**NEURON Database Table Partition**

Table of Contents

[Objective: 3](#_Toc14440583)

[When to use Partition: 3](#_Toc14440584)

[Things to keep in mind before partitioning 3](#_Toc14440585)

[Advantages of partitioning 3](#_Toc14440586)

[Disadvantages 3](#_Toc14440587)

[Left Based Partition vs Right Based Partition 4](#_Toc14440588)

[Pros and Cons: 5](#_Toc14440589)

[Steps on creating a Partitioned Table: 5](#_Toc14440590)

[1. Partition Function: 5](#_Toc14440591)

[2. Filegroup: 5](#_Toc14440592)

[3. Files: 6](#_Toc14440593)

[4. Partition Scheme: 6](#_Toc14440594)

[5. Table Creation: 6](#_Toc14440595)

[6. Querying in Partitioned Table: 7](#_Toc14440596)

[Performance Analysis: 7](#_Toc14440597)

[1. Simple SELECT Query 7](#_Toc14440598)

[2. Aggregation SELECT Query 7](#_Toc14440599)

[3. Parallel Performance Check: 8](#_Toc14440600)

# **Objective:**

The main purpose of this document is to gather some key information that would work as a guideline while deciding on whether and how to perform table partition on SQL Server. Several links are added for further inquiry.

# **When to use Partition**: [[Link](https://www.sqlshack.com/sql-server-2016-enhancements-truncate-table-table-partitioning/)]

* To Divide a very large table into smaller chunks
* To Easily maintain a window of historical data
* To Divide the data into equal parts on a certain criteria
  + To allow the SQL Query Optimizer decide and select the best plan for query execution
* To Improve data loading and reporting
* To Solve concurrency issue (getting locked or blocked)
* To Improve performance on maintenance operations
* To Improve user experience by metadata-only operation on latest/oldest data
* To Keep the size of the indexes to small

# **Things to keep in mind before partitioning** [[Link](https://stackoverflow.com/questions/49013911/disadvantages-of-table-partitioning-e-g-sql-server)]

* Check the explain plan of the query (if there is a particular access pattern) and tune it to make sure right indexes are used and disk operations are limited.
* Check the size of the table and index in question and how it relates to the configured database memory (buffer pool).
* Check if the table can be defragmented. All databases offer commands to do this (MySQL calls it OPTIMIZE, PostgreSQL calls it VACUUM)
* Check if you can add memory to server, RAM is cheap

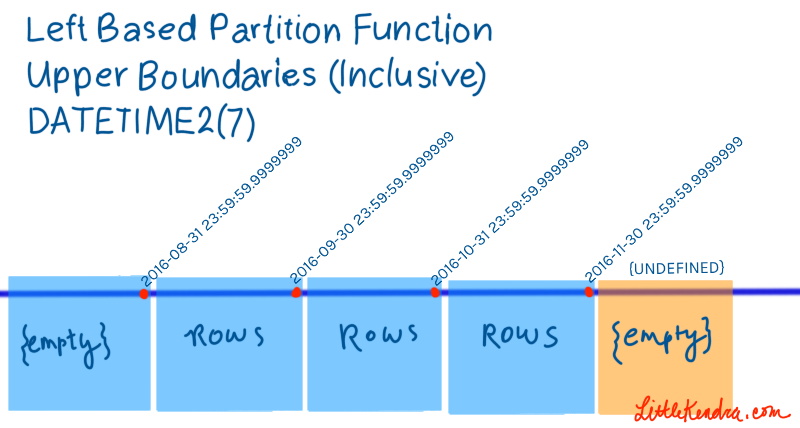
# **Advantages of partitioning**

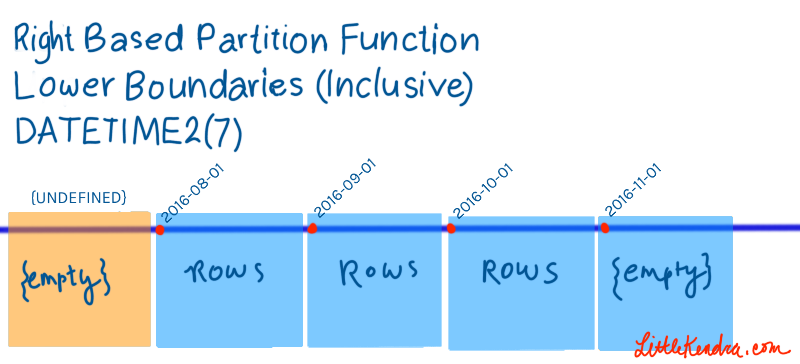
* Smaller tables
* smaller indexes
* lower query latency
* Metadata-only operations

