# Task 1

|  |
| --- |
| CREATE PARTITION FUNCTION monthlyConsumptionYears(CHAR(4))  AS RANGE LEFT FOR VALUES (2020, 2021, 2022, 2023);  CREATE PARTITION SCHEME monthlyConsumptionYearsPartition  AS PARTITION monthlyConsumptionYears  TO ([PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY]);  ALTER TABLE Energy.MonthlyConsumption DROP CONSTRAINT PK\_MonthlyConsumption;  CREATE CLUSTERED INDEX PK\_MonthlyConsumption  ON Energy.MonthlyConsumption (Year, Month, DistrictMunicipalityParishCode, VoltageLevel)  ON monthlyConsumptionYearsPartition(Year); |
| SELECT \* FROM sys.partitions as p  WHERE      p.rows IN (          SELECT COUNT(\*) FROM Energy.MonthlyConsumption      ) OR p.rows IN (          SELECT COUNT(\*) FROM Energy.MonthlyConsumption          GROUP BY Year      )  ORDER BY p.rows DESC |

To separate a table into multiple partitions it is necessary to create N partitions and to distinguish from them, a partition function, so that we can know in which partition to look for, given a certain record, in this case, we separate them by Year.

Before splitting the table into partitions, it is mandatory to remove the primary key’s clustered index, since it organises the records by their primary key inside a single partition. Only after, a new clustered index can be created following the newly created partition function, separating each record into its respective partition.

# Task 2

Since the column Year in used in the primary key clustered index, its information is already present in the pointer to the data table. This means that its presence in not necessary in a new index.

It is also important to note that the table is already ordered by year due to the clustered index.

Because the column ActiveEnergy is only used inside a sum statement, it is only important to include their values into the end of the tree instead of sorting the index tree by them.

**Initial state**

**A screenshot of a computer

Description automatically generated**

Since the main purpose of an index is to speed up the search of some given records, there is no doubt that an index with Municipality and Month would help greatly, speeding up the look up for the records with the given values, so the first index to try would be:

DROP INDEX IF EXISTS IX\_Month\_Municipality ON Energy.MonthlyConsumption

CREATE NONCLUSTERED INDEX IX\_Month\_Municipality ON Energy.MonthlyConsumption (Month, Municipality) INCLUDE (ActiveEnergy)

A screenshot of a computer

Description automatically generated

There is still a key lookup being executed using 96% of the total cost. It’s only used to get the data for the Parish column. Since it is on a group by and a order by statements, it should go inside the main index clause, but let’s include it first, appending its values to the end of the tree.

DROP INDEX IF EXISTS IX\_Month\_Municipality\_Recomended ON Energy.MonthlyConsumption

CREATE NONCLUSTERED INDEX IX\_Month\_Municipality\_Recomended ON Energy.MonthlyConsumption (Month, Municipality) INCLUDE (Parish, ActiveEnergy)

A screenshot of a computer

Description automatically generated

There was a clear benefit on removing that Key Lookup, turning its 0.354 estimated cost into the 0.000345 estimated cost of the hole Index Seek. But now the Sort is the step that is slowing down the most. Moving the Parish column to be used in the tree will automatically sort its values, removing completely the need for a Sort block.

DROP INDEX IF EXISTS IX\_Month\_Municipality\_Parish ON Energy.MonthlyConsumption

CREATE NONCLUSTERED INDEX IX\_Month\_Municipality\_Parish ON Energy.MonthlyConsumption (Month, Municipality, Parish) INCLUDE (ActiveEnergy)

A screenshot of a computer

Description automatically generated

Concluding, the best Index found uses the columns Month, Municipality and Parish, in this order, so after filtering the given Month and Municipality, the Parish is already sorted in the tree, and accesses the Year and ActiveEnergy data from the primary key pointer and the include clause, respectively.

# Task 3

The following SQL instructions were added at the end of the given query.

|  |
| --- |
| OPTION (LOOP JOIN) |
| 110389 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (MERGE JOIN) |
| 596.062 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (HASH JOIN) |
| 535.201 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (STREAM AGGREGATE) |
| 535.201 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (HASH GROUP) |
| 567.669 |

A screenshot of a computer

Description automatically generated

# Task 4

IF OBJECT\_ID ('Energy.energy', 'view') IS NOT NULL

   DROP VIEW Energy.energy;

GO

CREATE VIEW Energy.energy WITH SCHEMABINDING AS

    SELECT [DistrictMunicipalityParishCode],

        [District],

        [Municipality],

        [Parish],

        SUM([ActiveEnergy]) AS [ActiveEnergy],

        COUNT\_BIG(\*) AS COUNT

    FROM [Energy].[MonthlyConsumption]

    GROUP BY [DistrictMunicipalityParishCode],

        [District],

        [Municipality],

        [Parish];

GO

CREATE UNIQUE CLUSTERED INDEX IX\_DistrictMunicipalityParishCode

   ON Energy.energy (DistrictMunicipalityParishCode);

GO

IF OBJECT\_ID ('Energy.contracts', 'view') IS NOT NULL

   DROP VIEW Energy.contracts;

GO

CREATE VIEW Energy.contracts WITH SCHEMABINDING AS

    SELECT [DistrictMunicipalityParishCode],

        [District],

        [Municipality],

        [Parish],

        SUM([NumberContracts]) AS [NumberContracts],

        COUNT\_BIG(\*) AS COUNT

    FROM [Energy].[ActiveContracts]

    GROUP BY [DistrictMunicipalityParishCode],

        [District],

        [Municipality],

        [Parish];

