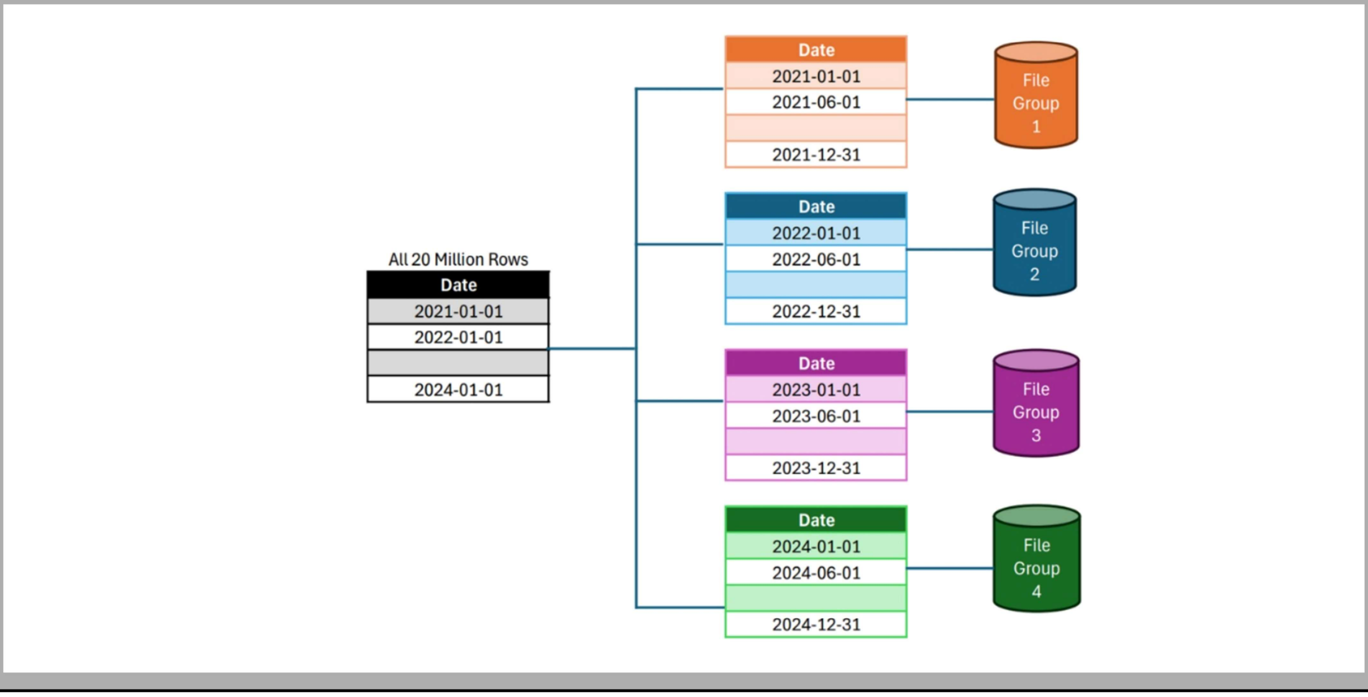
Microsoft

SQL Server Table Partitioning

CS340 | SBASSE | LUMS

Table Partitioning



https://blogs.magnusminds.net/blog/table-partitioning-in-microsoft-sql-server

Table partitioning is a powerful technique that allows large tables to be divided into smaller, more manageable pieces, improving query performance and simplifying maintenance tasks. In this blog, we'll explore the concept of table partitioning, its benefits, and a step-by-step guide to implementing it in SQL Server.

Table partitioning is a powerful technique that allows large tables to be divided into smaller, more manageable pieces, improving query performance and simplifying maintenance tasks. In this blog, we'll explore the concept of table partitioning, its benefits, and a step-by-step guide to implementing it in SQL Server.

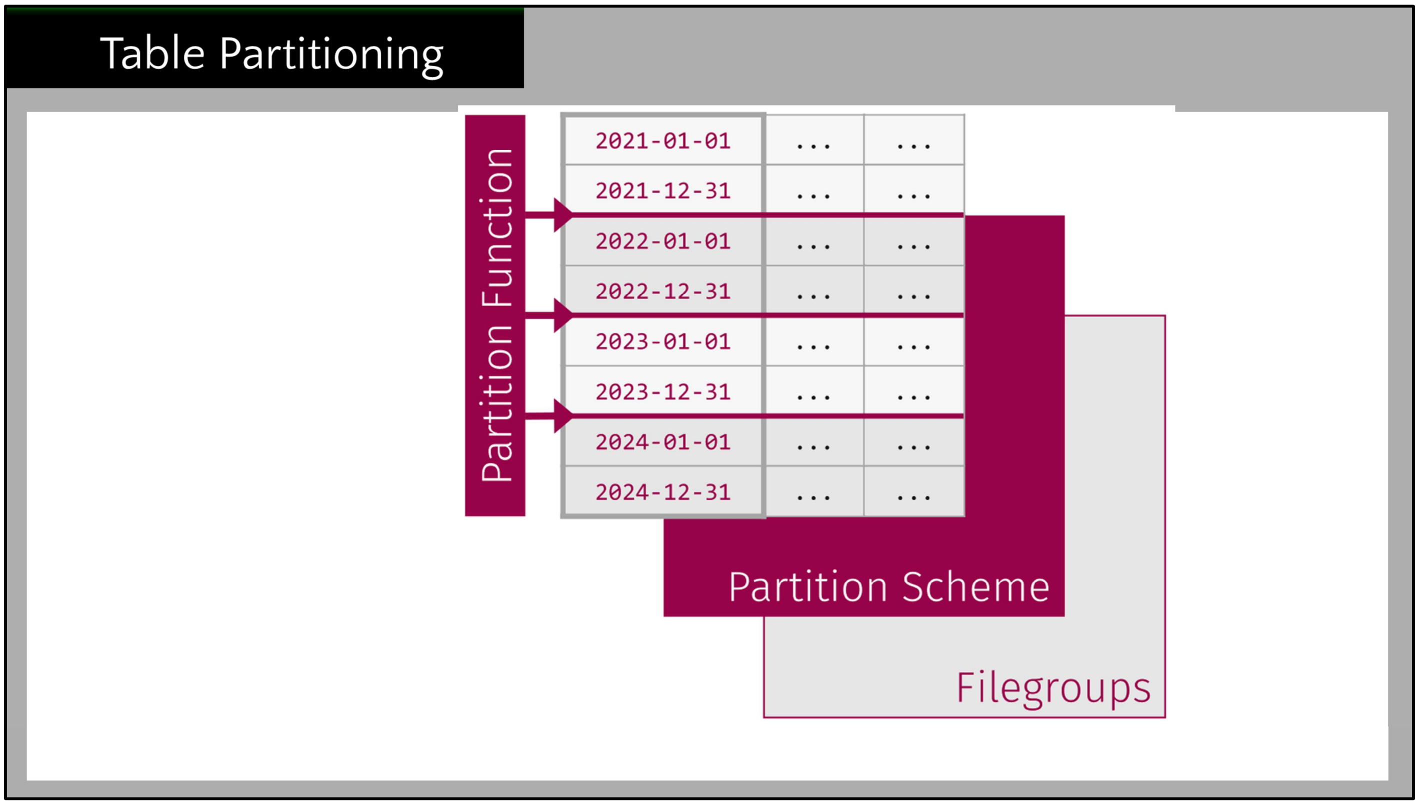
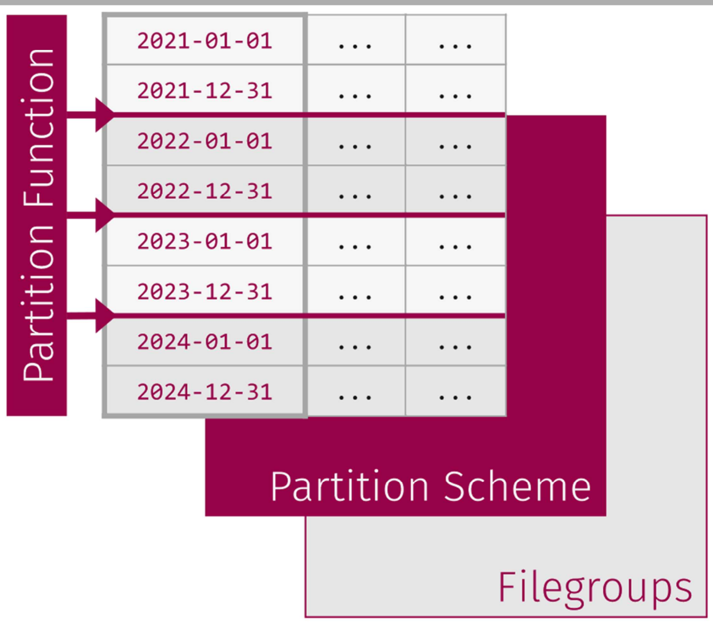


