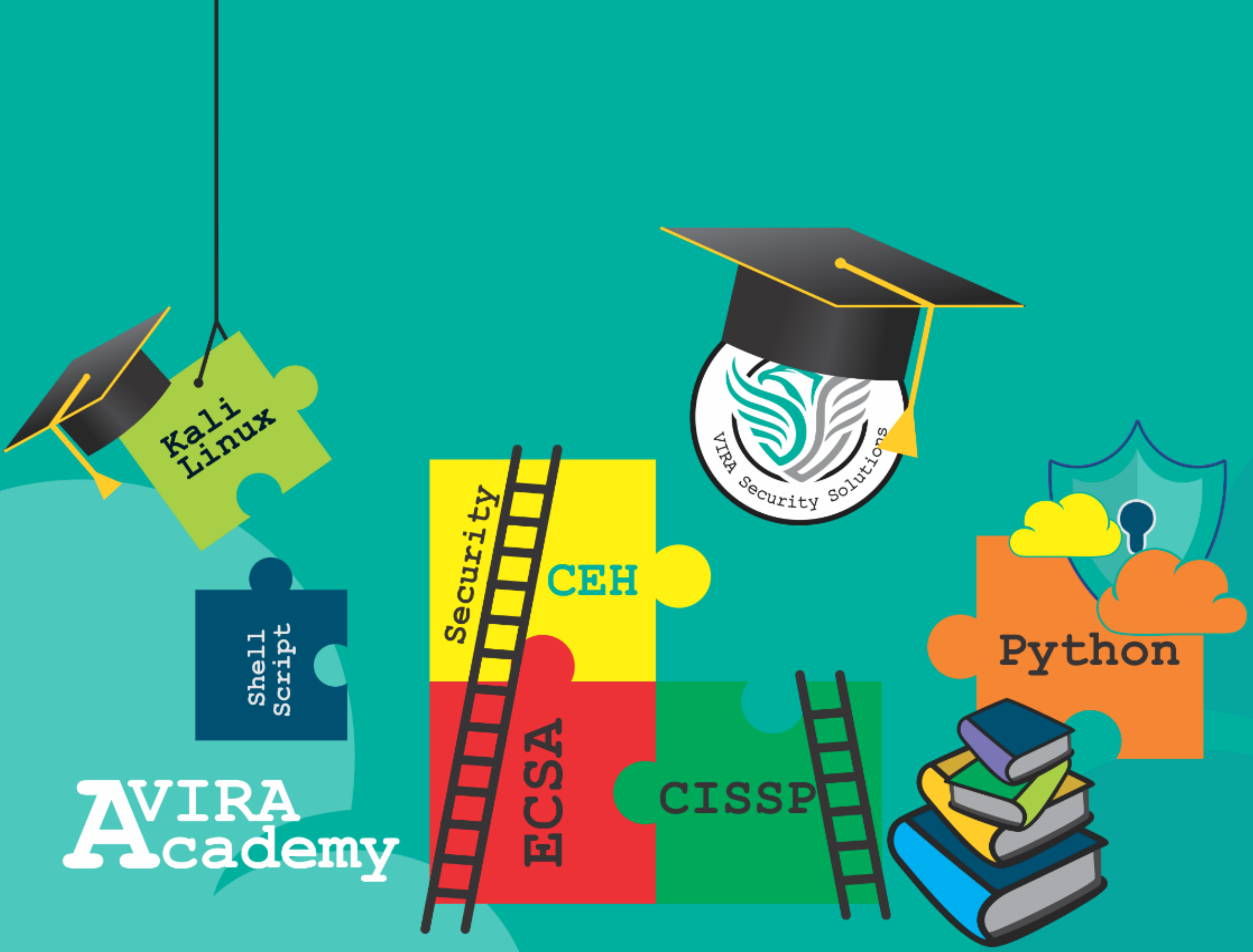
VIRA
Security
Solutions

راهکارهای امن ویرا
شماره ثبت: ۵۴۰۱۳۹

VIRA Security Solutions


Kali Linux
Shell Script
Security
CEH
ECSA
CISSP
Python
VIRA Security Solutions
VIRA Academy

DBA

SQL Server Administration

Part10

Mohammad Reza Gerami
mrgerami@aut.ac.ir
gerami@virasec.ir

Extended Events

Extended Events

Extended events is a lightweight performance monitoring system that enables users to collect data needed to monitor and troubleshoot problems in SQL Server. See Extended events overview to learn more about the extended events architecture. This article aims to help the SQL developer who is new to extended events, and who wants create an event session in just a few minutes. By using extended events, you can see details about the inner operations of the SQL system and your application. When you create an extended event session, you tell the system:

Which occurrences you are interested in.
How you want the system to report the data to you.

Database Snapshots

Database Snapshots

A database snapshot is a read-only, static view of a SQL Server database (the *source database*). The database snapshot is transactionally consistent with the source database as of the moment of the snapshot's creation. A database snapshot always resides on the same server instance as its source database.

BCP

```
bcp "Person.Address" out d:\Address.txt -S "127.0.0.1" -d AdventureWorks2019 -Usa
-Psql -c -T
```

```
bcp "Person.Address" in d:\Address.txt -S "127.0.0.1" -d AdventureWorks2019 -Usa -Psql
-c -T
```

https://docs.microsoft.com/en-us/sql/tools/bcp-utility?view=sql-server-ver15

Table Partitioning in SQL Server

Table Partitioning in SQL Server

What is a database table partitioning?

Partitioning is the database process where very large tables are divided into multiple smaller parts. By splitting a large table into smaller, individual tables, queries that access only a fraction of the data can run faster because there is less data to scan. The main of goal of partitioning is to aid in maintenance of large tables and to reduce the overall response time to read and load data for particular SQL operations.

