PARTITIONS PROGRAMMABILITY

Author: Kacper Prusiński

Database implementation: SQL Server 2022

Database: AdventureWorks2022

First version – 1.01

Document version – 1.03

Last Modified – 2025-06-07

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?

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. This documantation is about range partitions in Sql Server.

Ranges – Left and Right

Range left – example (> x, <= y):

1. **<=** 2023-12-31
2. **>** 2023-12-31, **<=** 2024-12-31
3. **>** 2024-12-31, **<=** 2025-12-31
4. **>** 2025-12-31, **<=** 2026-12-31

Range right – example (>= x, < y):

1. **<** 2023-12-31
2. **>=** 2023-12-31, **<** 2024-12-31
3. **>=** 2024-12-31, **<** 2025-12-31
4. **>=** 2025-12-31, **<** 2026-12-31

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** – returns the pattern of new partition function based on schema and table name (non-existing yet)
    - 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** - returns the pattern of a new partition scheme based on schema and table name (non-existing yet) and the column by which you can split a table into partitions.
    - 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
    a) 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';
    b) 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';
    c) EXEC dbo.create_partition_function @table_schema = 'Sales', @table_name = 'Order_Partitioned', @ranges = NULL, @type_values = 'Date', @range_kind = 'L', @print_execute = 'P';
  - Result
    a) CREATE PARTITION FUNCTION [Sales_Order_Partitioned_PF] (int) AS RANGE LEFT FOR VALUES (2023,2024,2025,2026)
    b) 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')
    c) 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
        a) EXEC dbo.create_partition_scheme @table_schema = 'Sales', @table_name ='Order_Partitioned', @partition_column = 'OrderYear', @print_execute = 'P';
    - Result
        a) 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 – RIGHT range (required)
        - @print_execute varchar(2) – execution mode: P(print), E (execute), PE(print and execute); default PE
    - Example
        a) 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
    a) 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
        a) EXEC dbo.add_partition @partition_function = 'Sales_Order_Partitioned_PF', @partition_scheme = 'Sales_Order_Partitioned_Scheme_OrderDate', @value = '''2022-12-31''', @print_execute = 'PE';
        b) EXEC dbo.add_partition @partition_function = 'Sales_Order_Partitioned_PF', @partition_scheme = 'Sales_Order_Partitioned_Scheme_OrderYear', @value = '2022', @print_execute = 'P';
    - Results
        a) ALTER PARTITION SCHEME Sales_Order_Partitioned_Scheme_OrderDate NEXT USED [PRIMARY]
        ALTER PARTITION FUNCTION Sales_Order_Partitioned_PF() SPLIT RANGE ('2022-12-31')
        b) 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), the operand is 'AND'
        - 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
        a) 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;
        b) 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;
        c) 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;
        d) EXEC dbo.prepare_table @table_schema = 'Sales', @table_name ='Order_Partitioned',

@partition_column = 'OrderYear', @value = '2024', @delete_condition = 'MONTH(OrderDate) = 11', @to_remove = NULL, @print_execute = 'PE';

e) 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
  a) ALTER PARTITION SCHEME Sales_Order_Partitioned_Scheme_OrderYear NEXT USED [PRIMARY]
  ALTER PARTITION FUNCTION Sales_Order_Partitioned_PF() SPLIT RANGE (2027)
  b) 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)
  c) TRUNCATE TABLE SALES.ORDER_PARTITIONED WITH(PARTITIONS (4))
  d) DELETE FROM SALES.ORDER_PARTITIONED WHERE OrderYear = 2024 AND MONTH(OrderDate) = 11 OPTION (MAXDOP 10)
  e) 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
    a) EXEC dbo.drop_table 'Sales', 'Order_Partitioned', 'PE';
    b) EXEC dbo.drop_table 'Sales', 'Order_Non_Partitioned', 'PE';
  - Results
    a) DROP TABLE Sales.Order_Partitioned
    DROP PARTITION SCHEME Sales_Order_Partitioned_Scheme_OrderYear
    DROP PARTITION FUNCTION Sales_Order_Partitioned_PF
    b) DROP TABLE Sales.Order_Non_Partitioned

All examples and procedures codes are defined in PARTITIONS_PROGRAMMABILITY.sql script.