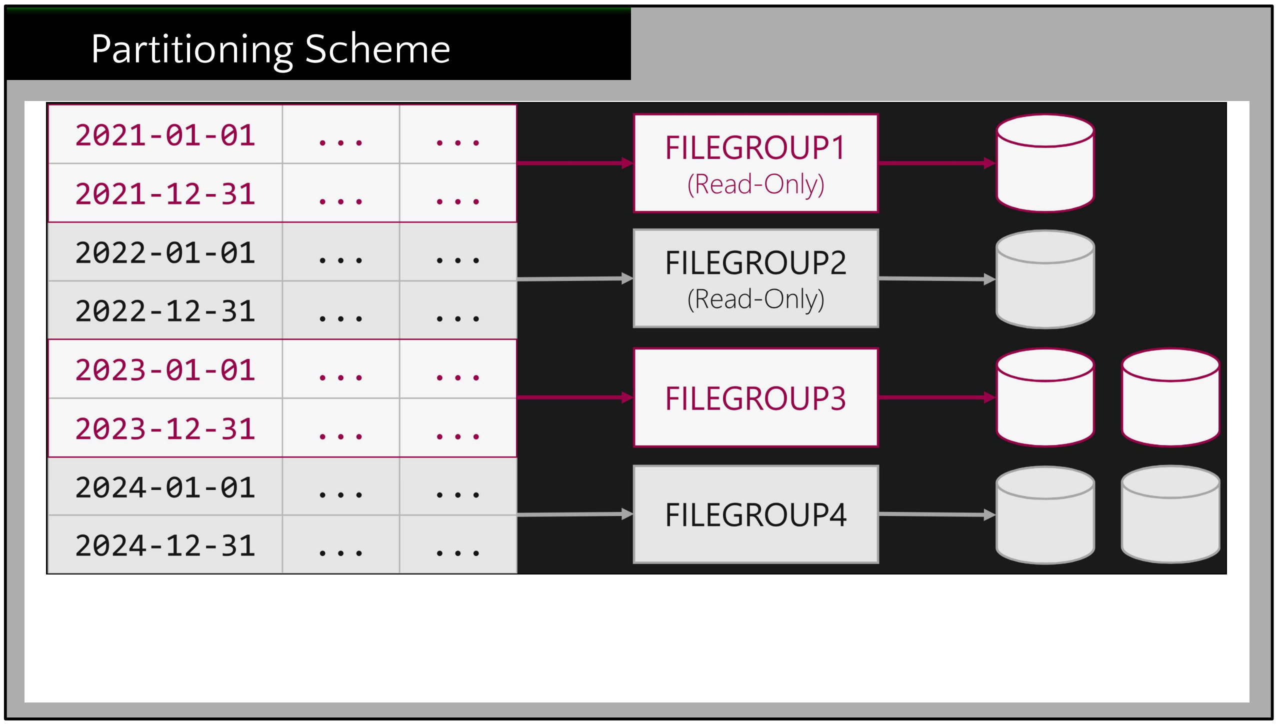
Table Partitioning



https://cabinet.matttroy.net/table-partitioning-in-sql-server-2017-step-by/

Key Concepts

Partition Function: Defines how data is distributed across partitions based on a specified column or columns.
Partition Scheme: Maps the partitions defined by the partition function to specific filegroups within the database.
Aligned Indexes: Indexes that are partitioned in the same way as the table, ensuring that queries using these indexes benefit from partitioning.



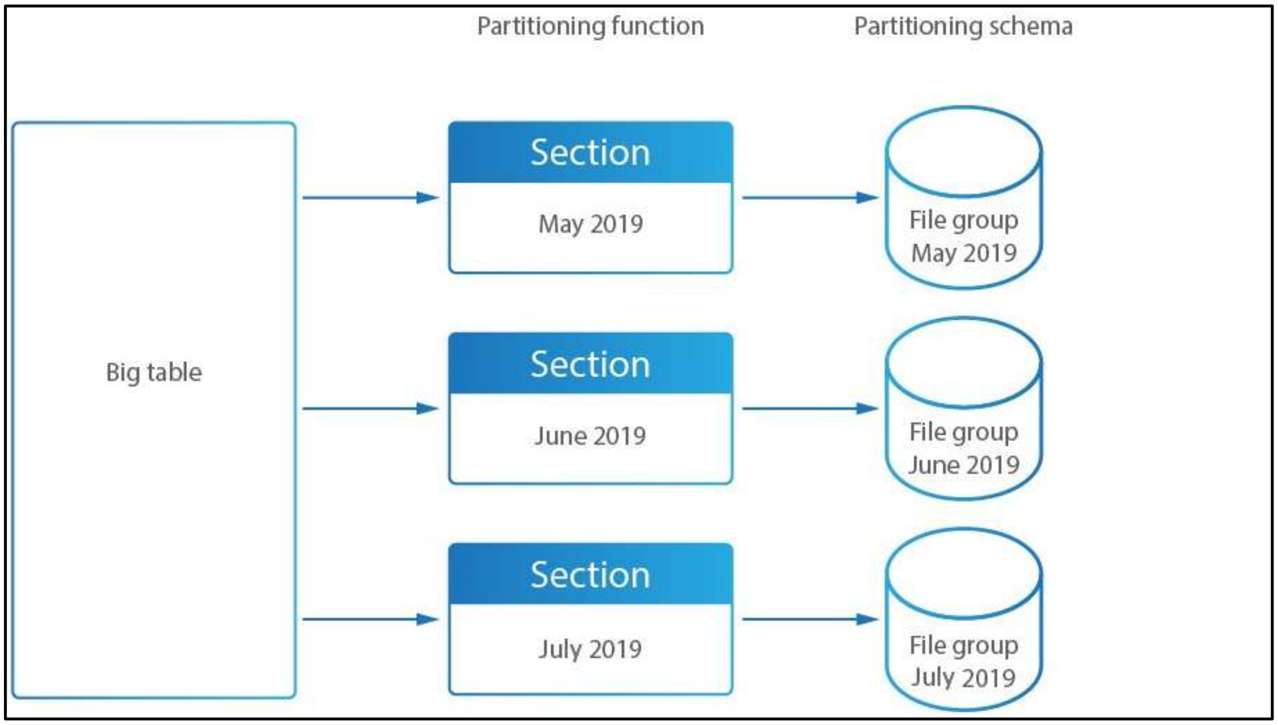
https://www.cathrinewilhelmsen.net/table-partitioning-in-sql-server/

