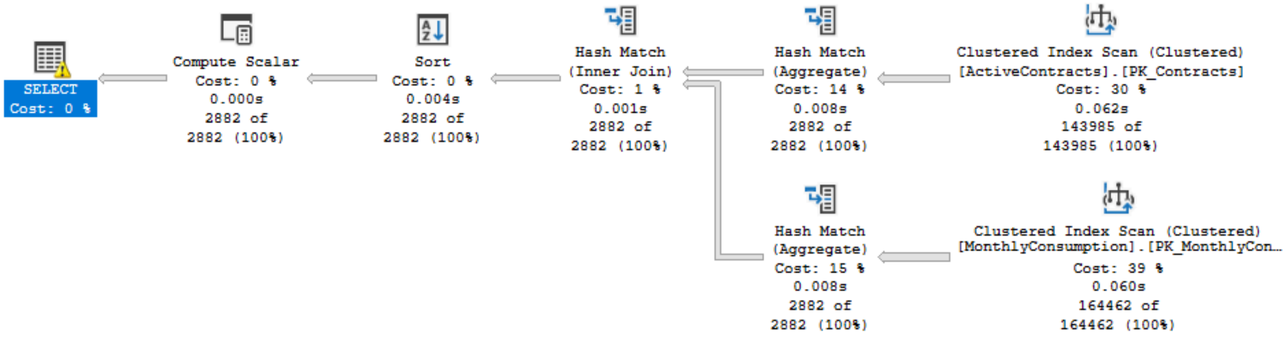
```
CREATE INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption
(DistrictMunicipalityParishCode, District, Municipality, Parish);
CREATE INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts
(DistrictMunicipalityParishCode, District, Municipality, Parish);
```

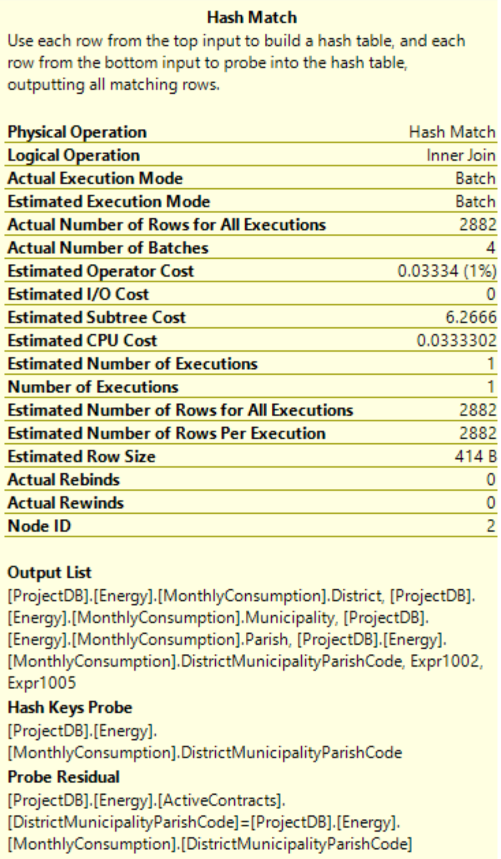
Please add this line to SQL Code:

```
OPTION (HASH JOIN);
```

After running SQL Code, delete indexes:

```
DROP INDEX IX_MonthlyConsumption_Grouping ON Energy.MonthlyConsumption;
DROP INDEX IX_ActiveContracts_Grouping ON Energy.ActiveContracts;
```



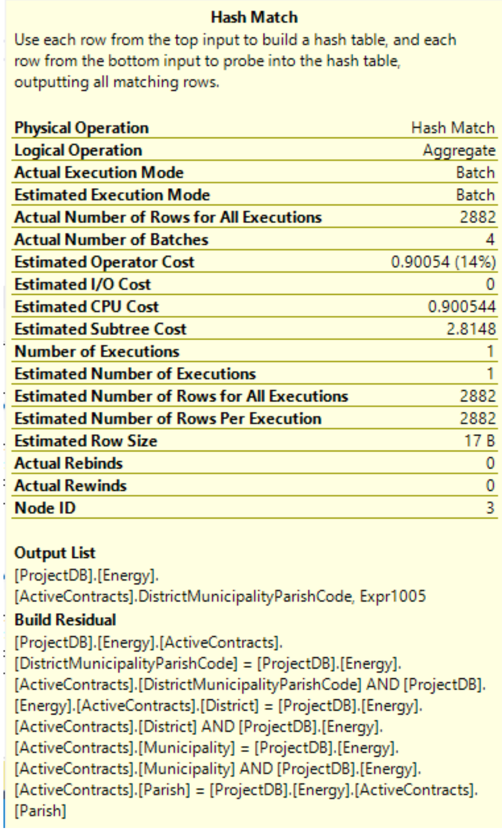


Hash Match

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	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.03334 (1%)
Estimated I/O Cost	0
Estimated Subtree Cost	6.2666
Estimated CPU Cost	0.0333302
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	414 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005
Hash Keys Probe
[ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode
Probe Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

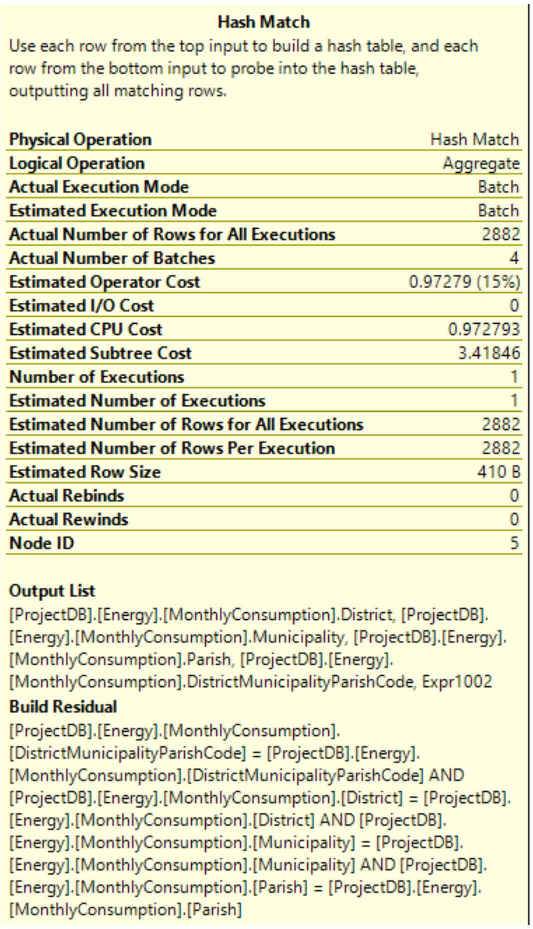


Hash Match

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
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.90054 (14%)
Estimated I/O Cost	0
Estimated CPU Cost	0.900544
Estimated Subtree Cost	2.8148
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	17 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	3

Output List
[ProjectDB].[Energy].[ActiveContracts].DistrictMunicipalityParishCode, Expr1005
Build Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[ActiveContracts].[District] = [ProjectDB].[Energy].[ActiveContracts].[District] AND [ProjectDB].[Energy].[ActiveContracts].[Municipality] = [ProjectDB].[Energy].[ActiveContracts].[Municipality] AND [ProjectDB].[Energy].[ActiveContracts].[Parish] = [ProjectDB].[Energy].[ActiveContracts].[Parish]