Table Partitioning

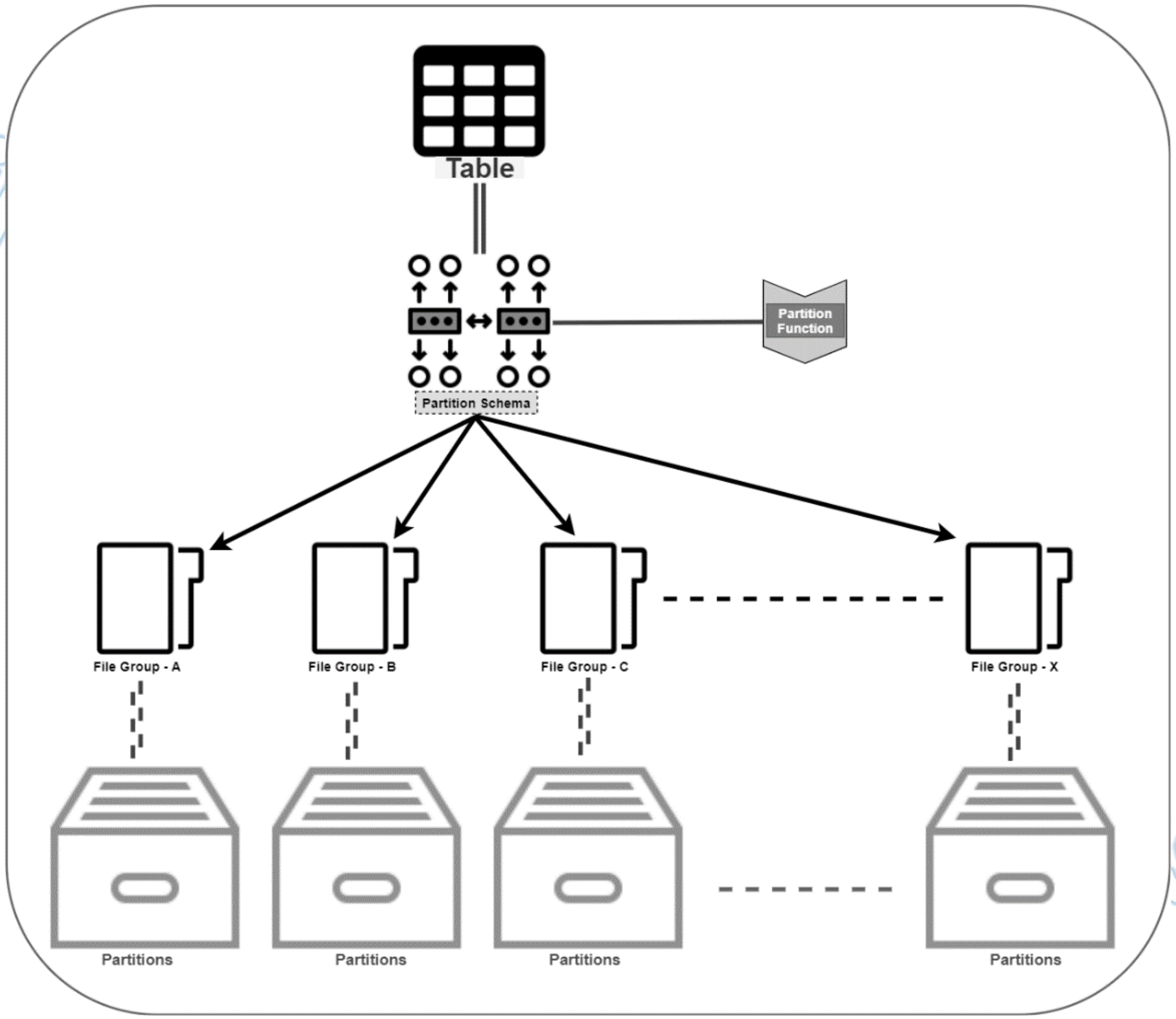