The partition scheme maps the logical partitions to physical filegroups. It is possible to map each partition to its own filegroup or all partitions to one filegroup.

A filegroup contains one or more data files that can be spread on one or more disks. Filegroups can be set to read-only, and filegroups can be backed up and restored individually.

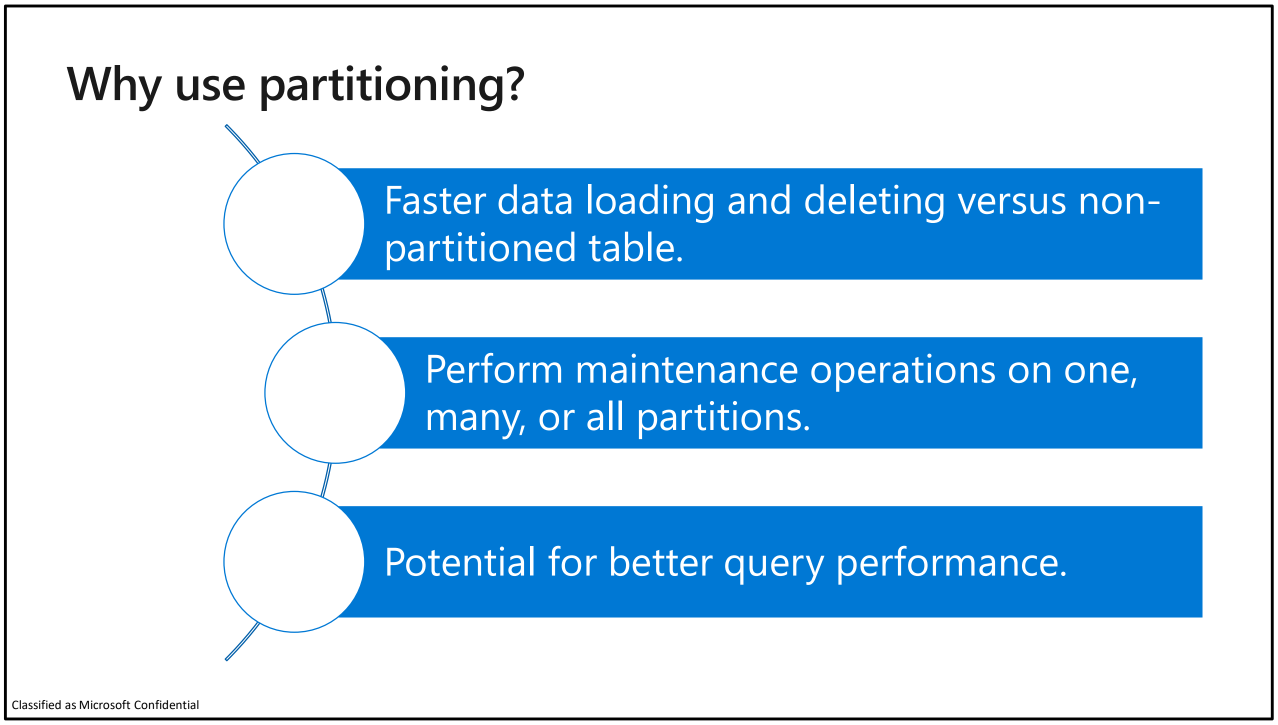
There are many benefits of mapping each partition to its own filegroup. Less frequently accessed data can be placed on slower disks and more frequently accessed data can be placed on faster disks.

Historical, unchanging data can be set to read-only and then be excluded from regular backups. If data needs to be restored it is possible to restore the partitions with the most critical data first.


Partitioning function
Partitioning schema
Big table
Section
May 2019
Section
June 2019
Section
July 2019
File group May 2019
File group June 2019
File group July 2019

What is Table Partitioning

- Technique in SQL Server that allows you to split a large table into smaller, more manageable pieces (partitions) based on the values in a specific column
- Each partition behaves like a separate table internally
- Partitions are invisible to the user: Treated as one entity when queried.



Why use partitioning?

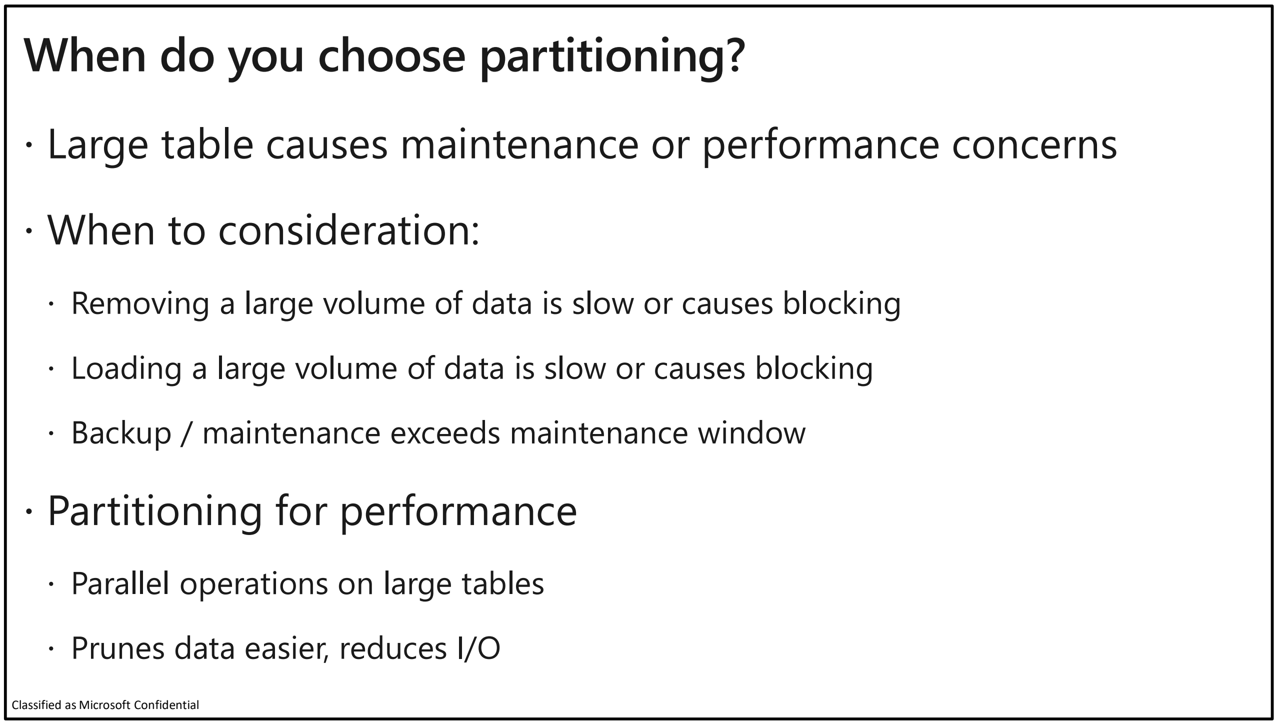
- Faster data loading and deleting versus non-partitioned table.
- Perform maintenance operations on one, many, or all partitions.
- Potential for better query performance.

