<https://www.littlekendra.com/2017/01/31/which-filegroup-is-that-partition-using-how-many-rows-does-it-have/>

# Which Filegroup is that Partition Using? How Many Rows Does It Have?

By Kendra Little, January 31, 2017 [**3**](https://www.littlekendra.com/2017/01/31/which-filegroup-is-that-partition-using-how-many-rows-does-it-have/#comments)

[](http://www.littlekendra.com/wp-content/uploads/2017/01/partitioning-by-ikea.png)Table Partitioning in SQL Server has a bit of a learning curve. It’s tricky to just figure out how much data you have and where the data is stored.

When you’re designing or managing partitioned tables, it’s useful to quickly verify:

* Which tables are partitioned
* The type of partition function they use (left or right)
* Which boundary points are assigned to which filegroup
* How many rows and pages are in each partition (and which boundary point they’re associated with)

This helps make sure that you’re designing your tables correctly, and it also helps you avoid goofs like merging the wrong boundary point and causing a bunch of data to move into another– which can be slow and painful.

All this information is available in TSQL, it’s just an ugly query, and it doesn’t come in any built-in reports or views.

So I’ve got an ugly query for you!

## Query Listing Partitioned Tables with Boundary Point, Filegroup, Row Count, Partition Size, and Partition Number By Index

This query gives an overview of partitioned tables and indexes in a database. The query is [also in a Gist, if you prefer](https://gist.github.com/LitKnd/1635ac3f5cf08b5f84c974ca4b5edf6a).

SELECT

sc.name + N'.' + so.name as [Schema.Table],

si.index\_id as [Index ID],

si.type\_desc as [Structure],

si.name as [Index],

stat.row\_count AS [Rows],

stat.in\_row\_reserved\_page\_count \* 8./1024./1024. as [In-Row GB],

stat.lob\_reserved\_page\_count \* 8./1024./1024. as [LOB GB],

p.partition\_number AS [Partition #],

pf.name as [Partition Function],

CASE pf.boundary\_value\_on\_right

WHEN 1 then 'Right / Lower'

ELSE 'Left / Upper'

END as [Boundary Type],

prv.value as [Boundary Point],

fg.name as [Filegroup]

FROM sys.partition\_functions AS pf

JOIN sys.partition\_schemes as ps on ps.function\_id=pf.function\_id

JOIN sys.indexes as si on si.data\_space\_id=ps.data\_space\_id

JOIN sys.objects as so on si.object\_id = so.object\_id

JOIN sys.schemas as sc on so.schema\_id = sc.schema\_id

JOIN sys.partitions as p on

si.object\_id=p.object\_id

and si.index\_id=p.index\_id

LEFT JOIN sys.partition\_range\_values as prv on prv.function\_id=pf.function\_id

and p.partition\_number=

CASE pf.boundary\_value\_on\_right WHEN 1

THEN prv.boundary\_id + 1

ELSE prv.boundary\_id

END

/\* For left-based functions, partition\_number = boundary\_id,

for right-based functions we need to add 1 \*/

JOIN sys.dm\_db\_partition\_stats as stat on stat.object\_id=p.object\_id

and stat.index\_id=p.index\_id

and stat.index\_id=p.index\_id and stat.partition\_id=p.partition\_id

and stat.partition\_number=p.partition\_number

JOIN sys.allocation\_units as au on au.container\_id = p.hobt\_id

and au.type\_desc ='IN\_ROW\_DATA'

/\* Avoiding double rows for columnstore indexes. \*/

/\* We can pick up LOB page count from partition\_stats \*/

JOIN sys.filegroups as fg on fg.data\_space\_id = au.data\_space\_id

ORDER BY [Schema.Table], [Index ID], [Partition Function], [Partition #];

