**PARTITIONS PROGRAMMABILITY**

Author: Kacper Prusiński

Database implementation: SQL Server 2022

Database: AdventureWorks2022

First version – 1.01

Document version – 1.01

Last Modified – 2025-05-27

List of Procedures and functions used to creating partition scheme, function, table, updating or removing objects.

SQL queries for creating and managing partitioned tables can be found in the PARTITIONS.sql and PARTITIONS\_ALL\_TO\_PRIMARY.sql scripts. The procedures defined in PARTITIONS\_PROGRAMMABILITY.sql manage data and partitions exclusively in the PRIMARY filegroup, similar to the approach in PARTITIONS\_ALL\_TO\_PRIMARY.sql.

**Table of Contents**

[**What is a Partition Function?** 3](#_Toc199251946)

[**What is a Partition Scheme?** 3](#_Toc199251947)

[**Functions** 3](#_Toc199251948)

[ **dbo.partition\_function\_name\_pattern** 3](#_Toc199251949)

[ **dbo.partition\_scheme\_name\_pattern** 3](#_Toc199251950)

[ **dbo.is\_table\_partitioned** 3](#_Toc199251951)

[**Views** 4](#_Toc199251952)

[ **dbo.v\_partitioned\_tables** 4](#_Toc199251953)

[**Procedures** 6](#_Toc199251954)

[ **dbo.print\_or\_execute** 6](#_Toc199251955)

[ **dbo.create\_partition\_function** 6](#_Toc199251956)

[ **dbo.create\_partitioned\_scheme** 7](#_Toc199251957)

[ **dbo.prepare\_new\_partition\_table** 7](#_Toc199251958)

[ **dbo.add\_partition** 8](#_Toc199251959)

[ **dbo.prepare\_table** 9](#_Toc199251960)

[ **dbo.drop\_table** 11](#_Toc199251961)

# **What is a Partition Function?**

A Partition Function in SQL Server defines how column values (e.g., dates or IDs) are **split across partitions**. It uses **boundary values** to segment the data logically.

# **What is a Partition Scheme?**

A Partition Scheme tells SQL Server **where to store each partition’s data** — i.e., which filegroup each partition should go to. It connects the **logical partitioning (function)** to **physical storage (filegroups)**.

**Our procedures connect all partitions to the main filegroup – PRIMARY.**

**Functions:**

* **dbo.partition\_function\_name\_pattern** - Create new partition function name pattern
  + input params:
    - @table\_schema (varchar(60)) – schema name (default dbo)
    - @table\_name (varchar(120)) – name of new table (before creating)
  + Output params:
    - @part\_func\_tab (nvarchar(300)) – function name: „[schema]\_[table]\_PF”
* **dbo.partition\_scheme\_name\_pattern** - Create new partition scheme name pattern
  + input params:
    - @table\_schema (varchar(60)) – schema name (default dbo)
    - @table\_name (varchar(120)) – name of new table (before creating)
    - @partition\_column (varchar(50)) – name of column by which you will partition a table
  + Output params:
    - @part\_scheme (nvarchar(300)) – scheme name: „[schema]\_[table]\_Scheme\_[column]”
* **dbo.is\_table\_partitioned** – based on the view dbo.v\_partitioned\_tables, returns one of the following values: -1, 0, or 1, depending on whether the table exists and whether it is partitioned.
  + Input params:
    - @table\_schema (varchar(60)) – schema name (default dbo)
    - @table\_name (varchar(120)) – name of new table (before creating)
  + Output params - @is\_partitioned int
    - -1 – table does not exists
    - 0 – table is not partitioned
    - 1 – table is partitioned

# **Views**

* **dbo.v\_partitioned\_tables** – returns information about tables, partition functions and schemes –is a table partitioned or not partitioned, returns function name, scheme, column, ranges, partition numer, range values etc.
  + CREATE OR ALTER VIEW dbo.v\_partitioned\_tables AS

SELECT tp.object\_id, tp.table\_name, tp.schema\_name,

CASE

WHEN tp.function\_id IS NULL THEN 'NON PARTITIONED'

WHEN tp.function\_id IS NOT NULL AND OBJECT\_ID IS NOT NULL THEN 'PARTITIONED'

WHEN tp.function\_id IS NOT NULL AND OBJECT\_ID IS NULL AND tp.data\_space\_id IS NOT NULL THEN 'EXISTS ONLY FUNCTION AND SCHEME'