Classified as Microsoft Confidential

You can transfer or access subsets of data quickly and efficiently, while maintaining the integrity of a data collection. For example, an operation such as loading data from an OLTP to an OLAP system takes only seconds, instead of the minutes and hours the operation takes when the data isn't partitioned.

You can perform maintenance or data retention operations on one or more partitions more quickly. The operations are more efficient because they target only these data subsets, instead of the whole table. For example, you can choose to compress data in one or more partitions, rebuild one or more partitions of an index, or truncate data in a single partition. You may also switch individual partitions out of one table and into an archive table.

You may improve query performance, based on the types of queries you frequently run. For example, the query optimizer can process equijoin queries between two or more partitioned tables faster when the partitioning columns are the same as the columns on which the tables are joined.



When do you choose partitioning?

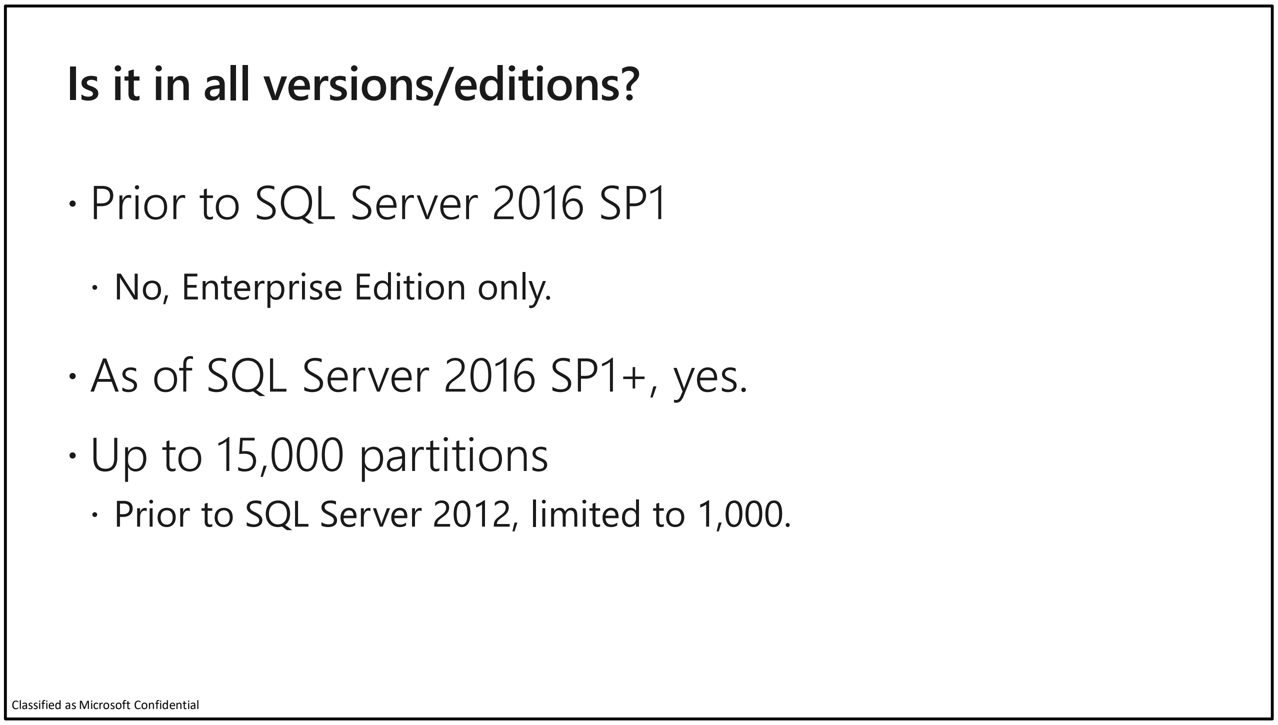
- Large table causes maintenance or performance concerns
- When to consideration:
  - Removing a large volume of data is slow or causes blocking
  - Loading a large volume of data is slow or causes blocking
  - Backup / maintenance exceeds maintenance window
- Partitioning for performance
  - Parallel operations on large tables
  - Prunes data easier, reduces I/O

There is no firm rule or formula that would determine when a table is large enough to be partitioned, or whether even a very large table would benefit from partitioning.

In general, any large table that has maintenance costs that exceed requirements, or that is not performing as expected due to its size, might be a candidate for table partitioning.

Partitioning may help when:
- Data must be removed from the table periodically, but deleting is slow, or blocks other operations.
- Data is loaded into the table periodically, but the loading is slow, or blocks other operations.
- Backup and maintenance, such as, index or statistics maintenance, of the entire table is exceeding a maintenance window, and doing maintenance on only a portion of the table would help.

High performance is often a major reason many choose to partition. If queries have a filtering key that corresponds to the partitioning key, response time will be faster compared to the same query against a monolithic table. This is because the use of partitioning encourages the use of parallel operations and the partitioning key in the query predicate makes pruning data easier.



Is it in all versions/editions?

- Prior to SQL Server 2016 SP1
  - No, Enterprise Edition only.
- As of SQL Server 2016 SP1+, yes.
- Up to 15,000 partitions
  - Prior to SQL Server 2012, limited to 1,000.