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
Actual Execution Mode	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	0.97279 (15%)
Estimated I/O Cost	0
Estimated CPU Cost	0.972793
Estimated Subtree Cost	3.41846
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	2882
Estimated Number of Rows Per Execution	2882
Estimated Row Size	410 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	5

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002
Build Residual
[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] = [ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode] AND [ProjectDB].[Energy].[MonthlyConsumption].[District] = [ProjectDB].[Energy].[MonthlyConsumption].[District] AND [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] = [ProjectDB].[Energy].[MonthlyConsumption].[Municipality] AND [ProjectDB].[Energy].[MonthlyConsumption].[Parish] = [ProjectDB].[Energy].[MonthlyConsumption].[Parish]

Join ↦ Group-by ↴	Nested Loops	Merge Join	Hash Match
Stream Aggregate	↥ **110388**	↥ **25.9537**	↥ **22.1901**
	↤ **22.58001**	↤ **16.91089**	↤ **16.91089**
Hash Match	↥ **46.1673**	↥ **7.34294**	↥ **6.2666**
	↤ **7.13381**	↤ **6.2333**	↤ **6.2333**

Best Estimated Subtree Cost for *Join Algorithm* with Stream Aggregate: 22.1901 (HASH MATCH - INNER JOIN)
Best Estimated Subtree Cost for *Join Algorithm* with Hash Aggregate: 6.2667 (HASH MATCH - INNER JOIN)
Best Estimated Subtree Cost for *Group-By Algorithm*: 2.8148 + 3.4185 = 6.2333 (HASH AGGREGATE WITH HASH MATCH - INNER JOIN)

Based on the complexity of the query and the nature of the data involved, the most efficient join algorithm appears to be the Hash Match - Inner Join. This algorithm provides a faster method for joining datasets.

For the GROUP BY operation, the most optimal algorithm seems to be the Hash Match (Aggregate). This algorithm efficiently aggregates our sorted data. By leveraging the Hash Match - Inner Join for the join operation and the Hash Aggregate for the GROUP BY operation, the query can achieve optimal performance in terms of execution time and resource utilization.

QUESTION 4

⇛ SQL Code:

```
USE ProjectDB;

SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District],
[Energy].[Municipality], [Energy].[Parish], [Energy].[ActiveEnergy],
[Contracts].[NumberContracts], [Energy].[ActiveEnergy] /
[Contracts].[NumberContracts] AS EnergyPerContract
FROM
      ( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS [ActiveEnergy]
      FROM [Energy].[MonthlyConsumption]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Energy],
      ( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([NumberContracts]) AS [NumberContracts]
      FROM [Energy].[ActiveContracts]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Contracts]
WHERE [Energy].[DistrictMunicipalityParishCode] =
 [Contracts].[DistrictMunicipalityParishCode]
ORDER BY  [Energy].[District], [Energy].[Municipality], [Energy].[Parish]
```

⇛ Procedure:

1. Identify each nested subquery. These are the **SELECT** statements within the parentheses.

   Nested Subquery 1:

```
FROM
      ( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS [ActiveEnergy]
      FROM [Energy].[MonthlyConsumption]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Energy],
```

Nested Subquery 2:

```sql
( SELECT [DistrictMunicipalityParishCode], [District],  [Municipality],
[Parish], SUM([NumberContracts]) AS [NumberContracts]
      FROM [Energy].[ActiveContracts]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Contracts]
```

2. A view is materialized when the results of that view are stored in the database, as opposed to storing only the view definition. In SQL Server, this is achieved by creating a unique clustered index on the view. For each nested subquery, a materialized view that stores the results of the query was created. The view should contain the same columns and data as the original subquery.

3. We have to add the COUNT_BIG since the query contains an aggregation (SUM) that is calculated from data in a table. To help keep the view updated more efficiently when there are aggregations, SQLserver uses a COUNT of rows to quickly understand if you need to update a view. The count works as a kind of flag to understand if an update is needed, detecting if there are new rows in the table or if something has been deleted.
   The COUNT actually has to be COUNT_BIG (bigint) in case it is necessary to count a number so large that it would overflow the 4 bytes of the COUNT.

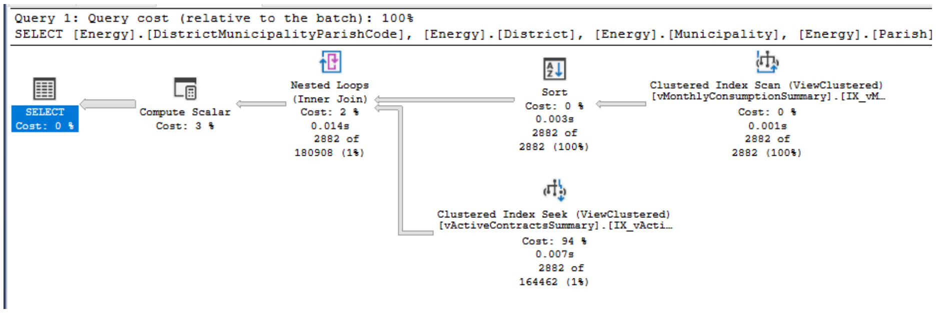
```sql
CREATE VIEW Energy.vMonthlyConsumptionSummary WITH SCHEMABINDING AS
SELECT [DistrictMunicipalityParishCode], [District],  [Municipality],
[Parish],  SUM([ActiveEnergy]) AS [ActiveEnergy],  COUNT_BIG(*) AS
[Counting]
FROM [Energy].[MonthlyConsumption]
GROUP BY [DistrictMunicipalityParishCode], [District],  [Municipality],
[Parish];

CREATE VIEW Energy.vActiveContractsSummary WITH SCHEMABINDING AS
```

```sql
SELECT [DistrictMunicipalityParishCode], [District],  [Municipality],
[Parish],  SUM([NumberContracts]) AS [NumberContracts],  COUNT_BIG(*) AS
[Counting]
FROM [Energy].[ActiveContracts]
GROUP BY [DistrictMunicipalityParishCode], [District],  [Municipality],
[Parish];
```

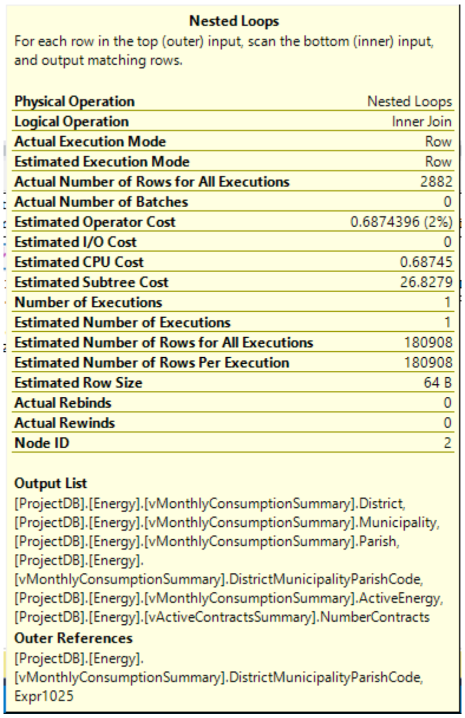
4. Then create a clustered index on them, **DistrictMunicipalityParishCode** column used as it is an unique identifier for both views:

```sql
CREATE UNIQUE CLUSTERED INDEX IX_vMonthlyConsumptionSummary
ON Energy.vMonthlyConsumptionSummary (DistrictMunicipalityParishCode);

CREATE UNIQUE CLUSTERED INDEX IX_vActiveContractsSummary
ON Energy.vActiveContractsSummary (DistrictMunicipalityParishCode);
```