# **Disadvantages**

* Intrusive as application needs changes to write to the correct partition. If the number of partitions is out-grown, you have the same issue with the partitions. AFAIK full re-partitioning would be needed to increase the partition count – which is a non-trivial activity.
* Query performance can be negatively affected
* Maintenance

# **Left Based Partition vs Right Based Partition** [[Link](https://littlekendra.com/2017/02/07/understanding-left-vs-right-partition-functions-with-diagrams/)]





Partitioning can NEGATIVELY affect Query Performance. [[Link](https://littlekendra.com/2016/05/03/why-table-partitioning-doesnt-speed-up-query-performance-video/)]

Query Performance can be improved using Clustered or Non-Clustered Indexing.

Historic data can be archived/removed instantaneously using the Sliding Window Partition technique. [[Link](https://www.mssqltips.com/sqlservertip/5296/implementation-of-sliding-window-partitioning-in-sql-server-to-purge-data/)]

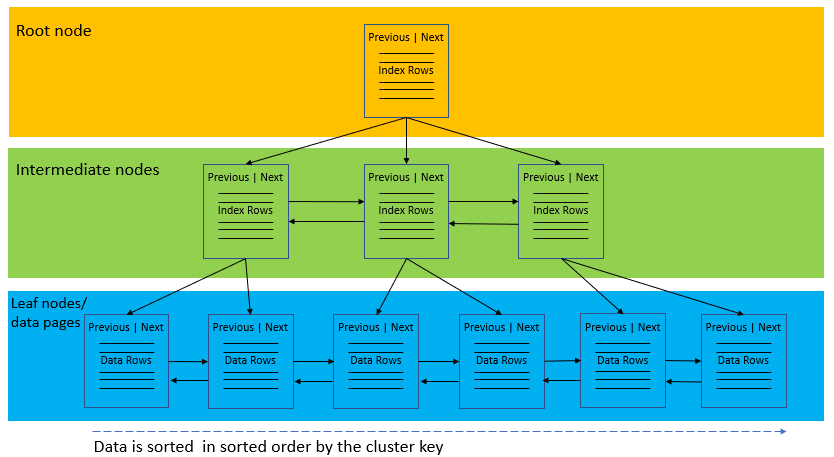
# **Pros and Cons:**

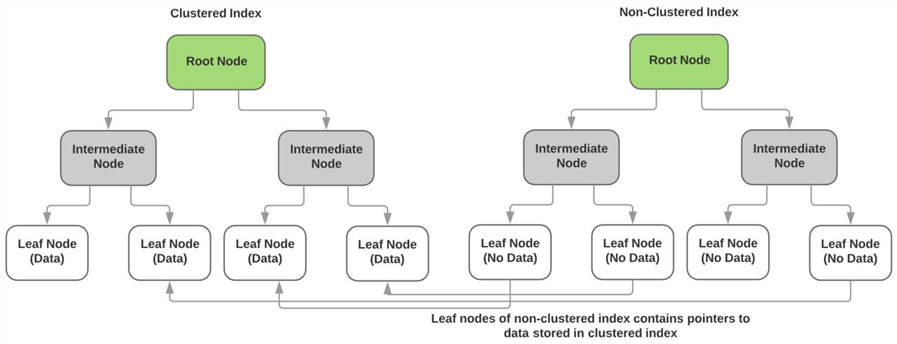
Some of the Pros and Cons of using Partition is listed below:

|  |  |
| --- | --- |
| Pros | Cons |
| Insert/Delete a bunch of data extremely fast by metadata-only operation (Switching) | Negatively affect Query Performance |
| Historic data can be archived by Sliding Window Technique | Maintenance |
| Can be automated by SP and Jobs | SQL Server does not automatically handle switching |

# **Clustered Index:**

[[Link](http://www.sqlservertutorial.net/sql-server-indexes/sql-server-clustered-indexes/)]