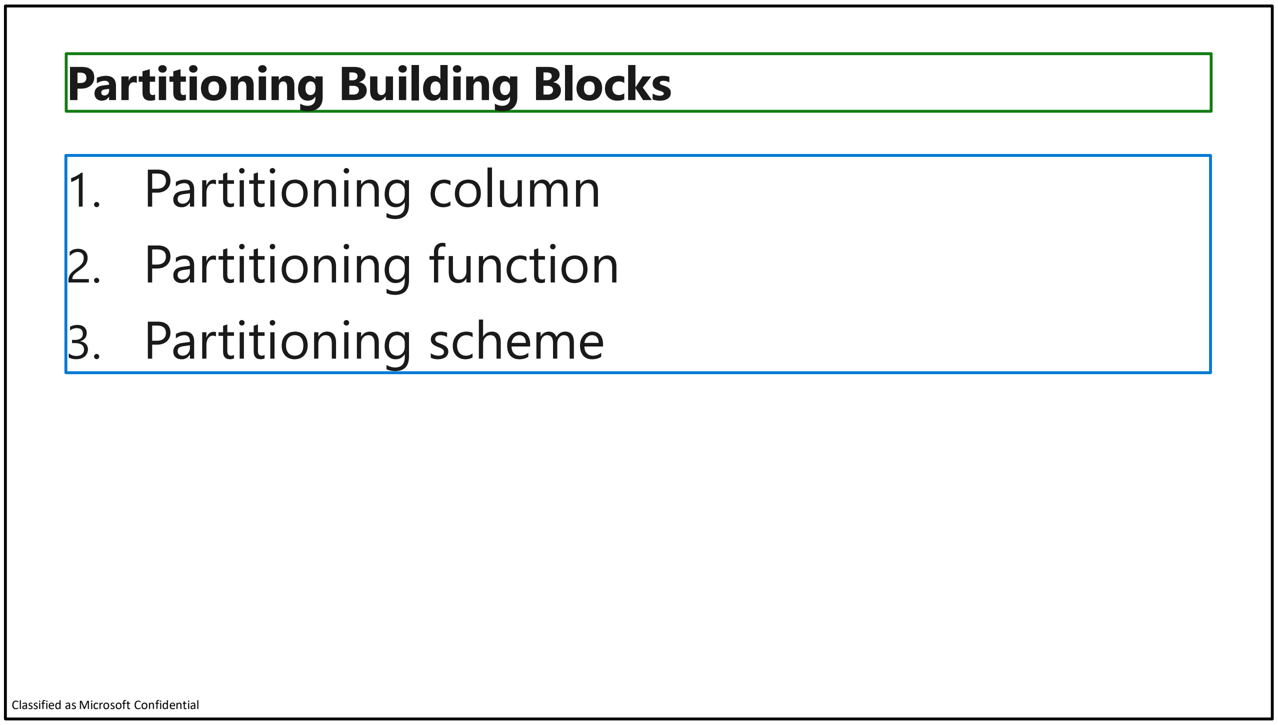
Prior to SQL Server 2016 (13.x) SP1, partitioned tables and indexes weren't available in every edition of SQL Server. As of SQL Server 2016 SP1 partitioning is available for all editions.

Partitioned tables and indexes are available in all service tiers of Azure SQL Database and Azure SQL Managed Instance.

The database engine supports up to 15,000 partitions by default. In versions earlier than SQL Server 2012 (11.x), the number of partitions was limited to 1,000 by default.

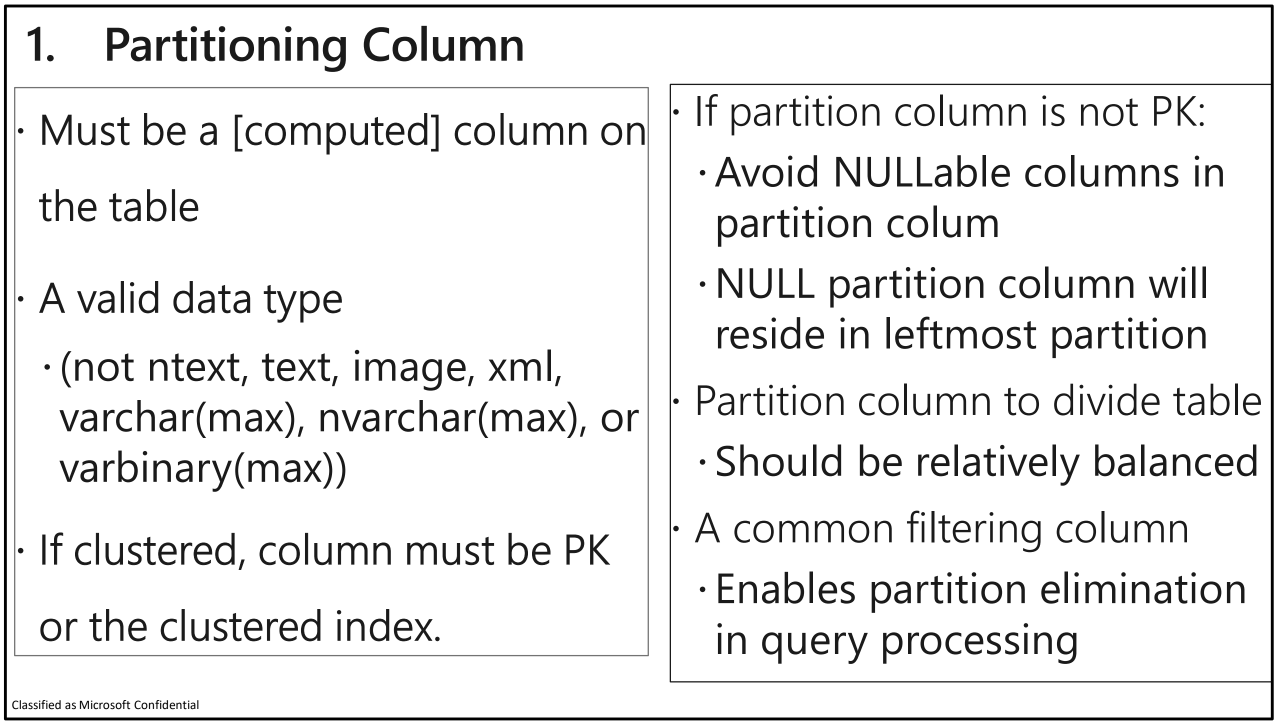
Microsoft

Table partitioning basics



Partitioning Building Blocks

1. Partitioning column
2. Partitioning function
3. Partitioning scheme



1. Partitioning Column

- Must be a [computed] column on the table
- A valid data type
  - (not ntext, text, image, xml, varchar(max), nvarchar(max), or varbinary(max))
- If clustered, column must be PK or the clustered index.

- If partition column is not PK:
  - Avoid NULLable columns in partition colum
  - NULL partition column will reside in leftmost partition
- Partition column to divide table
  - Should be relatively balanced
- A common filtering column
  - Enables partition elimination in query processing