WHEN tp.function\_id IS NOT NULL AND OBJECT\_ID IS NULL AND tp.data\_space\_id IS NULL THEN 'EXISTS ONLY FUNCTION'

END AS is\_partitioned,

pf.name function\_name, ps.name scheme\_name, tp.part\_column\_name, tp.type, tp.precision, tp.scale,

tp.partition\_number, prv.boundary\_id, COALESCE(prv.value, tp.value1) VALUE, tp.rows, ps.data\_space\_id, pf.function\_id

FROM sys.partition\_functions pf

LEFT JOIN sys.partition\_schemes ps ON pf.function\_id = ps.function\_id

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

FULL JOIN (

SELECT t.name table\_name, s.name schema\_name, c.name part\_column\_name, ty.name type, c.precision, c.scale, t.object\_id, ps.function\_id, ps.data\_space\_id, prv.value value1, p.partition\_number, p.rows

FROM sys.tables t

LEFT JOIN sys.schemas s ON t.schema\_id = s.schema\_id

LEFT JOIN sys.indexes i

ON t.object\_id = i.object\_id AND i.index\_id <= 1

LEFT JOIN sys.index\_columns ic

ON i.object\_id = ic.object\_id AND i.index\_id = ic.index\_id AND ic.partition\_ordinal = 1

LEFT JOIN sys.columns c

ON ic.column\_id = c.column\_id AND c.object\_id = t.object\_id

LEFT JOIN sys.types ty

ON c.system\_type\_id = ty.system\_type\_id

LEFT JOIN sys.partitions p

ON i.object\_id = p.object\_id AND i.index\_id = p.index\_id

LEFT JOIN sys.partition\_schemes ps

ON i.data\_space\_id = ps.data\_space\_id

LEFT JOIN sys.partition\_functions pf

ON ps.function\_id = pf.function\_id

LEFT JOIN sys.partition\_range\_values prv

ON prv.function\_id = pf.function\_id and prv.boundary\_id = p.partition\_number

) tp ON pf.function\_id = tp.function\_id AND prv.boundary\_id = tp.partition\_number;

* + Output columns:
    - Object\_id (if table exists)
    - Table\_name (if table exists)
    - Schema\_name (if table exists)
    - Is\_partitioned – is or is not partitioned or exists only function or function with scheme
    - Function\_name – partition function (if table is partitioned)
    - Scheme\_name – partition scheme (if table is partitioned)
    - Part\_column\_name – partition column name
    - Type – partition column data type
    - Precision – precision of partition column data type
    - Scale – scale of partition column data type
    - Partition\_number
    - Boundary\_id – partition numer
    - Value – partition range value (sql\_variant)
    - Rows – amount of rows in particular partition
    - Data\_space\_id – partition scheme ID
    - Function\_id – partition function ID
  + Sources
    - Sys.tables – view with tables info in database
    - Sys.schemas – view with schemas info in database
    - Sys.indexes – view with indexes info in database
    - Sys.indexes\_columns – other info about this indexes
    - Sys.columns – view with info about columns in particular tables in database
    - Sys.types – data types
    - Sys.partitions – view info with any partitions in database
    - Sys.partition\_schemes – view with partition schemes info
    - Sys.partition\_function – view with partition schemes info
    - Sys.partition\_range\_values – view with ranges of any partition (function)