# **Clustered vs Non-Clustered Indexes:**

[[Link](https://www.giantstride.gr/sql-indexing-part2/)]

|  |  |  |
| --- | --- | --- |
|  | CLUSTERED | NON-CLUSTERED |
| PROS | * Fast to return large range of data * Fast for presorted results | * Wide keys do not reflect on other indexes * Frequently updated key columns do not reflect on other indexes * Can be assigned on different FileGroup * Many non-clustered indexes per table * Smaller size than clustered indexes due to column subsets |
| CONS | * Frequently updated key columns reflect on non-clustered indexes * Wide keys increase the size of the non-clustered indexes * Only one clustered index per table | * Generally slower than clustered indexes due to bookmark lookup (except for covering indexes). * Not recommended for returning large data sets (except for covering indexes). |

# Steps on creating a Partitioned Table:

1. **Partition Function:**

Create a Partition Function with all the ranges for the partition. Specify whether we want RANGE LEFT (default) or RANGE RIGHT. With datetime, it is usually easier to use RANGE RIGHT.

-- Create Partition Function

CREATE PARTITION FUNCTION TEST\_PartitionFunc (DATETIME2(0))

AS RANGE RIGHT FOR VALUES

(

DATEADD(dd,-4,CAST(SYSDATETIME() AS DATE)),

DATEADD(dd,-3,CAST(SYSDATETIME() AS DATE)),

DATEADD(dd,-2,CAST(SYSDATETIME() AS DATE)),

DATEADD(dd,-1,CAST(SYSDATETIME() AS DATE)),

CAST(SYSDATETIME() AS DATE)

); -- 6 values

GO

1. **Filegroup**:

Add Filegroups to the database (in this case, it is [PartitionThis]). The number of filegroups should be at least one more than the partition ranges. It is also a good practice to have two empty filegroups at either sides of the partitions like a sandwich. For our example, we are only using one empty filegroup.

-- Adding FileGroups

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG1

GO

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG2

GO

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG3

GO

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG4

GO

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG5

GO

ALTER DATABASE PartitionThis ADD FILEGROUP TEXT\_FG6

GO

1. **Files**:

Add Files with filename, size, filegrowth to each of the Filegroups. Here is a script that creates 6 files dynamically.

--Add files to the filegroups

--This is being done dynamically so it will work on different instances,

--but it makes some big assumptions!

DECLARE @path NVARCHAR(256), @i TINYINT=1, @sql NVARCHAR(4000);

SELECT TOP 1 @path=LEFT(physical\_name,LEN(physical\_name)-4)

FROM sys.database\_files WHERE name='PartitionThis';

WHILE @i <= 6

BEGIN

SET @sql=N'ALTER DATABASE PartitionThis ADD FILE (name=TEXT\_F' +

CAST(@i AS NCHAR(1))+', filename=''' + @path + N'TEST\_F'+

CAST(@i AS NCHAR(1))+'.ndf' + ''', size=128MB, filegrowth=256MB)

TO FILEGROUP TEXT\_FG'+CAST(@i AS NCHAR(1))

--show the command we're running

RAISERROR (@sql,0,0)

--run it

EXEC sp\_executesql @sql;

SET @i+=1;

END

GO

1. **Partition Scheme:**

Create Partition Scheme with the filegroups.

-- Adding Partition Scheme

CREATE PARTITION SCHEME TEST\_PartitionScheme

AS PARTITION TEST\_PartitionFunc

TO (TEXT\_FG1, TEXT\_FG2, TEXT\_FG3, TEXT\_FG4, TEXT\_FG5, TEXT\_FG6);

1. **Table Creation:**

We can use our Partition Scheme at the time of Table creation. It should be noted that creating partition on an already existing table is possible. But the process is a little complex than this.

-- Create TEST tables

DROP TABLE dbo.TEST\_with\_Partition;

CREATE TABLE dbo.TEST\_with\_Partition

(

date\_time DATETIME2(0) NOT NULL,

id INT NOT NULL,

group\_name VARCHAR(10) NOT NULL

) ON TEST\_PartitionScheme(date\_time)

GO

1. **Querying in Partitioned Table:**

There is no change in writing the queries in a partitioned table. The Query Optimizer designs the optimal execution plan and executes the query accordingly.

SELECT TOP (100) \* FROM dbo.TEST\_with\_Partition;

# **Performance Analysis:**

The scripts for performance analysis can be found in SQL\_Partition\_Performance\_Analysis.sql.