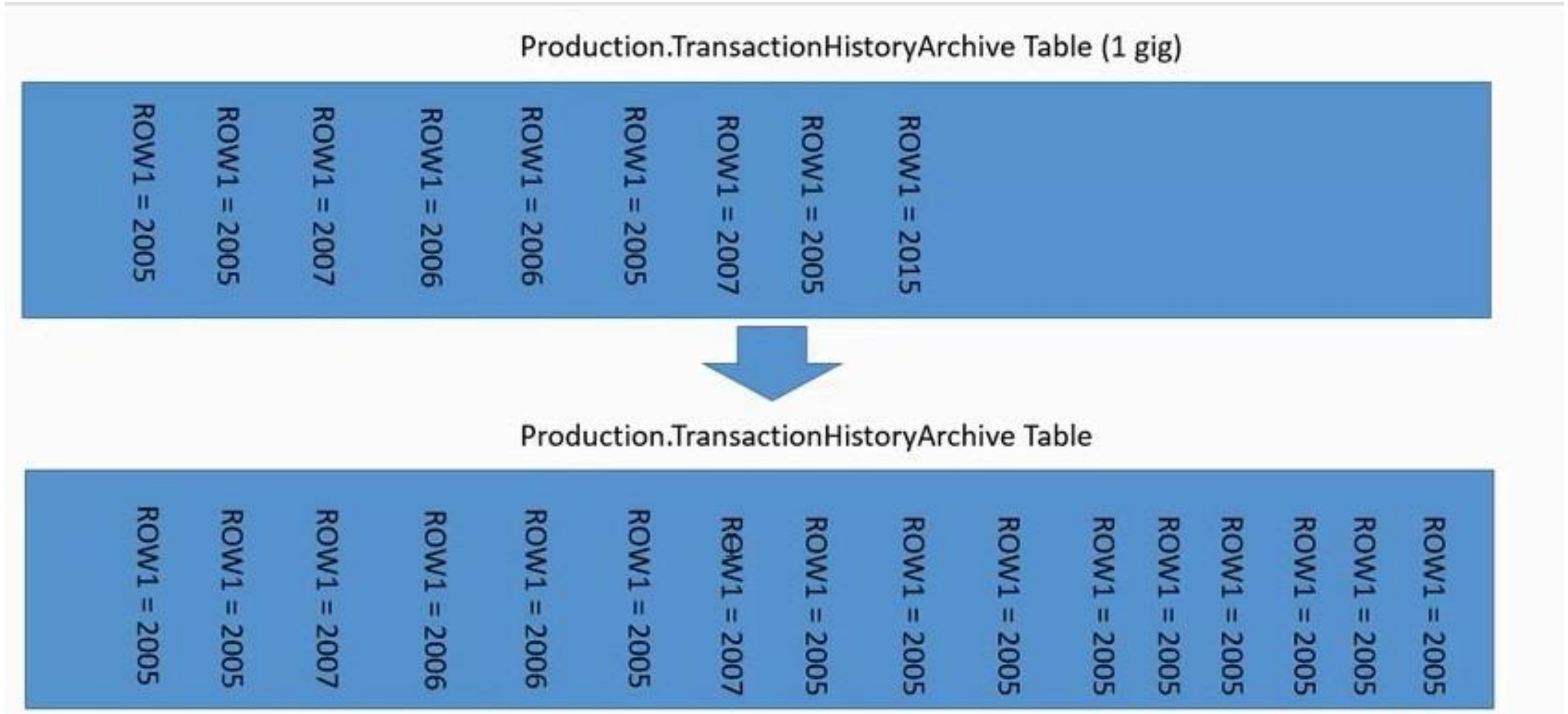
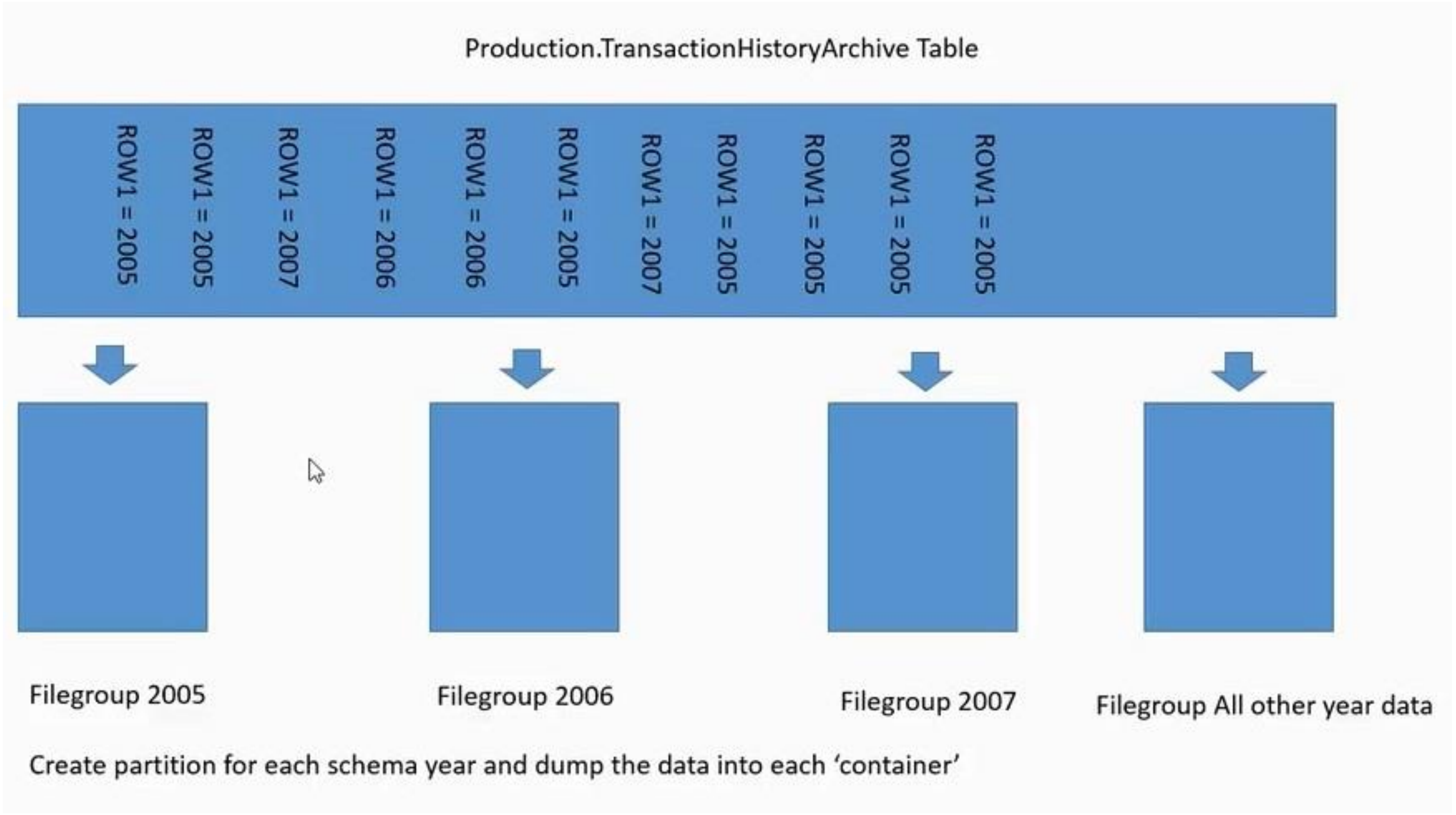