GO

CREATE UNIQUE CLUSTERED INDEX IX\_DistrictMunicipalityParishCode

   ON Energy.contracts (DistrictMunicipalityParishCode);

GO

SELECT [Energy].[DistrictMunicipalityParishCode],

    [Energy].[District],

    [Energy].[Municipality],

    [Energy].[Parish],

    [Energy].[ActiveEnergy],

    [Contracts].[NumberContracts],

    [Energy].[ActiveEnergy] / [Contracts].[NumberContracts] AS EnergyPerContract

FROM Energy.energy as [Energy], Energy.contracts as [Contracts]

WHERE [Energy].[DistrictMunicipalityParishCode] =

    [Contracts].[DistrictMunicipalityParishCode]

ORDER BY [Energy].[District],

    [Energy].[Municipality],

    [Energy].[Parish]

|  |
| --- |
| OPTION (LOOP JOIN) |
| 27.6459 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (MERGE JOIN) |
| 579.201 |

A screenshot of a computer

Description automatically generated

|  |
| --- |
| OPTION (HASH JOIN) |
| 518.119 |

A screenshot of a computer

Description automatically generated

# Task 5

In this task in order to get the best results possible run both queries from task 2 and 3 in the same workload in order to get the best results possible that apply to both.

This is our workload:

SELECT [Parish], [Year], SUM([ActiveEnergy]) AS [ActiveEnergy]

FROM [Energy].[MonthlyConsumption]

WHERE [Municipality] = 'Lisboa'

AND [Month] = '06'

GROUP BY [Parish], [Year]

ORDER BY [Parish], [Year]

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],

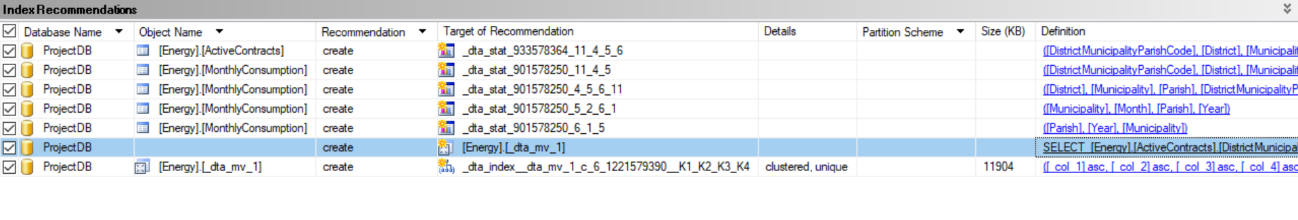
    [Energy].[Parish]

The workload will be run on Database Engine Tuning Advisor with the following settings: Uma imagem com texto, captura de ecrã, software

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

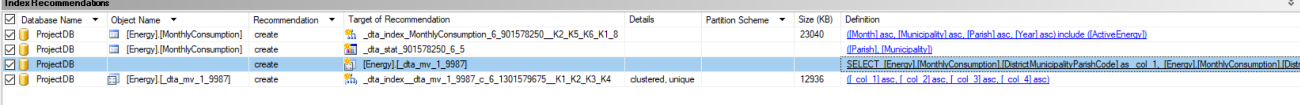
After running the analysis with these settings we get the following recommendations:



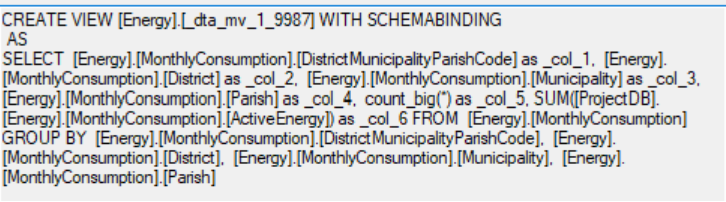
As we can notice the second to last recommendation looks familiar and that is because that is the second materialized view we created in task 4

Uma imagem com texto, Tipo de letra, file, captura de ecrã

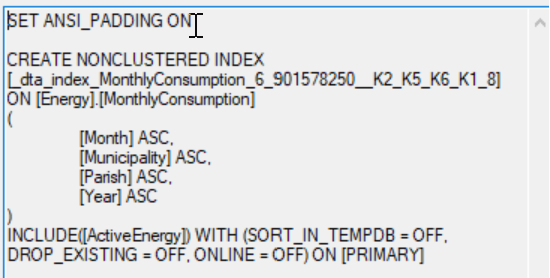
Descrição gerada automaticamente

By applying these recommendations and running a new analysis (with the same settings) we will get new recommendations

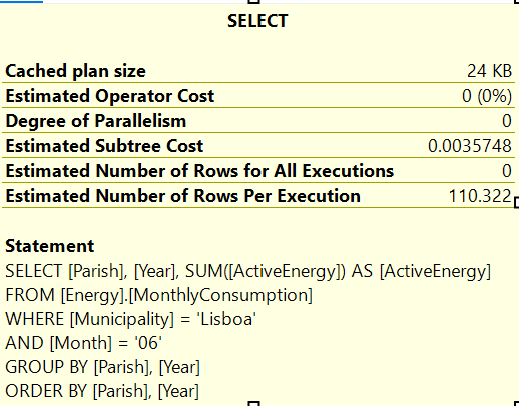
Again, as we can notice the third recommendation is the other materialized view we created in task 4



Also, the first recommendation is a new index that we can apply to the query on task 2 to get improved performance.



After creating this index we can compare the estimated subtree cost before and after

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente