Microsoft SQLServer 2008

PARTITION

About Speaker

Shrwan Krishna Shrestha

shrwan@sqlpassnepal.org / shrwan@gmail.com

98510-50947

Topics to be covered

- Partition in earlier versions
- Table Partitioning Overview
- Benefits of Partitioned Table
- Challenges
- Planning of Table Partitioning
- How to Partition a Table
- Partition and Parallel Execution
- Query Performance
- Sliding Window Scenario
- Best Practices

Partition in earlier versions

Till SQLServer 2000 only way to implement partition was to created a view UNIONed with multiple table.

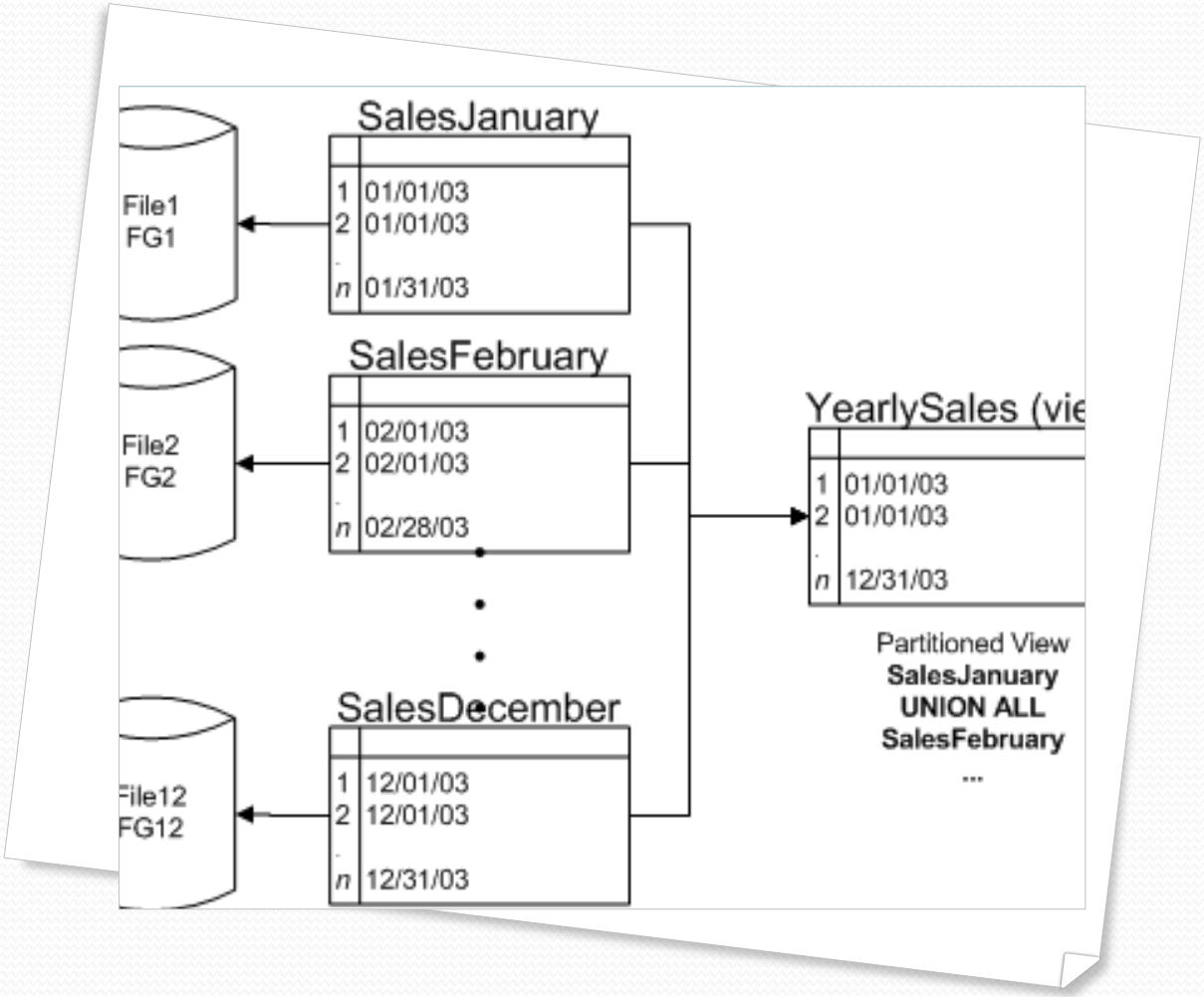
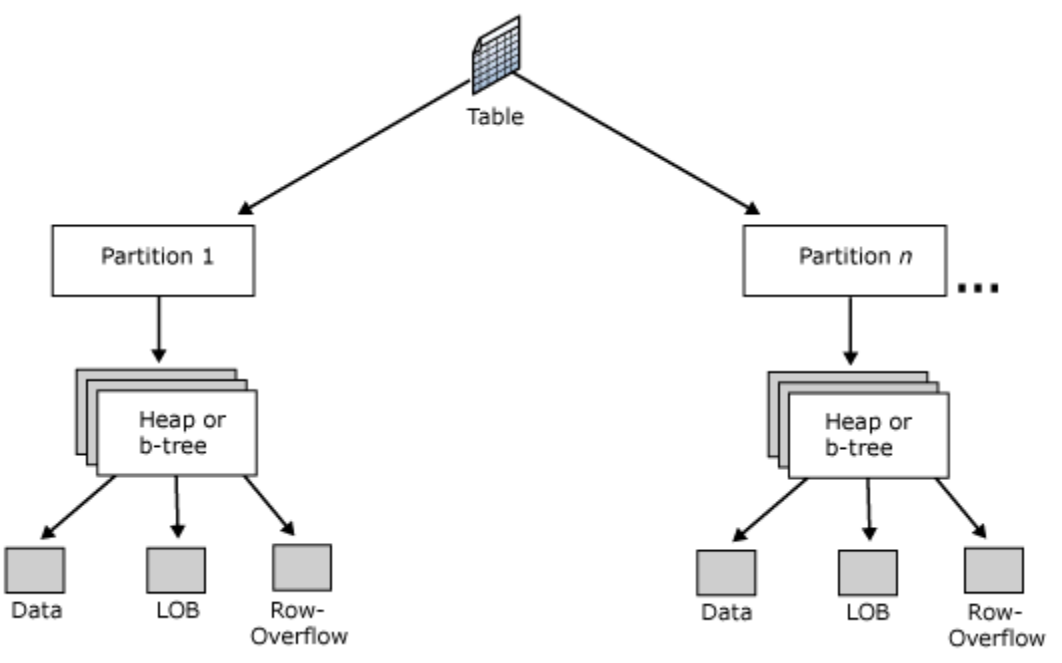