Table Partitioning



As the table gets bigger and bigger in time, it becomes more difficult to manage; the space required, the speed of data retrieval, the increased I/O activity, the hardware resources, the deletes, the inserts, the updates, the selects, the backups and restores all start to press upon the SQL Server and hardware. To remedy this one issue we can consider table partitioning. Table partitioning, simply put, divides the content of the table in to smaller manageable chunks of data in a file group.

Table Partitioning



Create partition for each schema year and dump the data into each 'container'

Table Partitioning

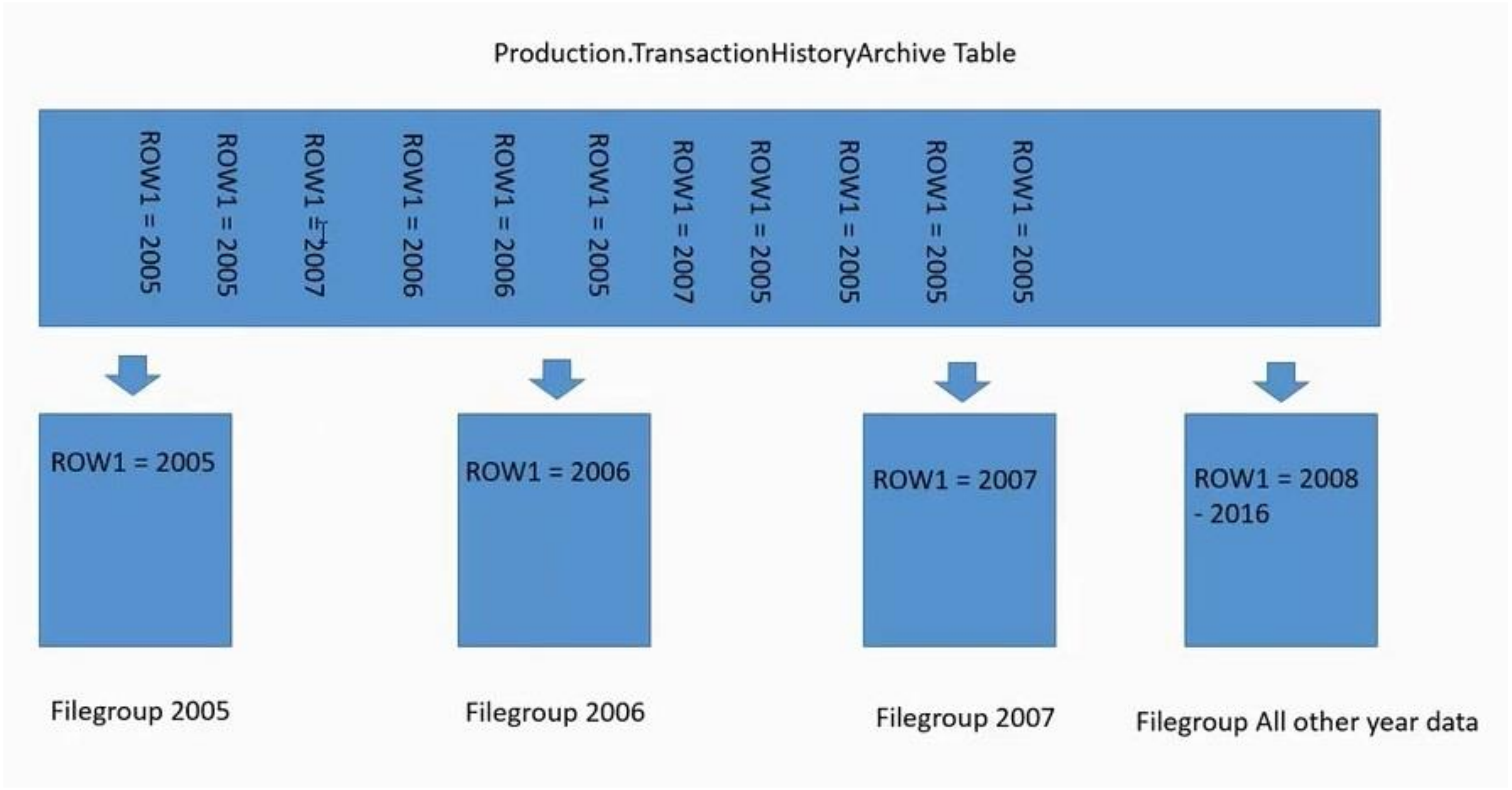


Table Partitioning

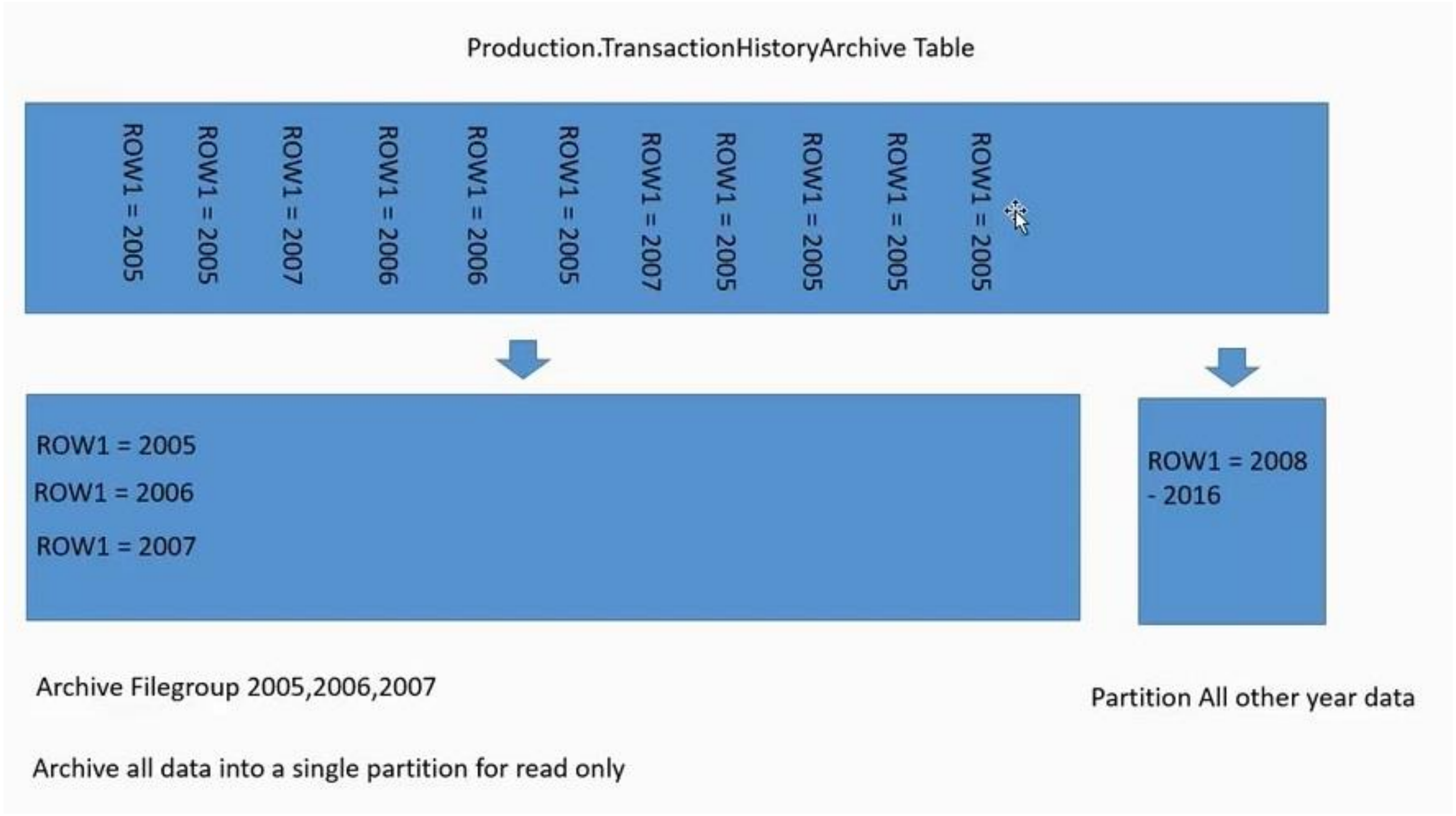


Table Partitioning

What is Table Partitioning in SQL Server?
Why is partitioning required in SQL Server?

Table Partition is the logical division of information with the physical distribution in the filegroups.

A table can be partitioned by applying the partition schema over the table schema. The role of the partition function is to divide the information in a logical division by partition range and partition scheme indexing the partition range to the filegroup.