Classified as Microsoft Confidential

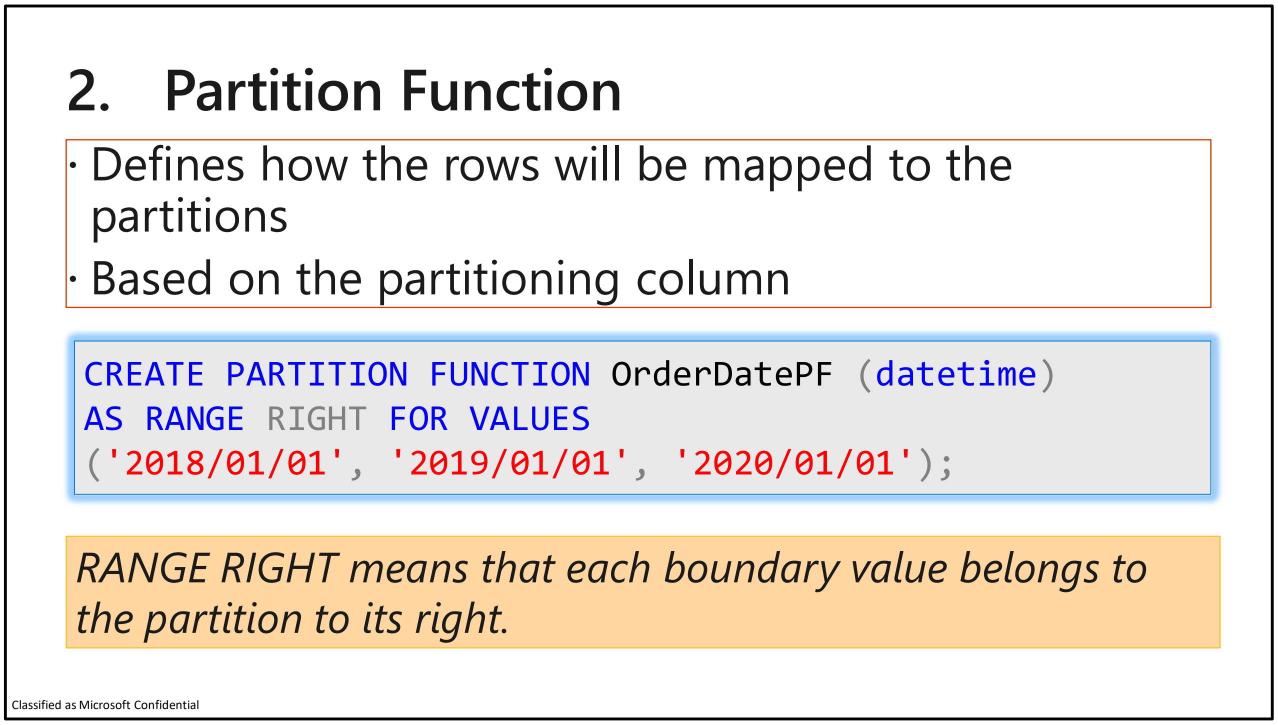
After you've decided a table could benefit from being partitioned, you need to determine the partition column. The partition column stores the values upon which you want to partition the table. The partition column choice is critical, because after you have partitioned a table, choosing a different partition column will require re-creating the table, reloading all the data, and rebuilding all the indexes.

Some things to note about the partition column:

- The partition column must be a single column in the table (either a single column or a computed column).
  - If you have a combination of columns that form the best partition column, you can add a persisted computed column to the table that combines the values of the original columns and then partition on it.
- The partition column must also have an allowable data type. All data types that are valid for use as index columns can be used as a partitioning column, except

timestamp. The ntext, text, image, xml, varchar(max), nvarchar(max), or varbinary(max) data types cannot be specified. Also, Microsoft .NET Framework common language runtime (CLR) user-defined type and alias data type columns cannot be specified.

- In a clustered table, the partition column must be part of either the primary key or the clustered index. If the partition column is not part of the table's primary key, the partition column might allow NULL. Any data with a NULL in the partition column will reside in the leftmost partition.
- The partitioning column should reflect the best way to subdivide the target table. You should look for a relatively balanced distribution of data across the resulting partitions, though it may not be possible to know that in advance.
- You should also try to choose a partitioned column that will be used as a filter criterion in most of the queries run against the table. This enables partition elimination, where the query processor can eliminate inapplicable partitions from the query plan, and just access the partitions implied by the filter on the queries.


2. Partition Function

- Defines how the rows will be mapped to the partitions
- Based on the partitioning column

```
CREATE PARTITION FUNCTION OrderDatePF (datetime)
AS RANGE RIGHT FOR VALUES
('2018/01/01', '2019/01/01', '2020/01/01');
```

RANGE RIGHT means that each boundary value belongs to the partition to its right.

Classified as Microsoft Confidential

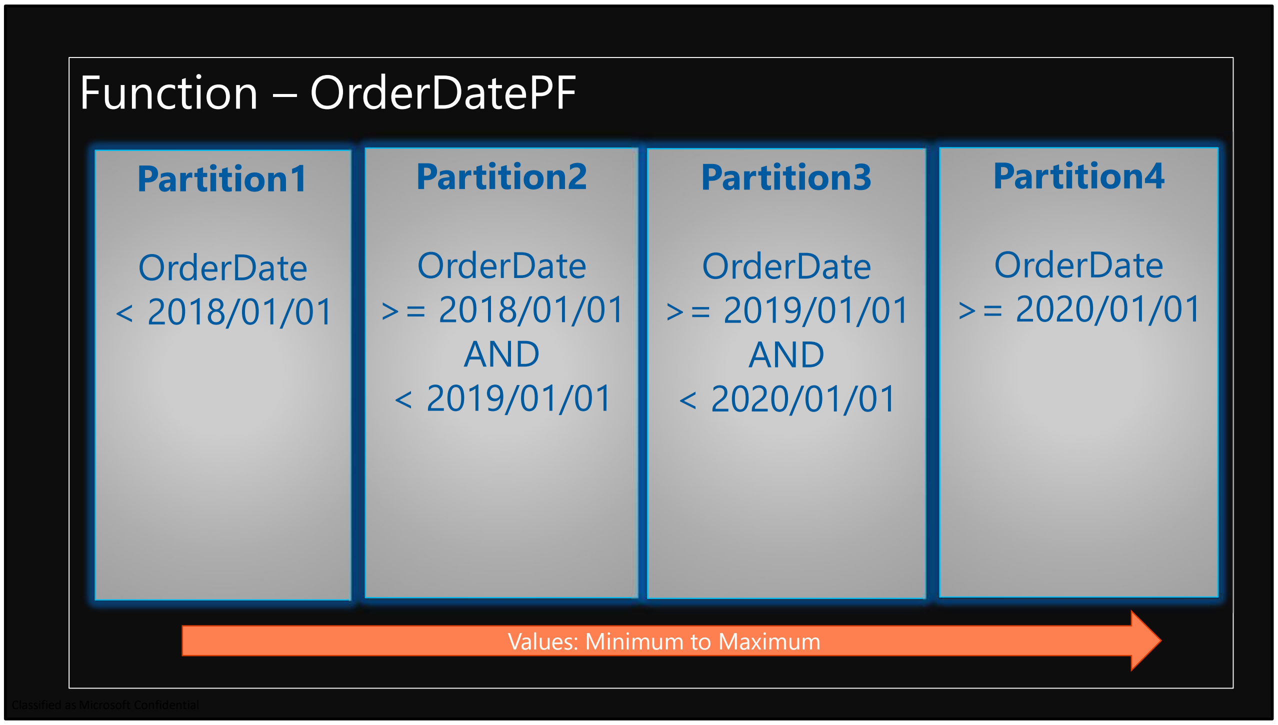
A partition function is a database object that defines how the rows of a table or index are mapped to a set of partitions based on the values of a certain column, called a partitioning column. Each value in the partitioning column is an input to the partitioning function, which returns a partition value.