5. Loop join:



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality], [Energy].[Parish]
```



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
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	0
Estimated Operator Cost	0.6874396 (2%)
Estimated I/O Cost	0
Estimated CPU Cost	0.68745
Estimated Subtree Cost	26.8279
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	180908
Estimated Number of Rows Per Execution	180908
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

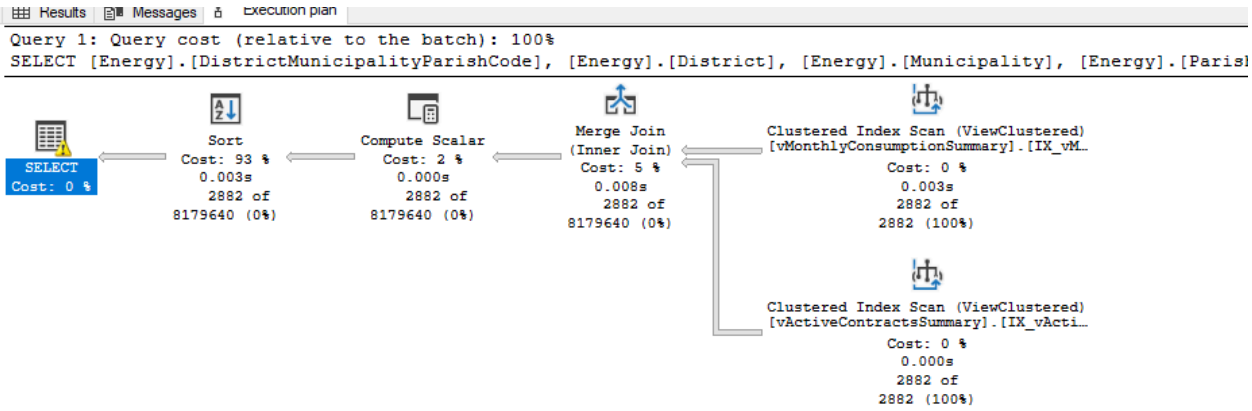
Output List

[ProjectDB].[Energy].[vMonthlyConsumptionSummary].District, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].Municipality, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].Parish, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].ActiveEnergy, [ProjectDB].[Energy].[vActiveContractsSummary].NumberContracts

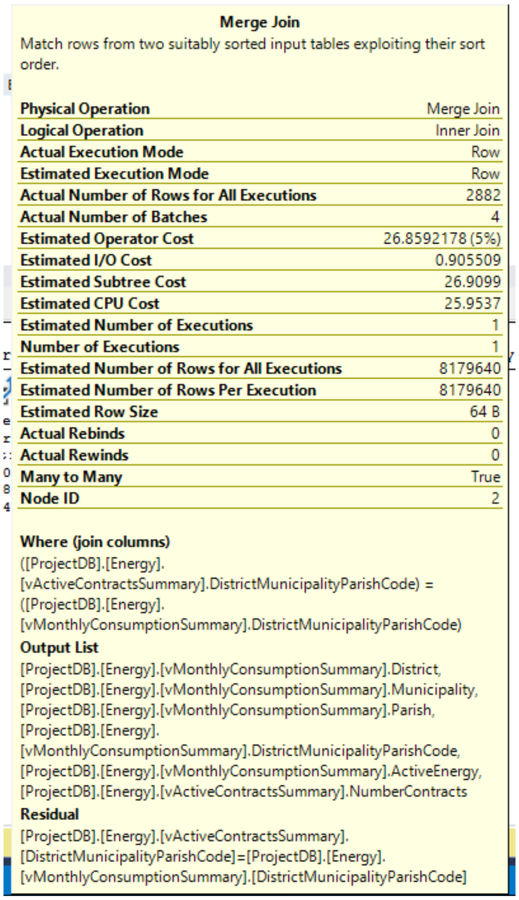
Outer References

[ProjectDB].[Energy].[vMonthlyConsumptionSummary].DistrictMunicipalityParishCode, Expr1025

6. Merge Join:



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality], [Energy].[Parisl
```



Merge Join

Match rows from two suitably sorted input tables exploiting their sort order.

Physical Operation	Merge Join
Logical Operation	Inner Join
Actual Execution Mode	Row
Estimated Execution Mode	Row
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	26.8592178 (5%)
Estimated I/O Cost	0.905509
Estimated Subtree Cost	26.9099
Estimated CPU Cost	25.9537
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	8179640
Estimated Number of Rows Per Execution	8179640
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Many to Many	True
Node ID	2

Where (join columns)

([ProjectDB].[Energy].[vActiveContractsSummary].DistrictMunicipalityParishCode) = ([ProjectDB].[Energy].[vMonthlyConsumptionSummary].DistrictMunicipalityParishCode)

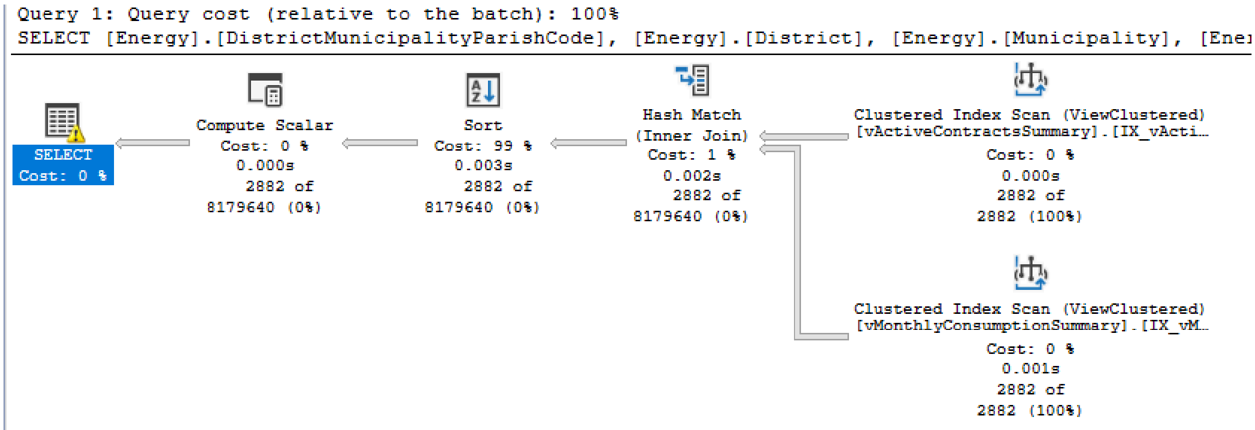
Output List

[ProjectDB].[Energy].[vMonthlyConsumptionSummary].District, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].Municipality, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].Parish, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].DistrictMunicipalityParishCode, [ProjectDB].[Energy].[vMonthlyConsumptionSummary].ActiveEnergy, [ProjectDB].[Energy].[vActiveContractsSummary].NumberContracts

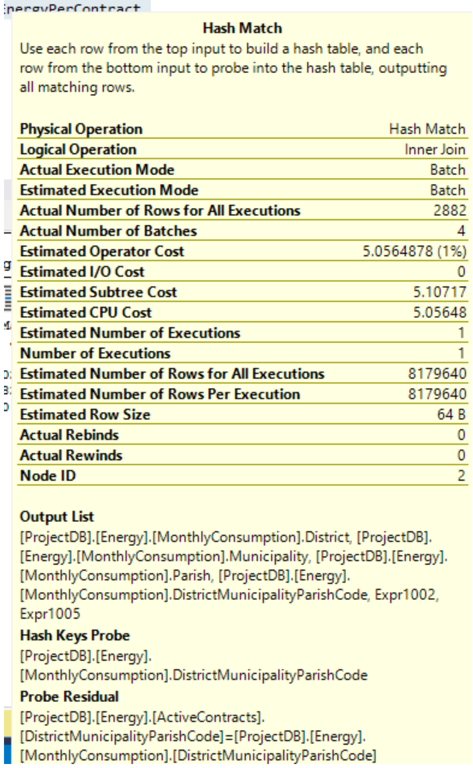
Residual

[ProjectDB].[Energy].[vActiveContractsSummary].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[vMonthlyConsumptionSummary].[DistrictMunicipalityParishCode]