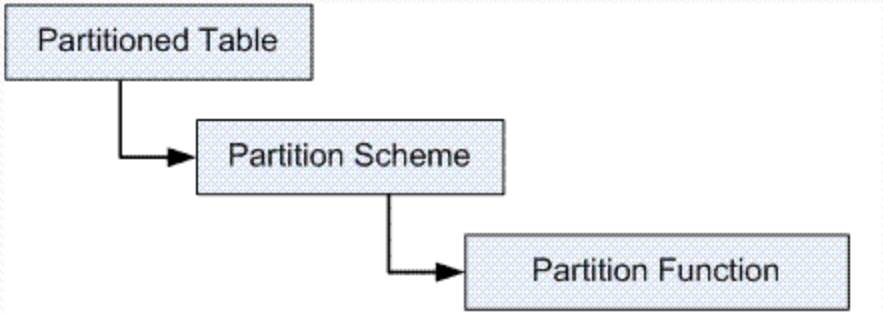
Table Partition Overview

- Column based solution.
- Horizontal Partitions.
- This is a new feature that allows built-in data partitioning that handles the movement of data without physically moving it.
- It gives an advantage of having smaller objects to manage and maintain.
- A hidden computed column is created internally to represent the ID of a table or index partition for a specific row.
- To the users it looks like a table as we were always been using. But data are stored in different partitions not in one large table.

Partitioned Table



Dependencies



Benefits of Partitioning

- SQL Server automatically manages the placement of data in the proper partitions.
- A partitioned table and its indexes appear as a normal database table with indexes, even though the table might have numerous partitions.
- The table can be managed at the partition and filegroup level for ease of maintenance.
- Partitioned tables support easier and faster data loading, aging, and archiving.
- Application queries that are properly filtered on the partition column can perform better by making use of partition elimination and parallelism.
- In cases where partitioned data will not be modified, we can mark some or most of a partitioned table's filegroups as read-only, making management of the filegroups easier.
- In SQL Server 2008, compress can be done to individual partitions as well as control lock escalation at a partition level.