GO

1. SELECT
2. sc.name + N'.' + so.name as [Schema.Table],
3. si.index\_id as [Index ID],
4. si.type\_desc as [Structure],
5. si.name as [Index],
6. stat.row\_count AS [Rows],
7. stat.in\_row\_reserved\_page\_count \* 8./1024./1024. as [In-Row GB],
8. stat.lob\_reserved\_page\_count \* 8./1024./1024. as [LOB GB],
9. p.partition\_number AS [Partition #],
10. pf.name as [Partition Function],
11. CASE pf.boundary\_value\_on\_right
12. WHEN 1 then 'Right / Lower'
13. ELSE 'Left / Upper'
14. END as [Boundary Type],
15. prv.value as [Boundary Point],
16. fg.name as [Filegroup]
17. FROM sys.partition\_functions AS pf
18. JOIN sys.partition\_schemes as ps on ps.function\_id=pf.function\_id
19. JOIN sys.indexes as si on si.data\_space\_id=ps.data\_space\_id
20. JOIN sys.objects as so on si.object\_id = so.object\_id
21. JOIN sys.schemas as sc on so.schema\_id = sc.schema\_id
22. JOIN sys.partitions as p on
23. si.object\_id=p.object\_id
24. and si.index\_id=p.index\_id
25. LEFT JOIN sys.partition\_range\_values as prv on prv.function\_id=pf.function\_id
26. and p.partition\_number=
27. CASE pf.boundary\_value\_on\_right WHEN 1
28. THEN prv.boundary\_id + 1
29. ELSE prv.boundary\_id
30. END
31. /\* For left-based functions, partition\_number = boundary\_id,
32. for right-based functions we need to add 1 \*/
33. JOIN sys.dm\_db\_partition\_stats as stat on stat.object\_id=p.object\_id
34. and stat.index\_id=p.index\_id
35. and stat.index\_id=p.index\_id and stat.partition\_id=p.partition\_id
36. and stat.partition\_number=p.partition\_number
37. JOIN sys.allocation\_units as au on au.container\_id = p.hobt\_id
38. and au.type\_desc ='IN\_ROW\_DATA'
39. /\* Avoiding double rows for columnstore indexes. \*/
40. /\* We can pick up LOB page count from partition\_stats \*/
41. JOIN sys.filegroups as fg on fg.data\_space\_id = au.data\_space\_id
42. ORDER BY [Schema.Table], [Index ID], [Partition Function], [Partition #];
43. GO

SELECT

sc.name + N'.' + so.name as [Schema.Table],

si.index\_id as [Index ID],

si.type\_desc as [Structure],

si.name as [Index],

stat.row\_count AS [Rows],

stat.in\_row\_reserved\_page\_count \* 8./1024./1024. as [In-Row GB],

stat.lob\_reserved\_page\_count \* 8./1024./1024. as [LOB GB],

p.partition\_number AS [Partition #],

pf.name as [Partition Function],

CASE pf.boundary\_value\_on\_right

WHEN 1 then 'Right / Lower'

ELSE 'Left / Upper'

END as [Boundary Type],

prv.value as [Boundary Point],

fg.name as [Filegroup]

FROM sys.partition\_functions AS pf

JOIN sys.partition\_schemes as ps on ps.function\_id=pf.function\_id

JOIN sys.indexes as si on si.data\_space\_id=ps.data\_space\_id

JOIN sys.objects as so on si.object\_id = so.object\_id

JOIN sys.schemas as sc on so.schema\_id = sc.schema\_id

JOIN sys.partitions as p on

si.object\_id=p.object\_id

and si.index\_id=p.index\_id

LEFT JOIN sys.partition\_range\_values as prv on prv.function\_id=pf.function\_id

and p.partition\_number=

CASE pf.boundary\_value\_on\_right WHEN 1

THEN prv.boundary\_id + 1

ELSE prv.boundary\_id

END

/\* For left-based functions, partition\_number = boundary\_id,

for right-based functions we need to add 1 \*/

JOIN sys.dm\_db\_partition\_stats as stat on stat.object\_id=p.object\_id

and stat.index\_id=p.index\_id

and stat.index\_id=p.index\_id and stat.partition\_id=p.partition\_id

and stat.partition\_number=p.partition\_number

JOIN sys.allocation\_units as au on au.container\_id = p.hobt\_id

and au.type\_desc ='IN\_ROW\_DATA'

/\* Avoiding double rows for columnstore indexes. \*/

/\* We can pick up LOB page count from partition\_stats \*/

JOIN sys.filegroups as fg on fg.data\_space\_id = au.data\_space\_id

ORDER BY [Schema.Table], [Index ID], [Partition Function], [Partition #];

GO

## Column Definitions and Notes

* **Schema.Table**: Schema name concatenated with table name
* **Index ID**: Included for reference and ordering
* **Structure**: This will decode if it’s a partitioned heap, clustered index, nonclustered index, clustered columnstore index, or nonclustered columnstore index
* **Index Name**: What it sounds like
* **Rows**: Number of rows in that partition
* **In-Row GB**: Reserved in-row pages for that partition
* **LOB GB**: Reserved LOB pages for that partition (reminder – columnstore indexes use LOB pages)
* **Partition #**: This can be useful in some queries. Remember that [partition numbers are reassigned when you modify your partition function](https://www.littlekendra.com/2016/06/07/merging-boundary-points-does-partition_number-changing-indicate-data-movement/) (split/merge)
* **Partition Function Name**: The partition function is the “algorithm” that defines the boundary points for the partitions
* **Boundary Type**: Whether the boundary point is a “right” type (lower inclusive boundary) or a “left” type (upper inclusive boundary)
* **Boundary Point**: The value of the boundary point that goes with that particular partition
* **Filegroup**: Where the data is located (defined by the partition scheme)

If you need to know the partition scheme name, it’s easy to add that column in (sys.partition\_schemes is already in the query). The partition scheme is what maps your partition function to the filegroups. In most cases, people just want to know where things currently are, so I left that out of the query.