**[Approach Results]**

## **1. Simple SELECT Query**

For Simple SELECT query execution across the same partition

* Elapsed Time: micro seconds 109377 + 93711 + 109377 (w Part)
* Elapsed Time: micro seconds 390614 + 218754 + 218751 (w/o part)

**Result Summary:** In Partition Table, the queries are twice faster due to seeking in the same partition. However, these results may vary across partition

For SELECT query execution across multiple partitions:

**Partitioned Table:**  
– (2056441 rows affected)  
– Table 'TEST\_with\_Partition'. Scan count 3, logical reads 11154, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.  
– Result: CPU time = 687 ms, elapsed time = 13790 ms.

**Non-Partitioned Table:**  
– (2056441 rows affected)  
– Table 'TEST\_without\_Partition'. Scan count 3, logical reads 18588, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.  
– Result: CPU time = 1578 ms, elapsed time = 12635 ms.

**Result Summary:**   
– In Partitioned Table, we require less number of logical reads.   
– Queries are also 2.5 times faster on Partition table (considering CPU time).   
– However, the total elapsed times are similar (Non-Partitioned Table is usually is a little faster because it uses Parallelism)  
– As a result, the faster CPU processing in Partitioned Table is overshadowed by the parallel processing in Non-Partitioned Table

**2. Aggregation SELECT Query**

**Partitioned Table**  
– Result: CPU time = 1563 ms, elapsed time = 842 ms.

**Non-Partitioned Table**  
– Result: CPU time = 1280 ms, elapsed time = 634 ms

**Result Summary:**  
– SQL Server uses Parallelism for both partitioned and non-partitioned tables.  
– Again, queries in the non-partitioned table perform faster than that of partitioned table

**3. Parallel Performance Check:**

Ran 5 different queries on the same on 5 different tabs (on 5 different partitions)

**Partitioned Table**  
– Result: elapsed time = 10220 ms.

**Non-Partitioned Table**  
– Result: elapsed time = 11042 ms

**Result Summary:**  
– Both partition and non-partition tables have similar query speed  
– However, partition table is a little faster

# **Partition SWITCH: [**[**Link**](https://www.brentozar.com/archive/2013/01/sql-server-table-partitioning-tutorial-videos-and-scripts/)**]**

## **SWITCH IN:**

1. Create a new Filegroup

ALTER DATABASE PartitionThis ADD FILEGROUP DailyFG7

1. Add File to the Filegroup

DECLARE @path NVARCHAR(256), @i TINYINT=7, @sql NVARCHAR(4000);

SELECT TOP 1 @path=LEFT(physical\_name,LEN(physical\_name)-4)

FROM sys.database\_files WHERE name='PartitionThis';

RAISERROR(N'The path is: %s',0,0, @path);

WHILE @i = 7

BEGIN

SET @sql=N'ALTER DATABASE PartitionThis ADD FILE (name=DailyF' + CAST(@i AS NCHAR(1))+',

filename=''' + @path + N'F'+ CAST(@i AS NCHAR(1))+'.ndf' + ''',

size=128MB, filegrowth=256MB) TO FILEGROUP DailyFG'+CAST(@i AS NCHAR(1))

--show the command we're running

RAISERROR (@sql,0,0)

--run it

EXEC sp\_executesql @sql

SET @i+=1

END

1. Create a new staging table

--Why are we seeding the identity here?

--What would happen if we didn't?

--Ans: this reduces the amount of contention for locks/resource on the primary table

CREATE TABLE OrdersDailyLoad (

OrderDate DATETIME2(0) NOT NULL,

OrderId int IDENTITY (10001,1) NOT NULL,

OrderName nvarchar(256) NOT NULL

) on [DailyFG6]

GO

1. Add data to our new table

INSERT OrdersDailyLoad(OrderDate, OrderName)

SELECT DATEADD(SECOND, t.N,

DATEADD(dd,1,CAST(SYSDATETIME() AS DATETIME2(0)))) AS OrderDate,

CASE WHEN t.N % 3 = 0 THEN 'Bow and Arrow' WHEN t.N % 2 = 0

THEN 'First Aid Kit'

ELSE 'Pen'

END AS OrderName

FROM ph.tally\_ten\_thousands AS t

WHERE N < = 5000

GO

1. Add necessary constraints, clustered index, boundary check

--Create indexes on our staging table

ALTER TABLE OrdersDailyLoad

ADD CONSTRAINT PKOrdersDailyLoad

PRIMARY KEY CLUSTERED(OrderDate,OrderId)

GO

--Create the aligned NC as well. It can have a different name.