Challenges of Partitioning

- There is a maximum of 1,000 partitions for a table.
- You must manage filegroups and file placement if you place partitions on individual filegroups.
- The metadata-only operations (SWITCH, MERGE, and SPLIT) can be blocked by other DML actions on the table at the time, until a schema-modification lock can be obtained.
- Managing date or time-based data can be complex.
- You cannot rebuild a partitioned index with the ONLINE option set to ON, because the entire table will be locked during the rebuild.
- Automating changes to partitioned tables, as in a sliding window scenario, can be difficult.

Planning for Table Partitioning

- Decide which table can benefit from the increase manageability and availability.
- Plan on how to store the partitions in filegroups using a partition scheme.
- Specify the partition boundaries in a partition function.

How to partition a table

- Create additional filegroups if you want to spread the partition over multiple filegroups.
- Create a Partition Function
- Create a Partition Scheme
- Create the table using the Partition Scheme

Function

- CREATE PARTITION FUNCTION partition_function_name (input_parameter_type)
  AS RANGE [LEFT | RIGHT]
  FOR VALUES ([boundary_value [,...n]]) [;]
  - **Left:** The first value is the maximum value of the first partition.
  - **Right:** The first value is the minimum value of the second partition.

Range Left & Right

- CREATE PARTITION FUNCTION myRangePF1 (int)
  AS RANGE LEFT FOR VALUES (1, 100, 1000);

Partition	1	2	3	4
Values	col1 <= 1	col1 > 1 AND col1 <= 100	col1 > 100 AND col1 <= 1000	col1 > 1000

- CREATE PARTITION FUNCTION myRangePF2 (int)
  AS RANGE RIGHT FOR VALUES (1, 100, 1000);

Partition	1	2	3	4
Values	col1 < 1	col1 >= 1 AND col1 < 100	col1 >= 100 AND col1 < 1000	col1 >= 1000

Scheme

- CREATE PARTITION SCHEME partition_scheme_name
  AS PARTITION partition_function_name
  [ALL] TO ({ file_group_name | [PRIMARY] } [,...n])
  [;]

Create Table

- CREATE TABLE
  [database_name . [schema_name] . | schema_name .] table_name
  ({ <column_definition> | <computed_column_definition> }
  [<table_constraint>] [,...n])
  [ON { **partition_scheme_name (partition_column_name)** |
  filegroup
  | "default" }]
  [{ TEXTIMAGE_ON { filegroup | "default" }] [;]