7. Hash Match(Inner Join):



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality], [Ene
```



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	Batch
Estimated Execution Mode	Batch
Actual Number of Rows for All Executions	2882
Actual Number of Batches	4
Estimated Operator Cost	5.0564878 (1%)
Estimated I/O Cost	0
Estimated Subtree Cost	5.10717
Estimated CPU Cost	5.05648
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows for All Executions	8179640
Estimated Number of Rows Per Execution	8179640
Estimated Row Size	64 B
Actual Rebinds	0
Actual Rewinds	0
Node ID	2

Output List
[ProjectDB].[Energy].[MonthlyConsumption].District, [ProjectDB].[Energy].[MonthlyConsumption].Municipality, [ProjectDB].[Energy].[MonthlyConsumption].Parish, [ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode, Expr1002, Expr1005

Hash Keys Probe
[ProjectDB].[Energy].[MonthlyConsumption].DistrictMunicipalityParishCode

Probe Residual
[ProjectDB].[Energy].[ActiveContracts].[DistrictMunicipalityParishCode]=[ProjectDB].[Energy].[MonthlyConsumption].[DistrictMunicipalityParishCode]

8. Materialized views provide precomputed summaries of data, which can simplify the query optimization process for the database optimizer.
Instead of dealing with complex subqueries involving aggregation functions like **SUM** and **GROUP BY**, the optimizer can work with the already aggregated data in the materialized views.
This allows the optimizer to generate more efficient query plans resulting in lower estimated subtree costs.

QUESTION 5

Task 2 as Workload:

⇛ SQL Code:

```
USE ProjectDB;

SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy]
FROM [Energy].MonthlyConsumption]
WHERE [Municipality] = ‘Lisboa’ AND [Month] = ‘06’
GROUP BY [Parish], [Year]
ORDER BY [Parish], [Year]
```

⇛ Procedure:

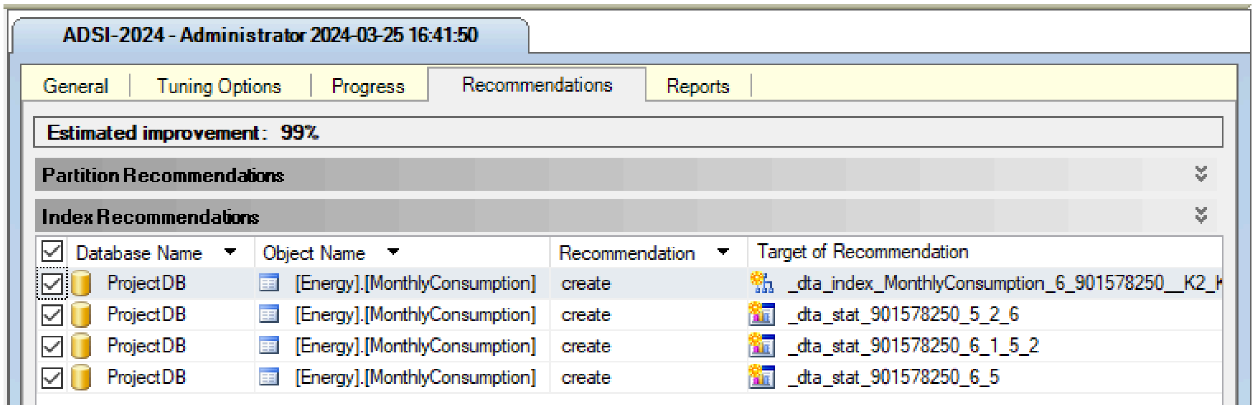
1 - SQL Server Management Studio

- Copy SQL Code and save as sql_2.

2 - Database Engine Tuning Advisor

- From the Start menu, launch Database Engine Tuning Advisor, and connect to SQL Server.
- In the General tab, under Workload, select File and browse to the location of sql_2.sql. This will be the input workload for this session.
- In the ***Database for workload analysis***, select ProjectDB.
- Under ***Select databases and tables to tune***, check ProjectDB in order to select all tables from this database.
- Change to the ***Tuning Options*** tab.
- Uncheck the option Limit tuning time.
- In ***Physical Design Structures (PDS) to use in the database***, select Indexes and indexed views.
- In ***Physical Design Structures (PDS) to keep in the database***, select Keep all existing PDS.
- In the ***Partitioning strategy to employ***, select No partitioning.
- In the toolbar, hit the ***Start Analysis*** button.

- A new Progress tab will open while the analysis is in progress. Wait for the analysis to conclude.
- Once the analysis finishes, two additional tabs will open: Recommendations, and Reports.
- In the ***Recommendations*** tab, have a look at the list of recommendations.
- In particular, hover the mouse over the Target of Recommendation column, until a tooltip appears with the details of each recommendation.

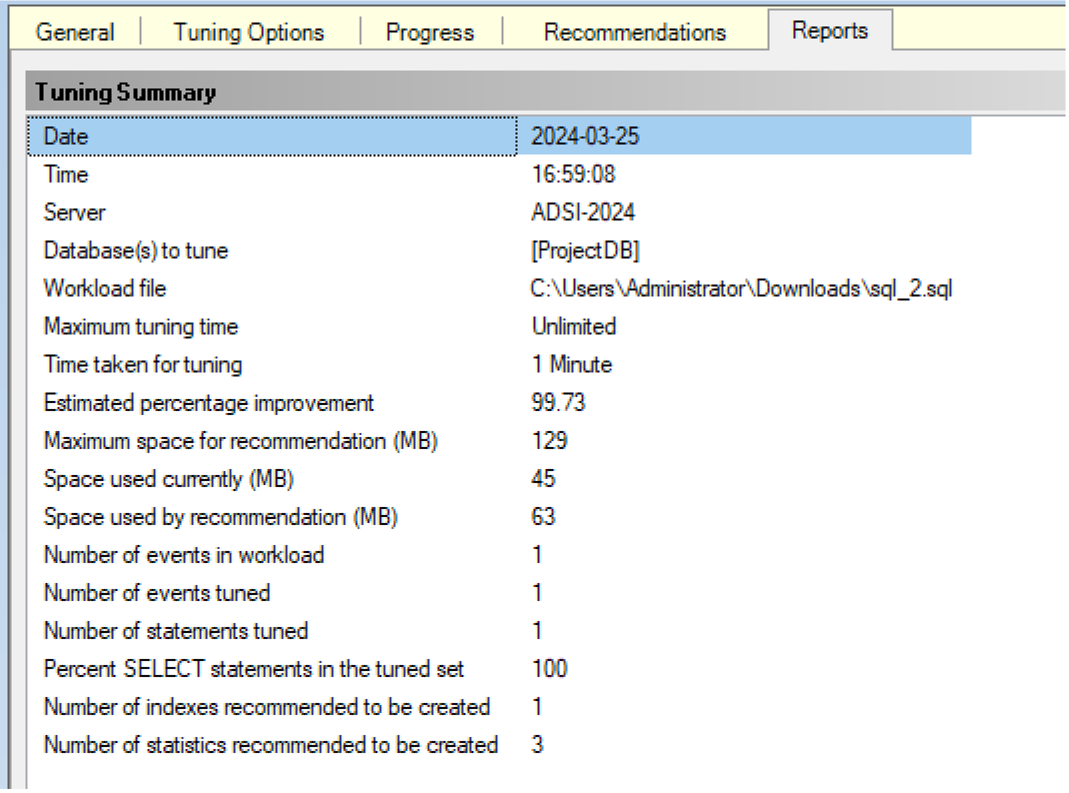


Estimated improvement: 99%

Partition Recommendations

Index Recommendations

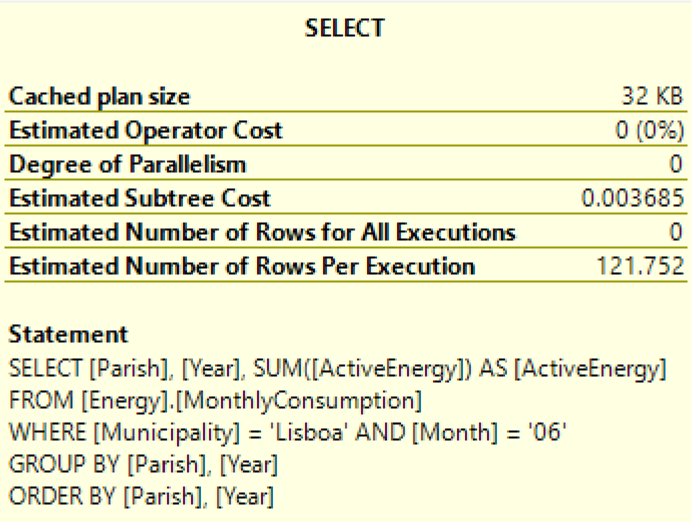
☑ Database Name	Object Name	Recommendation	Target of Recommendation
☑ ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_index_MonthlyConsumption_6_901578250__K2_K
☑ ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_stat_901578250_5_2_6
☑ ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_stat_901578250_6_1_5_2
☑ ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_stat_901578250_6_5



Tuning Summary

Date	2024-03-25
Time	16:59:08
Server	ADSI-2024
Database(s) to tune	[ProjectDB]
Workload file	C:\Users\Administrator\Downloads\sql_2.sql
Maximum tuning time	Unlimited
Time taken for tuning	1 Minute
Estimated percentage improvement	99.73
Maximum space for recommendation (MB)	129
Space used currently (MB)	45
Space used by recommendation (MB)	63
Number of events in workload	1
Number of events tuned	1
Number of statements tuned	1
Percent SELECT statements in the tuned set	100
Number of indexes recommended to be created	1
Number of statistics recommended to be created	3

- After applying recommendations, it is possible to verify the execution plan and estimated subtree cost.

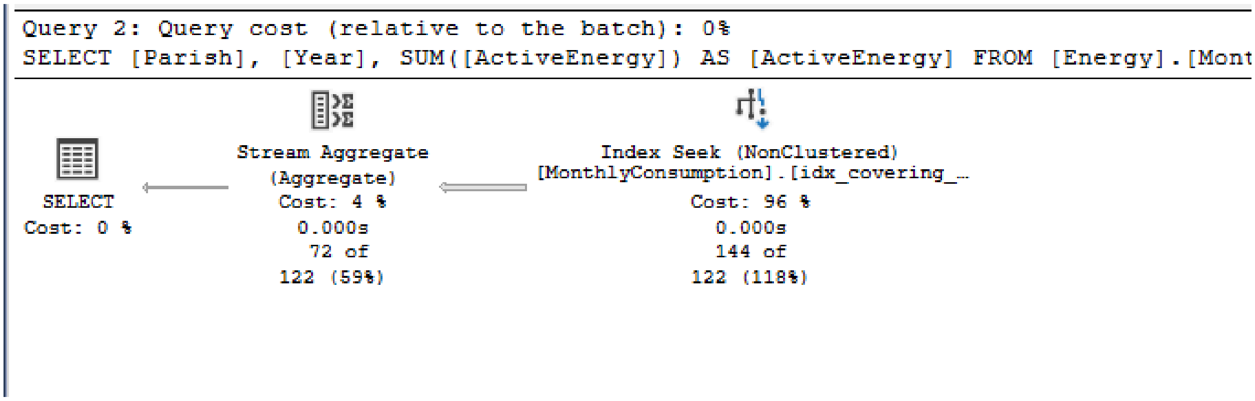


SELECT

Cached plan size	32 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	0
Estimated Subtree Cost	0.003685
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	121.752

Statement