After applying the partition on the table, rows of the table will be distributed in different secondary files by the filegroup.

Table Partitioning

Partition is one of the beneficial approaches for query performance over the large table. Table Index will be a part of each partition in SQL Server to return a quick query response. Actual partition performance will be achieved when you use a query with the partition column because the partition column will target those partitions only, which are required by the partition column. It wouldn't scan the information in all filegroups.

Table Partitioning

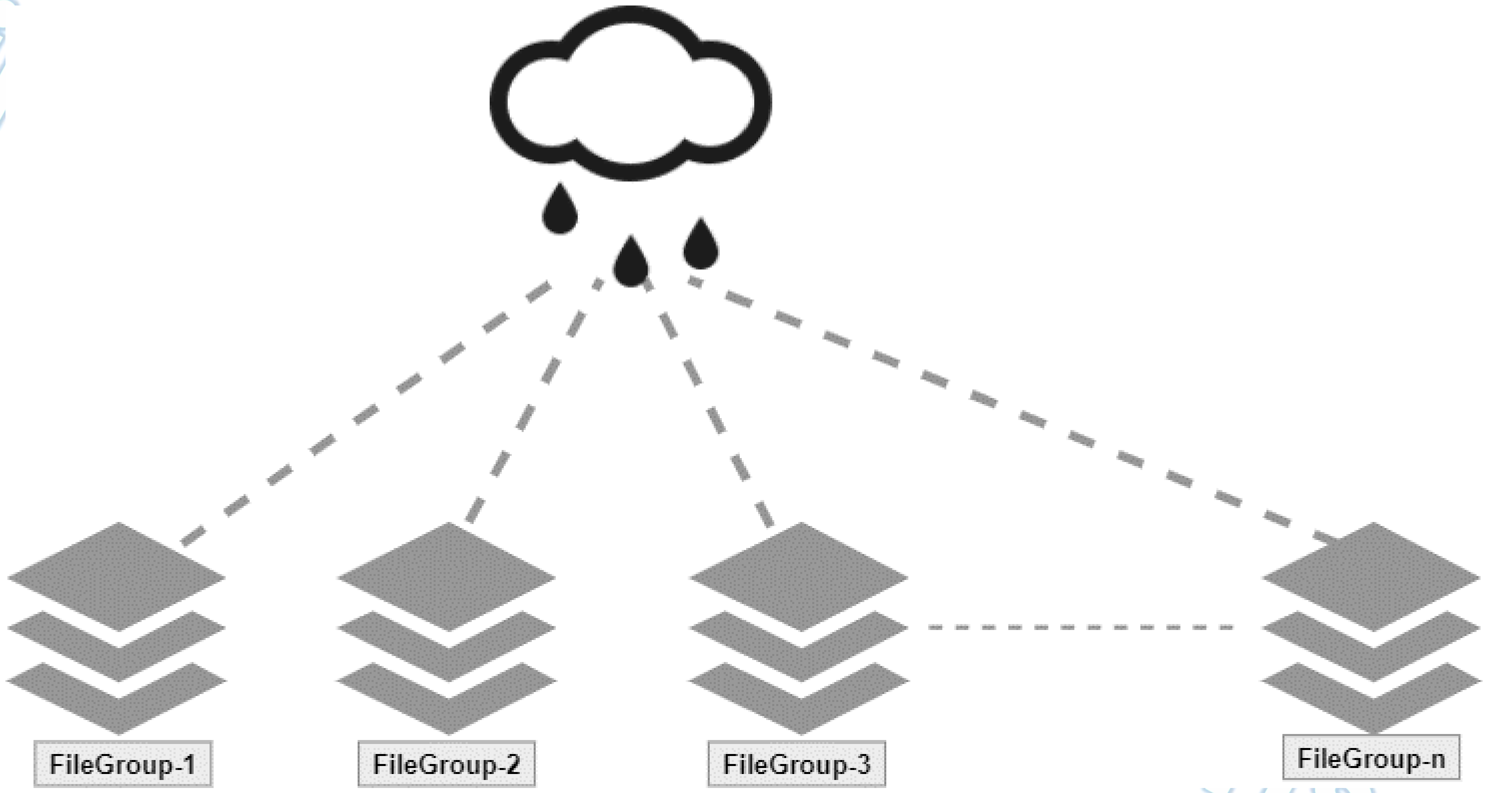


Table Partitioning

Creating a sample database to perform Table Partitioning in SQL Server:

```
CREATE DATABASE AutoPartition
```

Adding File Groups to the database:

```
ALTER DATABASE AutoPartition
ADD FILEGROUP FG_01_2020
GO
```

Here, we are going to set up a month-wise partition for the table. So, initially, we are creating 12 File Groups for the year 2020.