# **Procedures**

* **dbo.print\_or\_execute** – procedure used to only print, only execute or print and execute sql query
  + input params:
    - @print\_execute varchar(2)
      * P – only print
      * E – only execute
      * PE or Null - print sql and run it
    - @sql nvarchar(MAX) – sql query
* **dbo.create\_partition\_function**  - procedure used to create new partition table
  + input params:
    - **@**table\_schema (varchar(60)) – default dbo; schema
    - **@**table\_name (varchar(120)) new table name, not null
    - **@**ranges (nvarchar(MAX)) – values of partitions, the delimiter is ‘,’ for example „a,b,c,…”. @ranges, can be null
    - @type\_values (nvarchar(20)) – range values data type – not null
    - @range\_kind (nvarchar(1)) – L – LEFT, R – RIGHT range, not null
    - @print\_execute varchar(2) - execution mode: P(print), E (execute), PE(print and execute); default PE
  + Examples
    1. EXEC dbo.create\_partition\_function @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @ranges = '2023,2024,2025,2026', @type\_values = 'int', @range\_kind = 'L', @print\_execute = 'P';
    2. EXEC dbo. create\_partition\_function @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @ranges = '2023-12-31,2024-12-31,2025-12-31,2026-12-31', @type\_values = 'Date', @range\_kind = 'L', @print\_execute = 'P';
    3. EXEC dbo.create\_partition\_function @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @ranges = NULL, @type\_values = 'Date', @range\_kind = 'L', @print\_execute = 'P';
  + Result
    1. CREATE PARTITION FUNCTION [Sales\_Order\_Partitioned\_PF] (int) AS RANGE LEFT FOR VALUES (2023,2024,2025,2026)
    2. CREATE PARTITION FUNCTION [Sales\_Order\_Partitioned\_PF] (Date) AS RANGE LEFT FOR VALUES ('2023-12-31', '2024-12-31', '2025-12-31', '2026-12-31')
    3. CREATE PARTITION FUNCTION [Sales\_Order\_Partitioned\_PF] (Date) AS RANGE LEFT FOR VALUES ()
* **dbo.create\_partitioned\_scheme -** procedure used to create new partition table
  + input params
    - **@**table\_schema (varchar(60)) – default dbo; schema
    - **@**table\_name (varchar(120)) new table name, not null
    - @partition\_column varchar(50), -- partition column name
    - @print\_execute varchar(2), - execution mode: P(print), E (execute), PE(print and execute); default PE
  + Example
    1. EXEC dbo.create\_partition\_scheme @table\_schema = 'Sales', @table\_name ='Order\_Partitioned', @partition\_column = 'OrderYear', @print\_execute = 'P';
  + Result
    1. CREATE PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderYear AS PARTITION Sales\_Order\_Partitioned\_PF ALL TO ([PRIMARY])
* **dbo.prepare\_new\_partition\_table** - This procedure is used to create a complete partitioning structure for a table, including the **partition function** and **partition scheme**, based on provided parameters. It internally uses the helper procedures dbo.create\_partition\_function and dbo.create\_partition\_scheme.
  + Input params
    - @table\_schema varchar(60) – table schema, defaults to 'dbo'
    - @table\_name varchar(120) – name of the table for which the partitioning structure should be prepared (required)
    - @partition\_column varchar(50) – name of the column used for partitioning (required)
    - @ranges nvarchar(MAX) – list of boundary values separated by commas, e.g., '2023-12-31,2024-12-31,2025-12-31'. May be NULL
    - @type\_values varchar(20) – data type of the partitioning column, e.g., int, date, varchar(...) (required)
    - @range\_kind varchar(1)– type of partition boundary: L – LEFT or R – RANGE (required)
    - @print\_execute varchar(2) – execution mode: P(print), E (execute), PE(print and execute); default PE
  + Example
    1. EXEC dbo.prepare\_new\_partition\_table @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @partition\_column = 'OrderDate', @ranges = '2023-12-31,2024-12-31,2025-12-31,2026-12-31', @type\_values = 'Date', @range\_kind = 'L', @print\_execute = 'PE';
  + Result
    1. CREATE PARTITION FUNCTION [Sales\_Order\_Partitioned\_PF] (Date) AS RANGE LEFT FOR VALUES ('2023-12-31', '2024-12-31', '2025-12-31', '2026-12-31')

CREATE PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderDate AS PARTITION Sales\_Order\_Partitioned\_PF ALL TO ([PRIMARY])

* **dbo.add\_partition** – add new partition for table by spliting range in partition function. Because table is related to partition scheme and scheme is related to partition function, so in parameters you have to write partition function and scheme name.
  + Input params:
    - @partition\_function (varchar(300)) – partition function of table; required
    - @partition\_scheme (varchar(300)) - partition scheme of table; required
    - @value (nvarchar(100)) – ’’’text or date value’’’ or ’numeric value’; if null – partition will not be created
    - @print\_execute (varchar(2)) – execution mode: P(print), E (execute), PE(print and execute); default PE
  + Examples
    1. EXEC dbo.add\_partition @partition\_function = 'Sales\_Order\_Partitioned\_PF', @partition\_scheme = 'Sales\_Order\_Partitioned\_Scheme\_OrderDate', @value = '''2022-12-31''', @print\_execute = 'PE';
    2. EXEC dbo.add\_partition @partition\_function = 'Sales\_Order\_Partitioned\_PF', @partition\_scheme = 'Sales\_Order\_Partitioned\_Scheme\_OrderYear', @value = '2022', @print\_execute = 'P';
  + Results
    1. ALTER PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderDate NEXT USED [PRIMARY]

ALTER PARTITION FUNCTION Sales\_Order\_Partitioned\_PF() SPLIT RANGE ('2022-12-31')