We will discuss range types later, but briefly a range type (either LEFT or RIGHT), specifies how the boundary values of the partition function will be put into the resulting partitions:

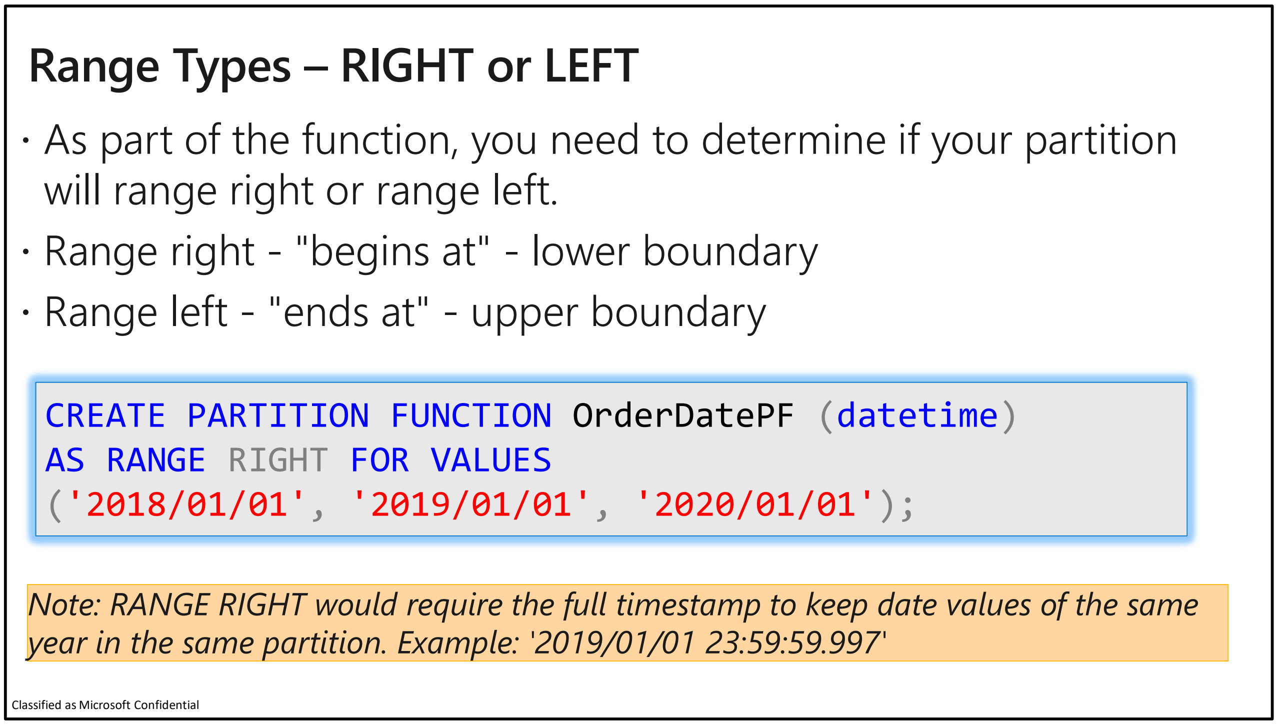
- A LEFT range specifies that the range extends to the left of the boundary value
- A RIGHT range specifies that the range extends to the right of the boundary value

Ref: https://www.cathrinewilhelmsen.net/table-partitioning-in-sql-server/

Range left means that the boundary value belongs to its left partition, it is the *last value in the left partition*.
Range right means that the boundary value belongs to its right partition, it is the *first value in the right partition*.



Function – OrderDatePF



Range Types – RIGHT or LEFT

- As part of the function, you need to determine if your partition will range right or range left.
- Range right - "begins at" - lower boundary
- Range left - "ends at" - upper boundary

```
CREATE PARTITION FUNCTION OrderDatePF (datetime)
AS RANGE RIGHT FOR VALUES
('2018/01/01', '2019/01/01', '2020/01/01');
```

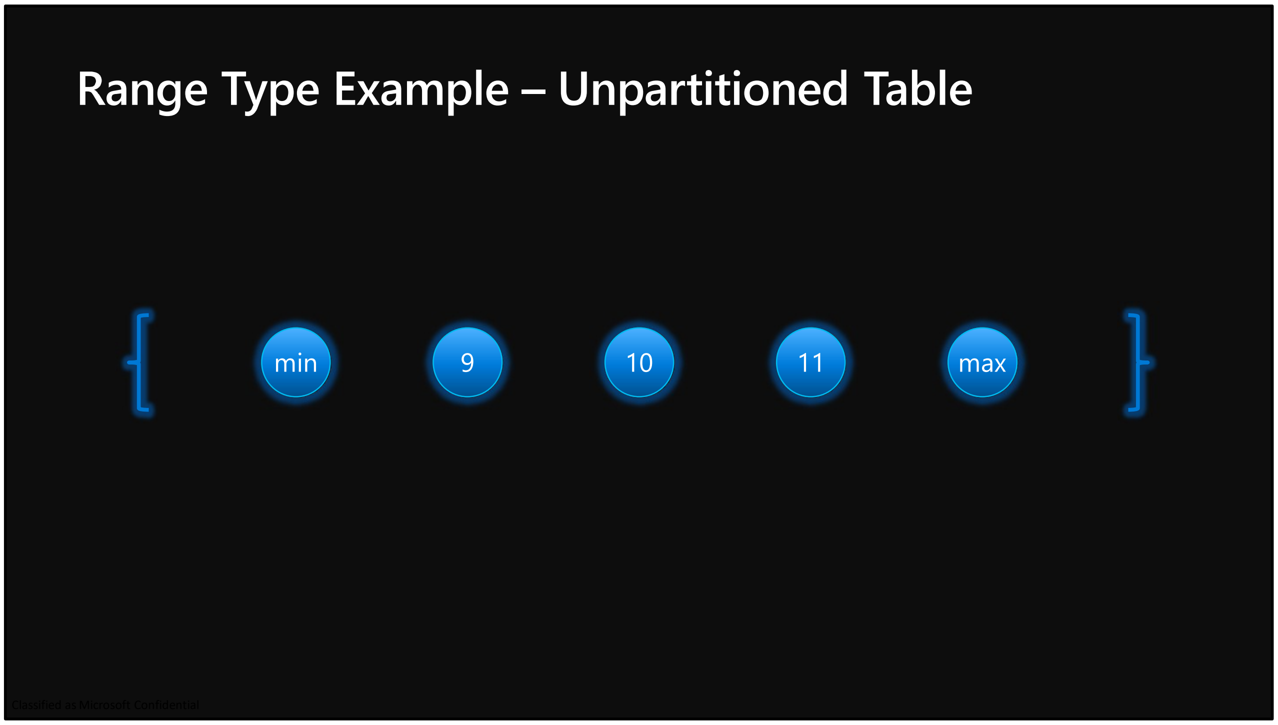
Note: RANGE RIGHT would require the full timestamp to keep date values of the same year in the same partition. Example: '2019/01/01 23:59:59.997'

A range type (either LEFT [***default if not specified***] or RIGHT), specifies how the boundary values of the partition function will be put into the resulting partitions:

- A LEFT range specifies that the boundary value belongs to the left side of the boundary value interval when interval values are sorted by the database engine in ascending order from left to right. In other words, the highest bounding value will be included within a partition.
- A RIGHT range specifies that the boundary value belongs to the right side of the boundary value interval when interval values are sorted by the database engine in ascending order from left to right. In other words, the lowest bounding value will be included in each partition.