```
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy]
FROM [Energy].[MonthlyConsumption]
WHERE [Municipality] = 'Lisboa' AND [Month] = '06'
GROUP BY [Parish], [Year]
ORDER BY [Parish], [Year]
```



```
Query 2: Query cost (relative to the batch): 0%
SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy] FROM [Energy].[Mont
```

- So, we can conclude that estimated subtree cost improvement is from **2.6069** to **0.0037**.

Task 3 as Workload:

⇛ SQL Code:

```
USE ProjectDB;

SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District],
[Energy].[Municipality], [Energy].[Parish], [Energy].[ActiveEnergy],
[Contracts].[NumberContracts], [Energy].[ActiveEnergy] /
[Contracts].[NumberContracts] AS EnergyPerContract
FROM
```

```
      ( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS [ActiveEnergy]
      FROM [Energy].[MonthlyConsumption]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Energy],
      ( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([NumberContracts]) AS [NumberContracts]
      FROM [Energy].[ActiveContracts]
      GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish] ) AS [Contracts]
WHERE [Energy].[DistrictMunicipalityParishCode] =
 [Contracts].[DistrictMunicipalityParishCode]
ORDER BY  [Energy].[District], [Energy].[Municipality], [Energy].[Parish]
```

⇛ **Procedure:**

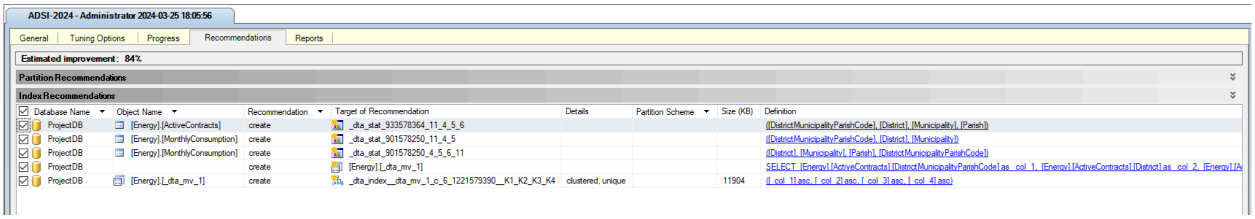
1 - SQL Server Management Studio

- Copy SQL Code and save as sql_3.

2 - Database Engine Tuning Advisor

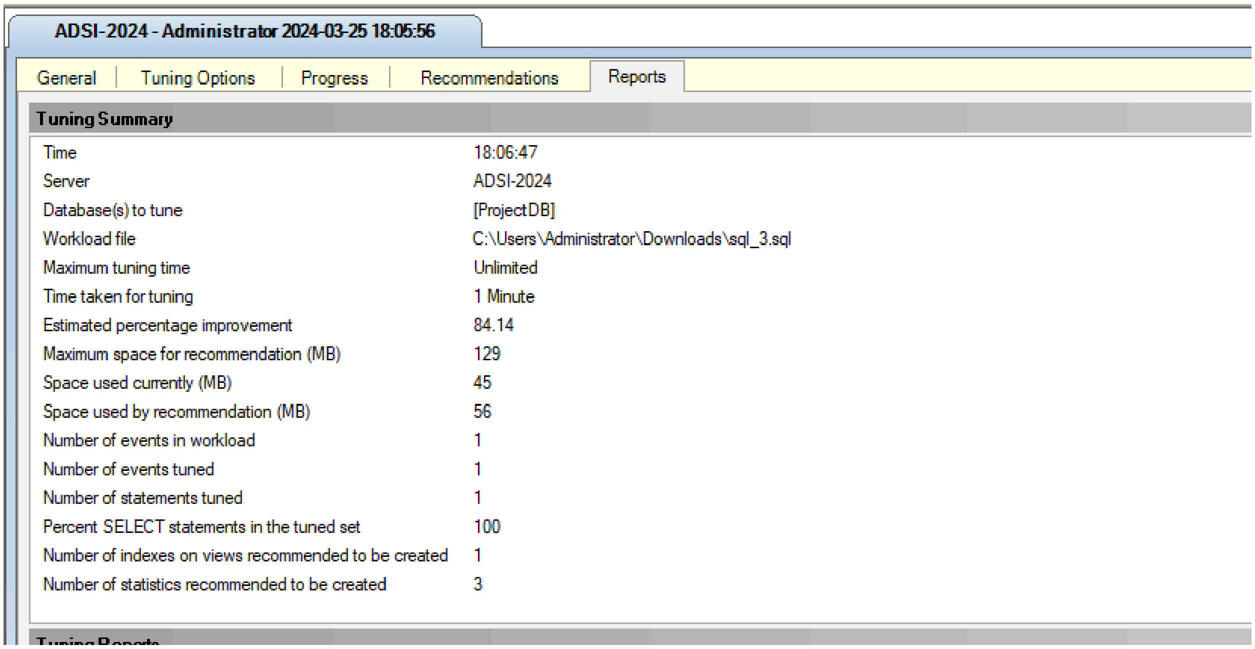
- From the Start menu, launch Database Engine Tuning Advisor, and connect to SQL Server.
- In the General tab, under Workload, select File and browse to the location of sql_3.sql. This will be the input workload for this session.
- In the ***Database for workload analysis***, select ProjectDB.
- Under ***Select databases and tables to tune***, check ProjectDB in order to select all tables from this database.
- Change to the ***Tuning Options*** tab.
- Uncheck the option Limit tuning time.
- In ***Physical Design Structures (PDS) to use in the database***, select Indexes and indexed views.
- In ***Physical Design Structures (PDS) to keep in the database***, select Keep all existing PDS.
- In the ***Partitioning strategy to employ***, select No partitioning.

- In the toolbar, hit the ***Start Analysis*** button.
- A new Progress tab will open while the analysis is in progress. Wait for the analysis to conclude.
- Once the analysis finishes, two additional tabs will open: Recommendations, and Reports.
- In the ***Recommendations*** tab, have a look at the list of recommendations.