Demo

How to partition a table

Partition and Parallel Execution

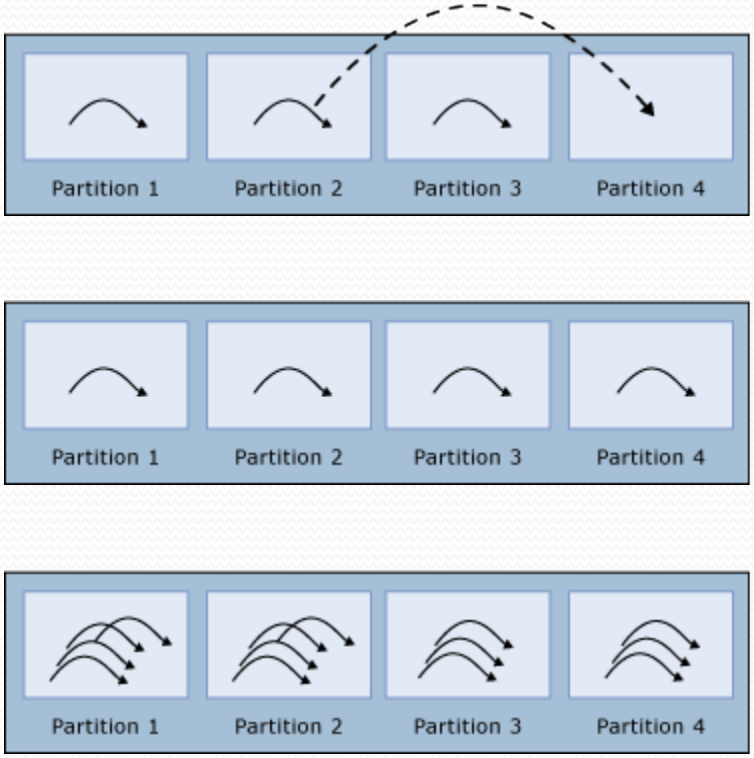
- MAXDOP

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE WITH OVERRIDE;
GO
sp_configure 'max degree of parallelism', 8;
GO
RECONFIGURE WITH OVERRIDE;
GO
```

- 'max worker threads'

Number of CPUs	32-bit computer	64-bit computer
<= 4 processors	256	512
8 processors	288	576
16 processors	352	704
32 processors	480	960

Parallel execution strategy



Query Performance

- It is a key benefit of table partitioning.
- Changes the way parallel and serial plans are represented.
- Enhances both compilation and run-time execution plans.

Query Performance

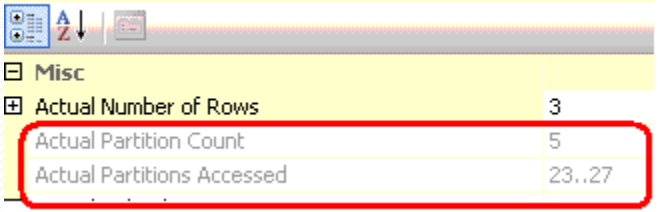
- Partitioned table allows partition aware seek operation.
- This is a key benefit of partitioning often called partition elimination or partition pruning.

```
SELECT *
FROM FactInternetSales
WHERE OrderDateKey BETWEEN 20030402 AND 20030822
AND SalesOrderNumber = 'SO51922'
```

Seek Predicates

Seek Keys[1]: Start: PtnId1000 >= Scalar Operator ((23)), End: PtnId1000 <= Scalar Operator((27)), Seek Keys[2]: Prefix: [AdventureWorksDW2008].[dbo].[FactInternetSales].SalesOrderNumber = Scalar Operator(N'SO51922')

Index Seek (NonClustered)



Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Number of Rows	3
Estimated I/O Cost	0.015625
Estimated CPU Cost	0.0007878
Estimated Operator Cost	0.0164128 (72%)
Estimated Subtree Cost	0.0164128
Estimated Number of Rows	2.52273
Estimated Row Size	38 B
Actual Rebinds	0
Actual Rewinds	0
Partitioned	True
Actual Partition Count	5
Ordered	True
Node ID	1

Query Performance

- Execution plan includes “skip scan: seek keys”
- This is a second level of seek operation thataccessess the combination of conditions that can be satisfied by the index, in this case SalesOrderNumber = 'SO51922'