CREATE NONCLUSTERED INDEX NCOrderIdOrdersDailyLoad ON OrdersDailyLoad(OrderId)

GO

--We must disable (or drop) this non-aligned index to make switching work

ALTER INDEX NCOrderNameOrdersDailyNonAligned ON OrdersDaily DISABLE;

GO

--Create two check constraints on the staging table.

--This will ensure data fits in with the allowed range

--for the partition we want to put it in

--Constraints WITH CHECK are required for switching in

--Create one constraint for the "low end"

DECLARE @tsql NVARCHAR(2000)=

'ALTER TABLE OrdersDailyLoad

WITH CHECK

ADD CONSTRAINT CKOrdersDailyLoad\_LowEnd

CHECK (OrderDate >= ''' +

convert(CHAR(10),DATEADD(dd,1,CAST(SYSDATETIME() AS DATE))) + ''')'

--Display what we're running

RAISERROR (@tsql,0,0)

--Run it

EXEC sp\_executesql @tsql;

GO

--Create one constraint for the "high end"

DECLARE @tsql NVARCHAR(2000)=

'ALTER TABLE OrdersDailyLoad

WITH CHECK

ADD CONSTRAINT CKOrdersDailyLoad\_HighEnd

CHECK (OrderDate < ''' +

convert(CHAR(10),DATEADD(dd,2,CAST(SYSDATETIME() AS DATE))) + ''')'

--Display what we're running

RAISERROR (@tsql,0,0)

--Run it

EXEC sp\_executesql @tsql;

1. Add the new Filegroup to the partition scheme

ALTER PARTITION SCHEME DailyPS

NEXT USED DailyFG7

1. Add a new boundary point to our partition function

ALTER PARTITION FUNCTION DailyPF()

SPLIT RANGE (DATEADD(dd,3,CAST(SYSDATETIME() AS DATE)))

GO

1. Apply SWITCH in command

ALTER TABLE OrdersDailyLoad

SWITCH TO OrdersDaily PARTITION 6

## **SWITCH OUT:**

1. Create a new staging out table

--PUT THIS ON THE SAME FILEGROUP YOU'RE SWITCHING OUT OF

CREATE TABLE OrdersDailyOut (

OrderDate DATETIME2(0) NOT NULL,

OrderId int IDENTITY NOT NULL,

OrderName nvarchar(256) NOT NULL

) on [DailyFG2];

GO

1. Add necessary Clustered index (CI)

--Create the primary key our switch out table

ALTER TABLE OrdersDailyOut

ADD CONSTRAINT PKOrdersDailyOut

PRIMARY KEY CLUSTERED(OrderDate,OrderId);

GO

1. Apply SWITCH OUT command

--Switch OUT!

ALTER TABLE OrdersDaily

SWITCH PARTITION 2 TO OrdersDailyOut;

GO

## **MERGE Partition Range:**

1. Get the min partition range

--We want to keep an empty partition on DailyFG1

--But we want to remove the empty partition on DailyFG2 (currently Partition 2)

--Programmatically find the boundary point to merge

DECLARE @MergeBoundaryPoint DATETIME2(0), @msg NVARCHAR(2000);

SELECT @MergeBoundaryPoint = CAST(MIN(range\_val.value) AS DATETIME2(0))

FROM sys.partition\_functions part\_func

JOIN sys.partition\_range\_values range\_val ON part\_func.function\_id=range\_val.function\_id

where part\_func.name='DailyPF'

--PRINT(@MergeBoundaryPoint);

1. Check if there is no record in that partition

- If true, merge the partition range

IF (

SELECT COUNT(\*)

FROM dbo.OrdersDaily

WHERE OrderDate < dateadd(DAY, 1, @MergeBoundaryPoint)

) =0

BEGIN

SET @msg='No records found, merging boundary point '

+ CAST(@MergeBoundaryPoint AS CHAR(10)) + '.'

RAISERROR (@msg,0,0)

ALTER PARTITION FUNCTION DailyPF ()

MERGE RANGE ( @MergeBoundaryPoint )

END

ELSE

BEGIN

SET @msg='ERROR: Records exist around boundary point '

+ CAST(@MergeBoundaryPoint AS CHAR(10)) + '. Not merging.'

RAISERROR (@msg,16,1)

END

1. DROP the switch out table

--Let's go ahead and drop the switch OUT table

--(Assuming we don't want to do anything with the rows we switched out!)

DROP TABLE OrdersDailyOut;

GO