- In particular, hover the mouse over the Target of Recommendation column, until a tooltip appears with the details of each recommendation.



Estimated improvement: 84%

☑	Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme	Size (KB)	Definition
☑	ProjectDB	[Energy].[ActiveContracts]	create	_dta_stat_933578364_11_4_5_6				([DistrictMunicipalityParishCode], [District], [Municipality], [Parish])
☑	ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_stat_901578250_11_4_5				([DistrictMunicipalityParishCode], [District], [Municipality])
☑	ProjectDB	[Energy].[MonthlyConsumption]	create	_dta_stat_901578250_4_5_6_11				([District], [Municipality], [Parish], [DistrictMunicipalityParishCode])
☑	ProjectDB		create	[Energy].[_dta_mv_1]				SELECT [Energy].[ActiveContracts].[DistrictMunicipalityParishCode] as _col_1, [Energy].[ActiveContracts].[District] as _col_2, [Energy].[A
☑	ProjectDB	[Energy].[_dta_mv_1]	create	_dta_index__dta_mv_1_c_6_1221579390__K1_K2_K3_K4	clustered, unique		11904	([_col_1] asc, [_col_2] asc, [_col_3] asc, [_col_4] asc)

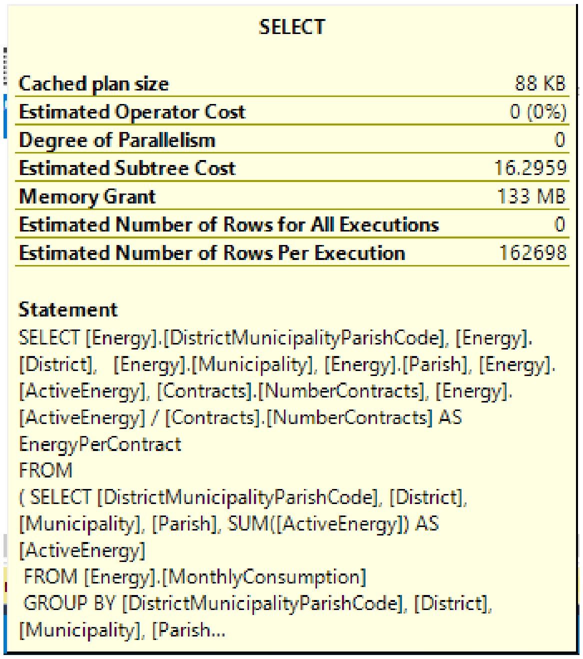


Tuning Summary

Time	18:06:47
Server	ADSI-2024
Database(s) to tune	[ProjectDB]
Workload file	C:\Users\Administrator\Downloads\sql_3.sql
Maximum tuning time	Unlimited
Time taken for tuning	1 Minute
Estimated percentage improvement	84.14
Maximum space for recommendation (MB)	129
Space used currently (MB)	45
Space used by recommendation (MB)	56
Number of events in workload	1
Number of events tuned	1
Number of statements tuned	1
Percent SELECT statements in the tuned set	100
Number of indexes on views recommended to be created	1
Number of statistics recommended to be created	3

Tuning Reports

- After applying recommendations, it is possible to verify the execution plan and estimated subtree cost.

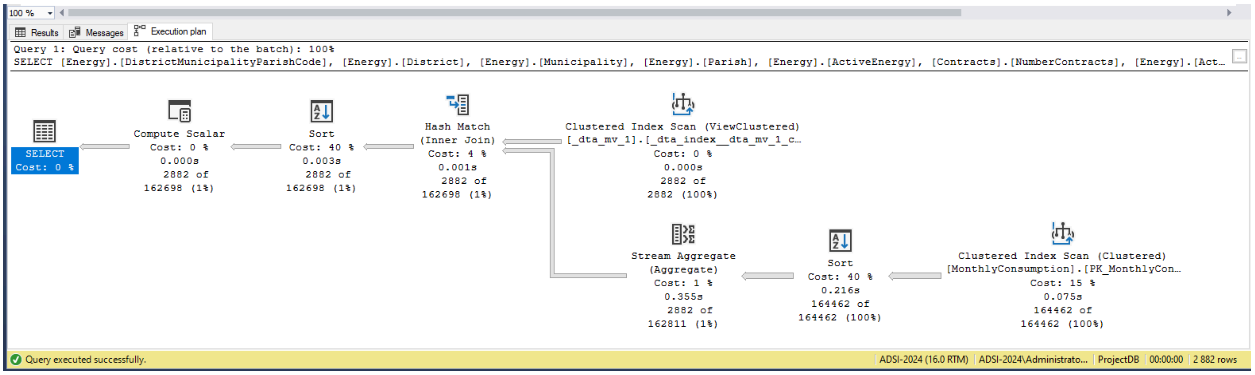


SELECT

Cached plan size	88 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	0
Estimated Subtree Cost	16.2959
Memory Grant	133 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	162698

Statement

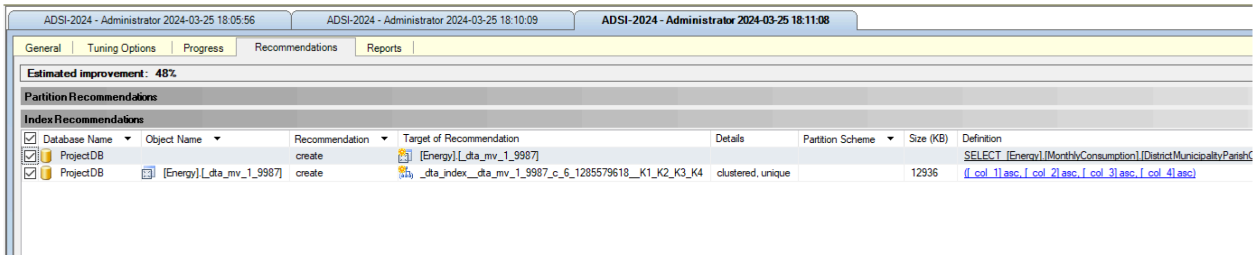
```
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].
[District],   [Energy].[Municipality], [Energy].[Parish], [Energy].
[ActiveEnergy], [Contracts].[NumberContracts], [Energy].
[ActiveEnergy] / [Contracts].[NumberContracts] AS
EnergyPerContract
FROM
( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS
[ActiveEnergy]
 FROM [Energy].[MonthlyConsumption]
 GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish...
```



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality], [Energy].[Parish], [Energy].[ActiveEnergy], [Contracts].[NumberContracts], [Energy].[Act…
```

Query executed successfully. | ADSI-2024 (16.0 RTM) | ADSI-2024\Administrato... | ProjectDB | 00:00:00 | 2 882 rows

- *Now, **Start Analysis** again and verify if estimated improvements are 0 or possible.*

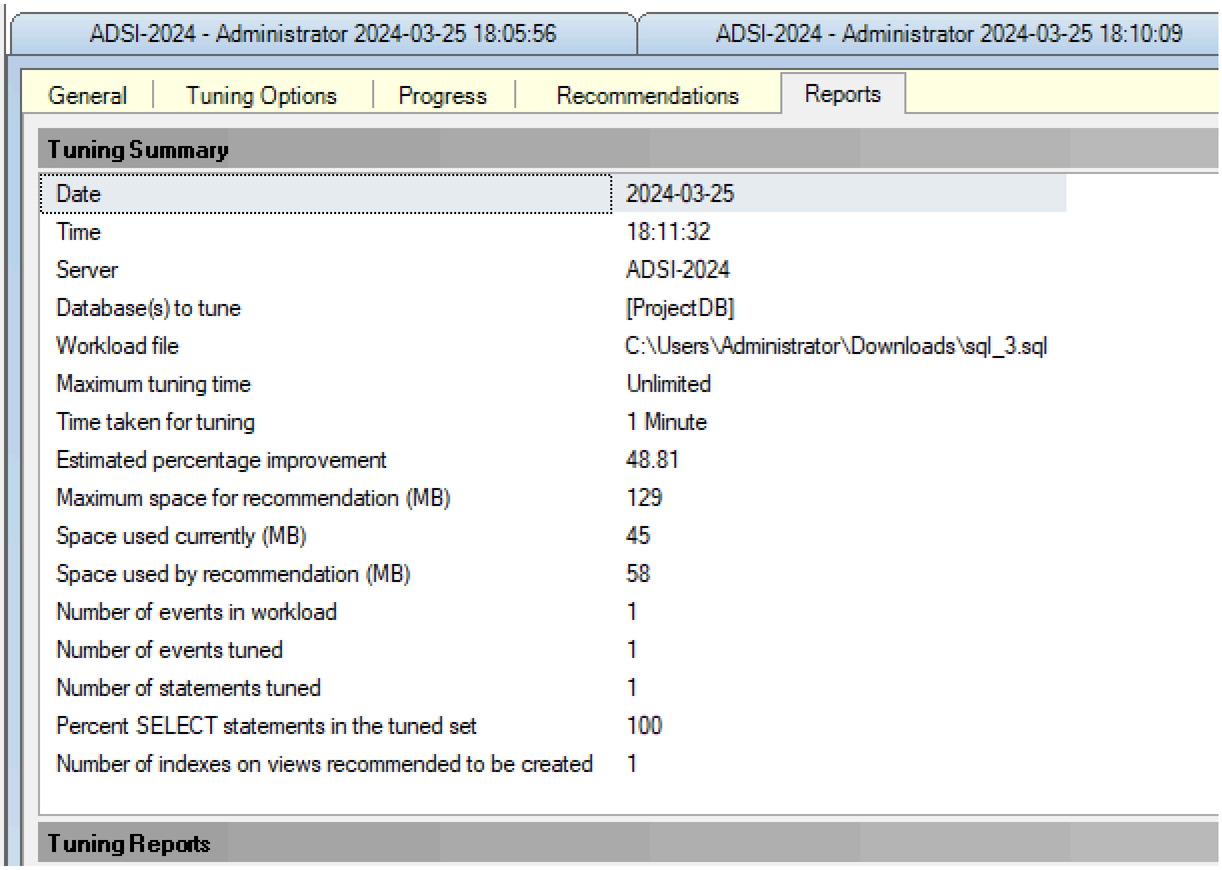

ADSI-2024 - Administrator 2024-03-25 18:05:56
ADSI-2024 - Administrator 2024-03-25 18:10:09
ADSI-2024 - Administrator 2024-03-25 18:11:08
General
Tuning Options
Progress
Recommendations
Reports

Estimated improvement: 48%

Partition Recommendations

Index Recommendations

☑	Database Name	Object Name	Recommendation	Target of Recommendation	Details	Partition Scheme	Size (KB)	Definition
☑	ProjectDB		create	[Energy].[_dta_mv_1_9987]				SELECT [Energy].[MonthlyConsumption].[DistrictMunicipalityParishC
☑	ProjectDB	[Energy].[_dta_mv_1_9987]	create	_dta_index__dta_mv_1_9987_c_6_1285579618__K1_K2_K3_K4	clustered, unique		12936	([_col_1] asc, [_col_2] asc, [_col_3] asc, [_col_4] asc)

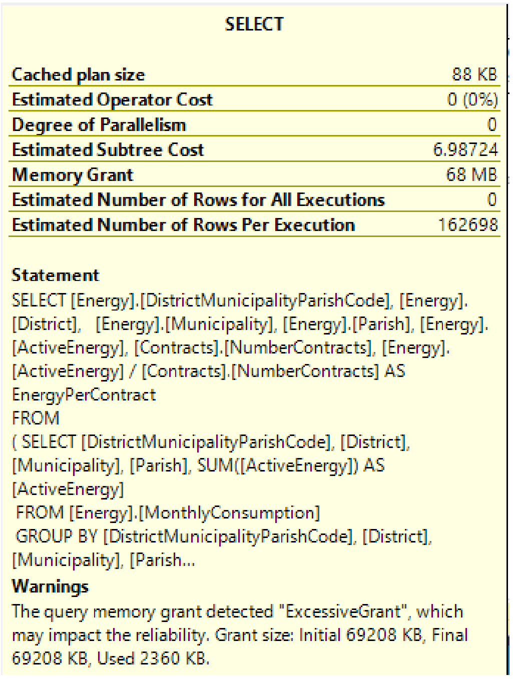

ADSI-2024 - Administrator 2024-03-25 18:05:56
ADSI-2024 - Administrator 2024-03-25 18:10:09
General
Tuning Options
Progress
Recommendations
Reports

Tuning Summary

Date	2024-03-25
Time	18:11:32
Server	ADSI-2024
Database(s) to tune	[ProjectDB]
Workload file	C:\Users\Administrator\Downloads\sql_3.sql
Maximum tuning time	Unlimited
Time taken for tuning	1 Minute
Estimated percentage improvement	48.81
Maximum space for recommendation (MB)	129
Space used currently (MB)	45
Space used by recommendation (MB)	58
Number of events in workload	1
Number of events tuned	1
Number of statements tuned	1
Percent SELECT statements in the tuned set	100
Number of indexes on views recommended to be created	1

Tuning Reports

- After applying recommendations, it is possible to verify the execution plan and estimated subtree cost.



SELECT

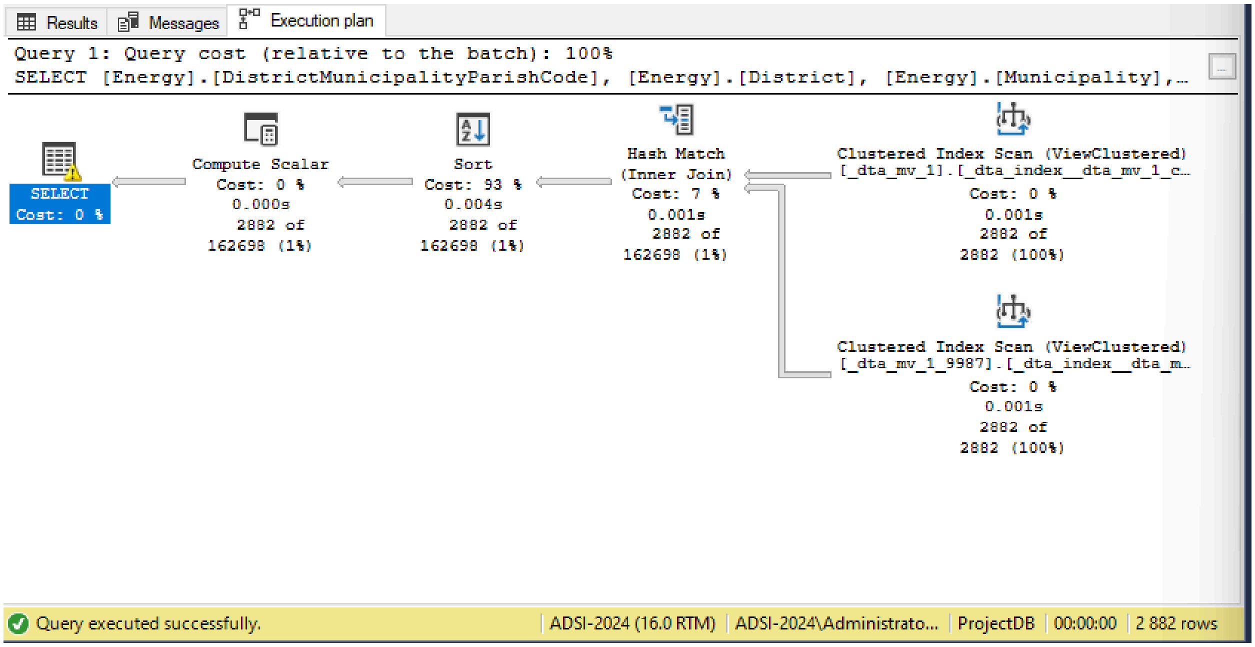
Cached plan size	88 KB
Estimated Operator Cost	0 (0%)
Degree of Parallelism	0
Estimated Subtree Cost	6.98724
Memory Grant	68 MB
Estimated Number of Rows for All Executions	0
Estimated Number of Rows Per Execution	162698

Statement

```
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].
[District],   [Energy].[Municipality], [Energy].[Parish], [Energy].
[ActiveEnergy], [Contracts].[NumberContracts], [Energy].
[ActiveEnergy] / [Contracts].[NumberContracts] AS
EnergyPerContract
FROM
( SELECT [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish], SUM([ActiveEnergy]) AS
[ActiveEnergy]
 FROM [Energy].[MonthlyConsumption]
 GROUP BY [DistrictMunicipalityParishCode], [District],
[Municipality], [Parish...
```

Warnings

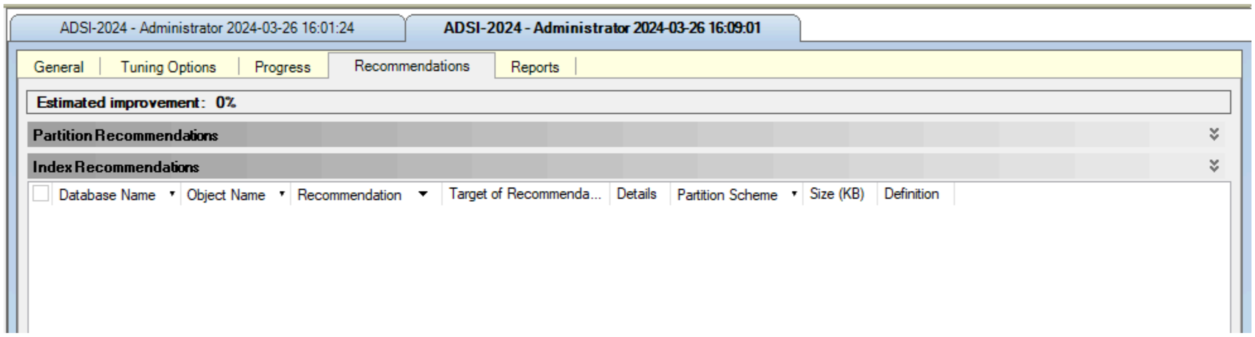
The query memory grant detected "ExcessiveGrant", which may impact the reliability. Grant size: Initial 69208 KB, Final 69208 KB, Used 2360 KB.



```
Query 1: Query cost (relative to the batch): 100%
SELECT [Energy].[DistrictMunicipalityParishCode], [Energy].[District], [Energy].[Municipality],…
```

Query executed successfully. | ADSI-2024 (16.0 RTM) | ADSI-2024\Administrato... | ProjectDB | 00:00:00 | 2 882 rows

- *Now, Start Analysis again and verify if estimated improvements are 0 or possible.*
- *Now, no recommendations are suggested. So estimated improvements are 0% as you can observe.*


ADSI-2024 - Administrator 2024-03-26 16:01:24
ADSI-2024 - Administrator 2024-03-26 16:09:01
General
Tuning Options
Progress
Recommendations
Reports
Partition Recommendations
Index Recommendations

Estimated improvement: 0%

□	Database Name	Object Name	Recommendation	Target of Recommenda...	Details	Partition Scheme	Size (KB)	Definition

- *After applying recommendations, we can conclude that estimated subtree cost improvement is from **523.135** to **6.98724**.*