```
SELECT *
FROM FactInternetSales
WHERE OrderDateKey BETWEEN 20030402 AND 20030822
AND SalesOrderNumber = 'SO51922'
```

Seek Predicates

Seek Keys[1]: Start: PtnId1000 >= Scalar Operator ((23)), End: PtnId1000 <= Scalar Operator((27)),
Seek Keys[2]: Prefix: [AdventureWorksDW2008].[dbo].[FactInternetSales].SalesOrderNumber = Scalar Operator(N'SO51922')

Index Seek (NonClustered)

Misc	
Actual Number of Rows	3
Actual Partition Count	5
Actual Partitions Accessed	23..27

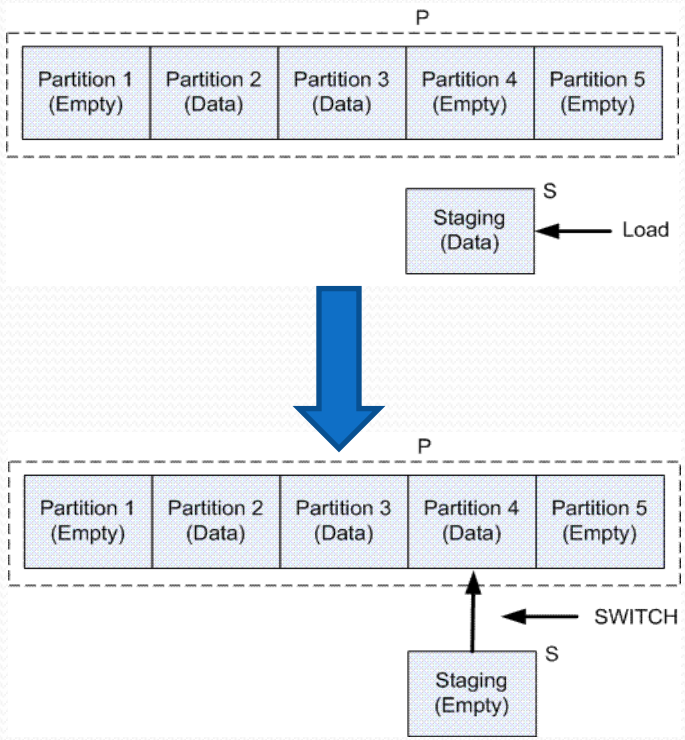
Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

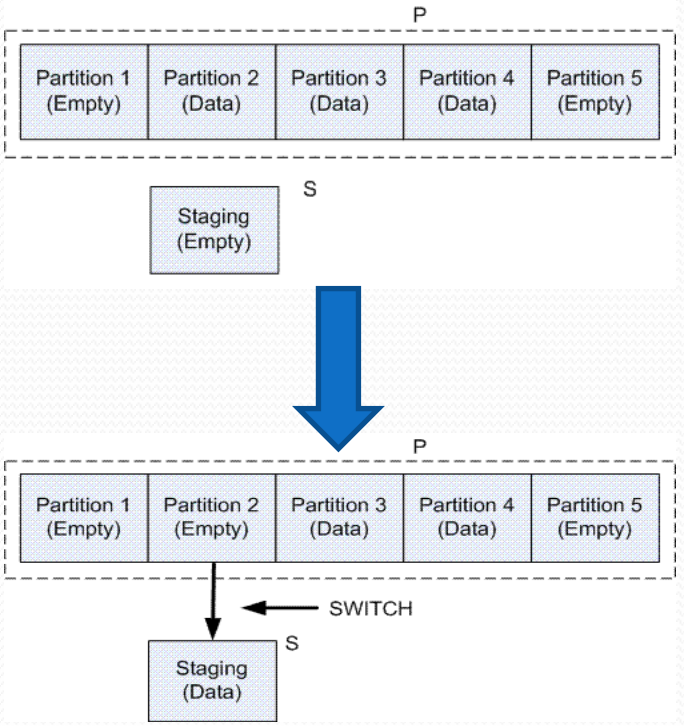
Physical Operation	Index Seek
Logical Operation	Index Seek
Actual Number of Rows	3
Estimated I/O Cost	0.015625
Estimated CPU Cost	0.0007878
Estimated Operator Cost	0.0164128 (72%)
Estimated Subtree Cost	0.0164128
Estimated Number of Rows	2.52273
Estimated Row Size	38 B
Actual Rebinds	0
Actual Rewinds	0
Partitioned	True
Actual Partition Count	5
Ordered	True
Node ID	1

Sliding Window Scenario

Load new data



Remove old data



Demo

Sliding window scenario

Best Practice

- Make sure that the configuration of **max degree of parallelism** is set sufficiently high to take advantage of parallel operations, or else add a MAXDOP query hint to fine-tune the degree of parallelism.
- Maintain an empty partition on both ends of the partitioned table and ensure that only empty partitions are split and merged in a sliding window scenario.
- Remember that RANGE RIGHT may be more convenient than RANGE LEFT in partition functions, especially when you are specifying date ranges.
- Use data types without fractional components as partition columns, such as a date or an integer.
- Always use a standard language-independent date format when specifying partition function boundary values.
- Use an integer-based date and date dimension in data warehouses.
- Use a single column of the table as the partitioned column whenever possible. If you must partition across more than one column, you can use a persisted computed column as the partitioning column. But then to achieve partition elimination, you must control queries to ensure they reference the partition column in their filters.
- Ensure that queries against the partitioned tables have filters based on the partition column.
- Use SWITCH with MERGE to drop partition data: Switch out the partition and remove the partition's boundary value using MERGE.
- Use TRUNCATE TABLE to delete partition data by switching a partition out to a staging table and truncating the staging table.
- Check for partition elimination in query plans.
- Place read only data on read-only filegroups to reduce locking and simplify recovery for piecemeal restores.
- Spread filegroups across all disks for maximum I/O performance.
- Automate sliding window scenarios using available tools.
- When possible, use a server with enough main memory to fit frequently accessed partitions or all partitions in memory to reduce I/O cost.
- If the data you query will not fit in memory, compress the tables and indexes. This will reduce I/O cost.
- Use a server with fast processors and as many processor cores as you can afford, to take advantage of parallel query processing capability.
- Ensure the server has sufficient I/O controller bandwidth.
- Create a clustered index on every large partitioned table to take advantage of B-tree scanning optimizations.

Thank you

http://www.sqlpassnepal.org

ug@sqlpassnepal.org