* + 1. ALTER PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderYear NEXT USED [PRIMARY]

ALTER PARTITION FUNCTION Sales\_Order\_Partitioned\_PF() SPLIT RANGE (2022)

* **dbo.prepare\_table** – prepare table before data loading. If table is non partitioned run delete or truncate query. If table is partitioned, you can add, truncate or remove partition table or run delete query with different condition.
  + Input params:
    - @table\_schema (varchar(60)) – table schema
    - @table\_name (nvarchar(120)) – table name
    - @partition\_column (nvarchar(50)) – partition (or other) column;
    - @value (nvarchar(100)) – value; ‘different type value’
    - @delete\_condition (nvarchar(MAX)) - optional
    - @to\_remove (char(1)) – if table is partitioned, Y – partition will be removed, N – not to remove; default N;

if table is not partitioned - @to\_removed is skiped.

* + - @print\_execute (char(2)) – execution mode: P(print), E (execute), PE(print and execute); default PE
  + Scenarious
    - If @partition\_column, @value and @delete\_condition are NULL – truncate table
    - If @partition\_column, @value and @delete\_condition are NOT NULL – delete with both conditions (first is @partition\_column = @value)
    - If @partition\_column and @value is null but @delete\_condition is not nul or vice versa, then delete data using only one condition.
    - If table is partitioned there is checking an existing the partition. If not exists – create it. If exists – check amount of rows. If it is not empty – truncate it.
  + Examples
    1. EXEC dbo.prepare\_table @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @partition\_column = 'OrderYear', @value = '2027', @delete\_condition = NULL, @to\_remove = NULL, @print\_execute = NULL;
    2. EXEC dbo.prepare\_table @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @partition\_column = 'OrderYear', @value = '2025', @delete\_condition = NULL, @to\_remove = 'Y', @print\_execute = NULL;
    3. EXEC dbo.prepare\_table @table\_schema = 'Sales', @table\_name = 'Order\_Partitioned', @partition\_column = 'OrderYear', @value = '2026', @delete\_condition = NULL, @to\_remove = 'N', @print\_execute = NULL;
    4. EXEC dbo.prepare\_table @table\_schema = 'Sales', @table\_name ='Order\_Partitioned', @partition\_column = NULL, @value = NULL, @delete\_condition = 'OrderYear <= 2024 and OrderID > 10', @to\_remove = NULL, @print\_execute = 'PE';
    5. EXEC dbo.prepare\_table @table\_schema = 'Sales', @table\_name ='Order\_Partitioned',@partition\_column = NULL, @value = NULL, @delete\_condition = NULL, @to\_remove = NULL, @print\_execute = 'PE';
  + Results
    1. ALTER PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderYear NEXT USED [PRIMARY]

ALTER PARTITION FUNCTION Sales\_Order\_Partitioned\_PF() SPLIT RANGE (2027)

* + 1. TRUNCATE TABLE SALES.ORDER\_PARTITIONED WITH(PARTITIONS (4))

ALTER PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderYear NEXT USED [PRIMARY]

ALTER PARTITION FUNCTION Sales\_Order\_Partitioned\_PF() MERGE RANGE (2025)

* + 1. TRUNCATE TABLE SALES.ORDER\_PARTITIONED WITH(PARTITIONS (4))
    2. DELETE FROM SALES.ORDER\_PARTITIONED WHERE OrderYear <= 2024 and OrderID > 10 OPTION (MAXDOP 10)
    3. TRUNCATE TABLE SALES.ORDER\_PARTITIONED
* **dbo.drop\_table** – procedure used to removing table. If table is partitioned remove also partition function and scheme
  + Input params:
    - * @table\_schema (varchar(60)) – table schema
      * @table\_name (varchar(120)) – table name
      * @print\_execute (char(2)) – execution mode: P(print), E (execute), PE(print and execute); default PE
  + Examples
    - 1. EXEC dbo.drop\_table 'Sales', 'Order\_Partitioned', 'PE';
      2. EXEC dbo.drop\_table 'Sales', 'Order\_Non\_Partitioned', 'PE';
    - Results
    1. DROP TABLE Sales.Order\_Partitioned

DROP PARTITION SCHEME Sales\_Order\_Partitioned\_Scheme\_OrderYear

DROP PARTITION FUNCTION Sales\_Order\_Partitioned\_PF

* + 1. DROP TABLE Sales.Order\_Non\_Partitioned

All examples and procedures codes are defined in PARTITIONS\_PROGRAMMABILITY.sql script.