Table Partitioning

Adding Files to each File Group

```
ALTER DATABASE AutoPartition
ADD FILE
(
 NAME = [File_012020],
 FILENAME = 'C:\Program Files\Microsoft SQL Server\MSSQL13.JRAIYANI\MSSQL\DATA\File_012020.ndf',
  SIZE = 5 MB,
  MAXSIZE = UNLIMITED,
  FILEGROWTH = 10 MB
) TO FILEGROUP FG_01_2020
GO
```

Here, adding one file to each filegroup for data distribution to the physical storage to perform the table partitioning in SQL Server. If you have crores of row per day, then you can add multiple files to the single file group as well, and data will be distributed to the number of files in the filegroup.

Table Partitioning

Adding a Partition Function with Month wise range

```
USE AutoPartition
GO
CREATE PARTITION FUNCTION [PF_MonthlyPartition] (DATETIME)
AS RANGE RIGHT FOR VALUES
(
  '2020-01-31 23:59:59.997', '2020-02-29 23:59:59.997', '2020-03-31 23:59:59.997',
  '2020-04-30 23:59:59.997', '2020-05-31 23:59:59.997', '2020-06-30 23:59:59.997',
  '2020-07-31 23:59:59.997', '2020-08-31 23:59:59.997', '2020-09-30 23:59:59.997',
  '2020-10-31 23:59:59.997', '2020-11-30 23:59:59.997', '2020-12-31 23:59:59.997'
);
```

Here, 12 ranges are defined with the last day of the month and last ms of the day. Users can use the month and year combination as well to perform the table partitioning in SQL Server. But I would recommend to define it with full datetime to perform insert operations quickly. In the above sample code, the partition function is defined with the Right direction.

Table Partitioning

Adding a Partition Scheme with File Groups to the Partition Function

```sql
USE AutoPartition
GO
CREATE PARTITION SCHEME PS_MonthWise
AS PARTITION PF_MonthlyPartition
TO
(
  'FG_01_2020', 'FG_02_2020', 'FG_03_2020',
  'FG_04_2020', 'FG_05_2020', 'FG_06_2020',
  'FG_07_2020', 'FG_08_2020', 'FG_09_2020',
  'FG_10_2020', 'FG_11_2020', 'FG_12_2020',
  'Primary'
);
```

Table Partitioning

Adding a filegroup to the partition schema with attaching the partition function

Here, the Primary filegroup is an additional filegroup in the partition scheme definition.

The primary filegroup is used to store those rows which are exceeding the partition range in the function. It works when users forget to add new ranges and new filegroups with the file.

Table Partitioning

Creating a Table with the Partition Scheme

```
USE AutoPartition
GO
CREATE TABLE orders
(
 [order_id] BIGINT IDENTITY(1,1) NOT NULL,
 [user_id] BIGINT,
 [order_amt] DECIMAL(10,2),
 [address_id] BIGINT,
 [status_id] TINYINT,
 [is_active] BIT,
 [order_date] [datetime]
) ON PS_MonthWise ([order_date]);
GO
CREATE CLUSTERED INDEX CI_orders_order_id ON orders(order_id)
GO
CREATE NONCLUSTERED INDEX IX_user_id ON orders(user_id)
GO
```

Here, the table is defined with applying the Partition to the column **[order_date]** of table **orders**.

Table Partitioning

Inserting data into Table [orders]

```
INSERT INTO orders (user_id, order_amt, address_id, status_id, is_active, order_date)
VALUES
(43, 1623.78, 51, 1, 1, '2020-01-14 13:21:51.869'),
(51, 8963.17, 43, 1, 1, '2020-02-17 05:07:43.193'),
(27, 7416.93, 66, 1, 1, '2020-03-21 07:53:07.743'),
(58, 9371.54, 45, 1, 1, '2020-04-26 16:19:27.852'),
(53, 8541.56, 65, 1, 1, '2020-05-08 19:21:58.654'),
(98, 6971.85, 54, 1, 1, '2020-06-17 21:34:52.426'),
(69, 5217.74, 78, 1, 1, '2020-07-03 07:37:51.391'),
(21, 9674.14, 98, 1, 1, '2020-08-27 23:49:53.813'),
(52, 1539.96, 32, 1, 1, '2020-09-01 17:17:07.317'),
(17, 7193.63, 21, 1, 1, '2020-10-23 10:23:37.307'),
(68, 3971.25, 19, 1, 1, '2020-11-30 09:01:27.079'),
(97, 5973.58, 97, 1, 1, '2020-12-06 13:43:21.190'),
(76, 4163.95, 76, 1, 1, '2021-01-03 18:51:17.764')
GO
```

Table Partitioning

Inserting data into Table [orders]

We have inserted sample rows with each partition range of the partition function. Now let’s check how many rows exist with partition and its filegroups.

Table Partitioning

Partition details with Row count

Below are the DMVs that return the number of rows that exist in the filegroup with partition range.

```
SELECT DISTINCT o.name as table_name, rv.value as partition_range, fg.name as file_groupName, p.partition_num
ber, p.rows as number_of_rows
FROM sys.partitions p
INNER JOIN sys.indexes i ON p.object_id = i.object_id AND p.index_id = i.index_id
INNER JOIN sys.objects o ON p.object_id = o.object_id
INNER JOIN sys.system_internals_allocation_units au ON p.partition_id = au.container_id
INNER JOIN sys.partition_schemes ps ON ps.data_space_id = i.data_space_id
INNER JOIN sys.partition_functions f ON f.function_id = ps.function_id
INNER JOIN sys.destination_data_spaces dds ON dds.partition_scheme_id = ps.data_space_id AND dds.destination
_id = p.partition_number
INNER JOIN sys.filegroups fg ON dds.data_space_id = fg.data_space_id
LEFT OUTER JOIN sys.partition_range_values rv ON f.function_id = rv.function_id AND p.partition_number = rv.bou
ndary_id
WHERE o.object_id = OBJECT_ID('orders');
```

Table Partitioning

Here, the partition range was not defined for January 2021. But we added the last file group with the Primary. Therefore, if rows are exceeding the range, then those will be allocated to the primary filegroup.

Table Partitioning

Table Rows with Partition Number

Users can find the partition number with each row of the table as well. Users can bifurcate the row allocation to the logical partition number with the help of a $PARTITION() function.

```
SELECT $PARTITION.PF_MonthlyPartition(order_date) AS PartitionNumber, *
FROM orders
```

Table Partitioning

Automate the Partition flow

We have explained the above examples to understand the requirement of post activities and maintenance on the partition function. A primary filegroup is defined to manage those rows which are out of the partition range. However, the database team has to monitor that the range of any partition function is ending or not? To avoid some manual tasks, users can set up the SQL Server job to perform it automatically.

SQL Server job will be executed in a pre-defined scheduled time (monthly or weekly) and helps to find out the partition functions which are needed to be maintained. In the above sample, we used the DATETIME column type for the partition range. Users can write a program with T-SQL statements as below:
Find the Partition Function whose range does not exist for the next month. For example, in the above example, partition range is defined till Dec 2020, and a current timestamp is 2020-12-27 16:27:09.500. After three days, the partition range will be exceeded because the maximum range is 2020-12-31 23:59:59.997 for the order table. Now, we will find the partition functions which are required to be maintained using the below T-SQL.

Table Partitioning

Automate the Partition flow

```
SELECT o.name as table_name,
  pf.name as PartitionFunction,
  ps.name as PartitionScheme,
  MAX(rv.value) AS LastPartitionRange,
  CASE WHEN MAX(rv.value) <= DATEADD(MONTH, 2, GETDATE()) THEN 1 else 0 END AS isRequiredMainten
ance
--INTO #temp
FROM sys.partitions p
INNER JOIN sys.indexes i ON p.object_id = i.object_id AND p.index_id = i.index_id
INNER JOIN sys.objects o ON p.object_id = o.object_id
INNER JOIN sys.system_internals_allocation_units au ON p.partition_id = au.container_id
INNER JOIN sys.partition_schemes ps ON ps.data_space_id = i.data_space_id
INNER JOIN sys.partition_functions pf ON pf.function_id = ps.function_id
INNER JOIN sys.partition_range_values rv ON pf.function_id = rv.function_id AND p.partition_number = rv.
boundary_id
GROUP BY o.name, pf.name, ps.name
```

Table Partitioning

Automate the Partition flow

Here, the above result set returned the partition function (**PF_MonthlyPartition)** for adding the new range.
The following code helps to insert information to the new temp table for those partition functions that are required to SPLIT.

```
SELECT table_name,
  PartitionFunction,
  PartitionScheme,
  LastPartitionRange,
  CONVERT(VARCHAR, DATEADD(MONTH, 1, LastPartitionRange), 25) AS NewRange,
  'FG_' + CAST(FORMAT(DATEADD(MONTH, 1, LastPartitionRange),'MM') AS VARCHAR(2)) +
    '_' +
   CAST(YEAR(DATEADD(MONTH, 1, LastPartitionRange)) AS VARCHAR(4)) AS NewFileGroup,
  'File_'+ CAST(FORMAT(DATEADD(MONTH, 1, LastPartitionRange),'MM') AS VARCHAR(2)) +
   CAST(YEAR(DATEADD(MONTH, 1, LastPartitionRange)) AS VARCHAR(4)) AS FileName,
  'C:\Program Files\Microsoft SQL Server\MSSQL13.JRAIYANI\MSSQL\DATA\' AS file_path
INTO #generateScript
FROM #temp
WHERE isRequiredMaintenance = 1
```

Table Partitioning

We can also generate a dynamic script to create File Group, File, add a new file group to partition scheme, and new range to the partition function as below:

```
DECLARE @filegroup NVARCHAR(MAX) = ''
DECLARE @file NVARCHAR(MAX) = ''
DECLARE @PScheme NVARCHAR(MAX) = ''
DECLARE @PFunction NVARCHAR(MAX) = ''

SELECT @filegroup = @filegroup +
  CONCAT('IF NOT EXISTS(SELECT 1 FROM AutoPartition.sys.filegroups WHERE name = ''',NewFileGroup,''')
  BEGIN
   ALTER DATABASE AutoPartition ADD FileGroup ',NewFileGroup,'
  END;'),
  @file = @file + CONCAT('IF NOT EXISTS(SELECT 1 FROM AutoPartition.sys.database_files WHERE name = ''',FileName,''')
  BEGIN
  ALTER DATABASE AutoPartition ADD FILE
  (NAME = ''',FileName,''',
  FILENAME = ''',File_Path,FileName,'.ndf'',
  SIZE = 5MB, MAXSIZE = UNLIMITED,
  FILEGROWTH = 10MB )
  TO FILEGROUP ',NewFileGroup, '
  END;'),
  @PScheme = @PScheme + CONCAT('ALTER PARTITION SCHEME ', PartitionScheme, ' NEXT USED ',NewFileGroup,';'),
  @PFunction = @PFunction + CONCAT('ALTER PARTITION FUNCTION ', PartitionFunction, '() SPLIT RANGE (''',NewRange,''');')
FROM #generateScript

EXEC (@filegroup)
EXEC (@file)
EXEC (@PScheme)
EXEC (@PFunction)
```

Test for Connectivity

Test for Connectivity

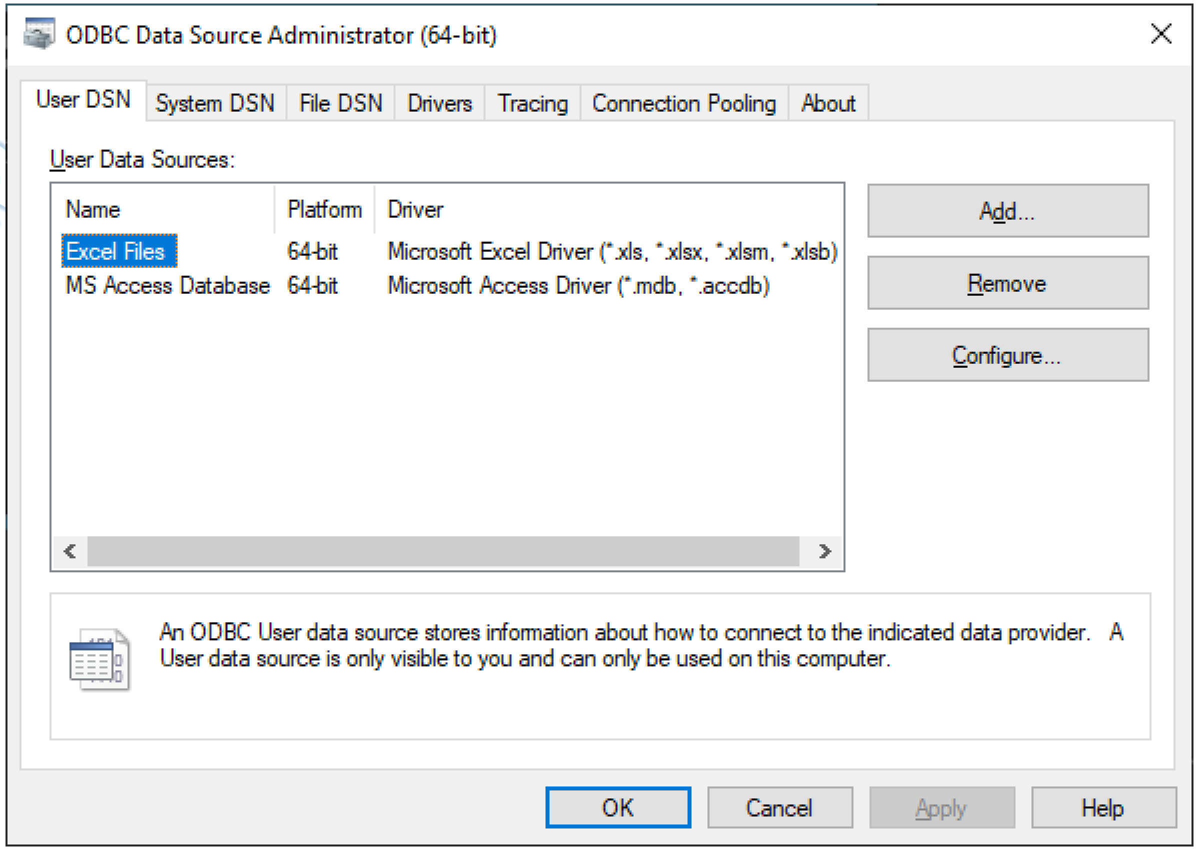
Telnet

ODBC connection Manager

sqlcmd

Test for Connectivity

ODBC connection Manager



ODBC Data Source Administrator (64-bit)

User Data Sources:

Name	Platform	Driver
Excel Files	64-bit	Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
MS Access Database	64-bit	Microsoft Access Driver (*.mdb, *.accdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you and can only be used on this computer.

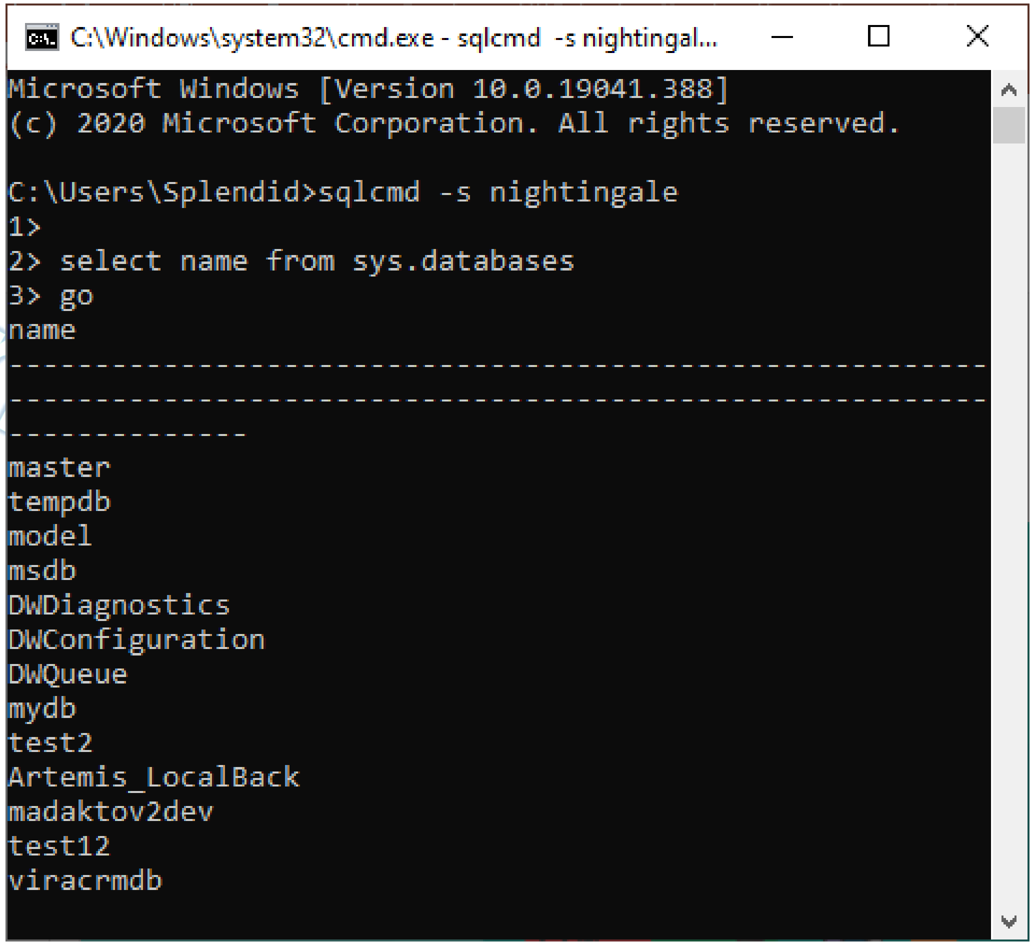
Test for Connectivity

Sqlcmd

sqlcmd -S [ServerName]

sqlcmd -s nightingale

1>select name from sys.databases



```
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Splendid>sqlcmd -s nightingale
1>
2> select name from sys.databases
3> go
name
----------------------------------------------------------
----------------------------------------------------------
--------------
master
tempdb
model
msdb
DWDiagnostics
DWConfiguration
DWQueue
mydb
test2
Artemis_LocalBack
madaktov2dev
test12
viracrmdb
```