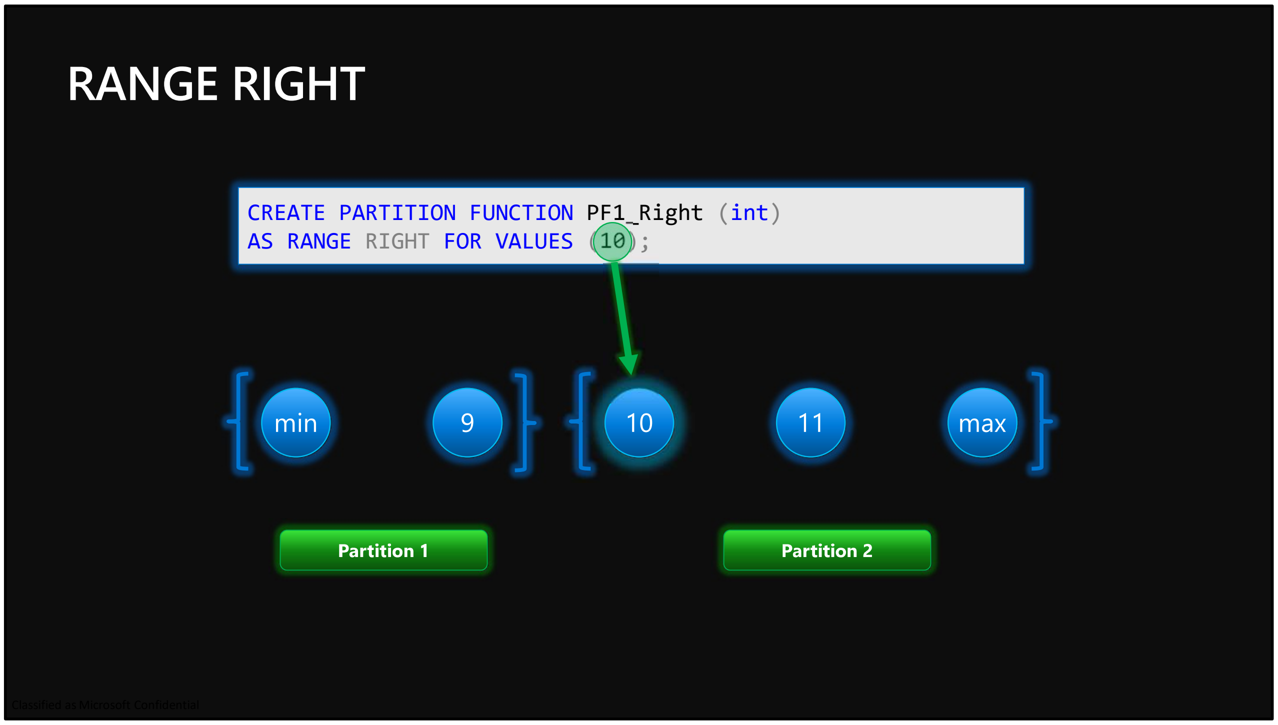
In this example, the partition function partitions a table with 3 partitions boundry, one for each year of 3 years' worth of data in a datetime column. A RIGHT range is used, indicating that boundary values will serve as lower bounding values in each partition. RIGHT ranges are often simpler to work with when partitioning a table based on a column of date, datetime or datetime2 data types, as rows with a value of midnight will be stored in the same partition as rows with later values on the same day. This aids in precise Partition elimination when querying an entire day's worth of data.

If you where to use a RANGE RIGHT here you would need the full time stamps that includes hour, minute, seconds, and milliseconds '2019/01/01 23:59:59.997'



Range Type Example – Unpartitioned Table

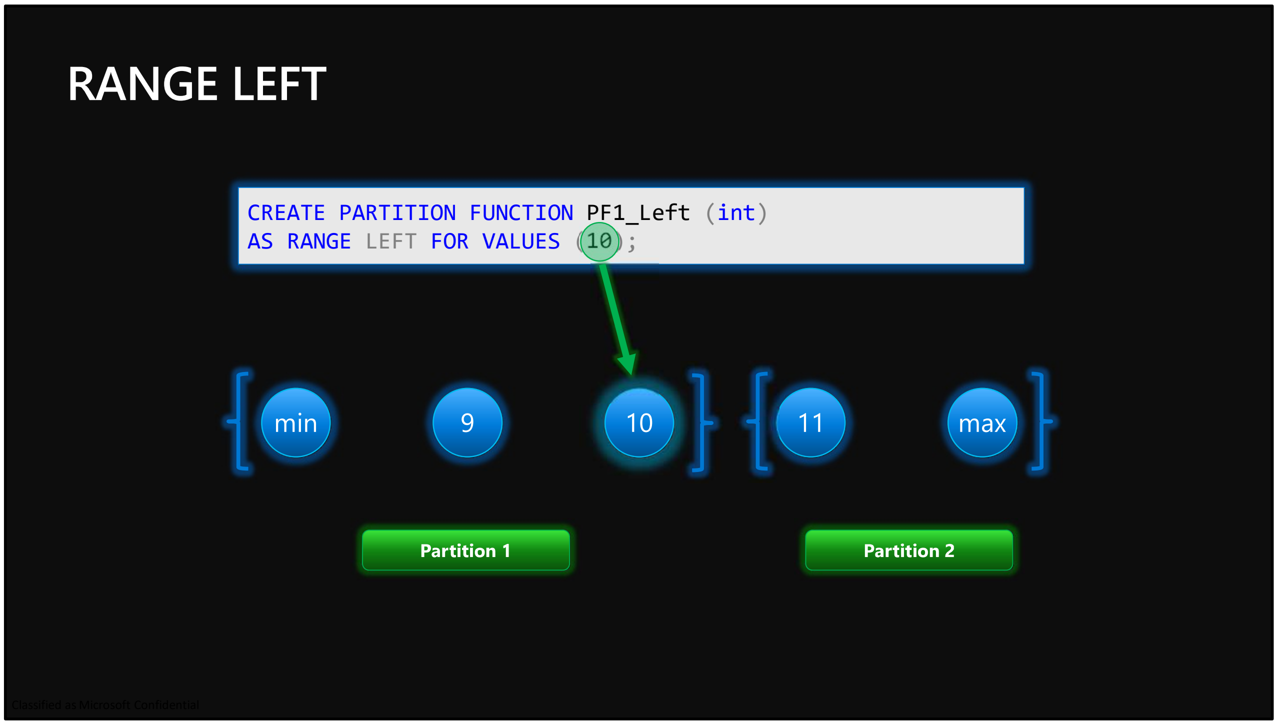
Here's another range type example. Let's take a look at an unpartitioned table that has these values.



RANGE RIGHT

```
CREATE PARTITION FUNCTION PF1_Right (int)
AS RANGE RIGHT FOR VALUES (10);
```

For a range right, we can see in this example, the values extend to the right of the value of 10.



RANGE LEFT

```
CREATE PARTITION FUNCTION PF1_Left (int)
AS RANGE LEFT FOR VALUES (10);
```

For a range left, we can see in this example, the values extend to the left of the value of 10.

Microsoft

Paritioning Scheme

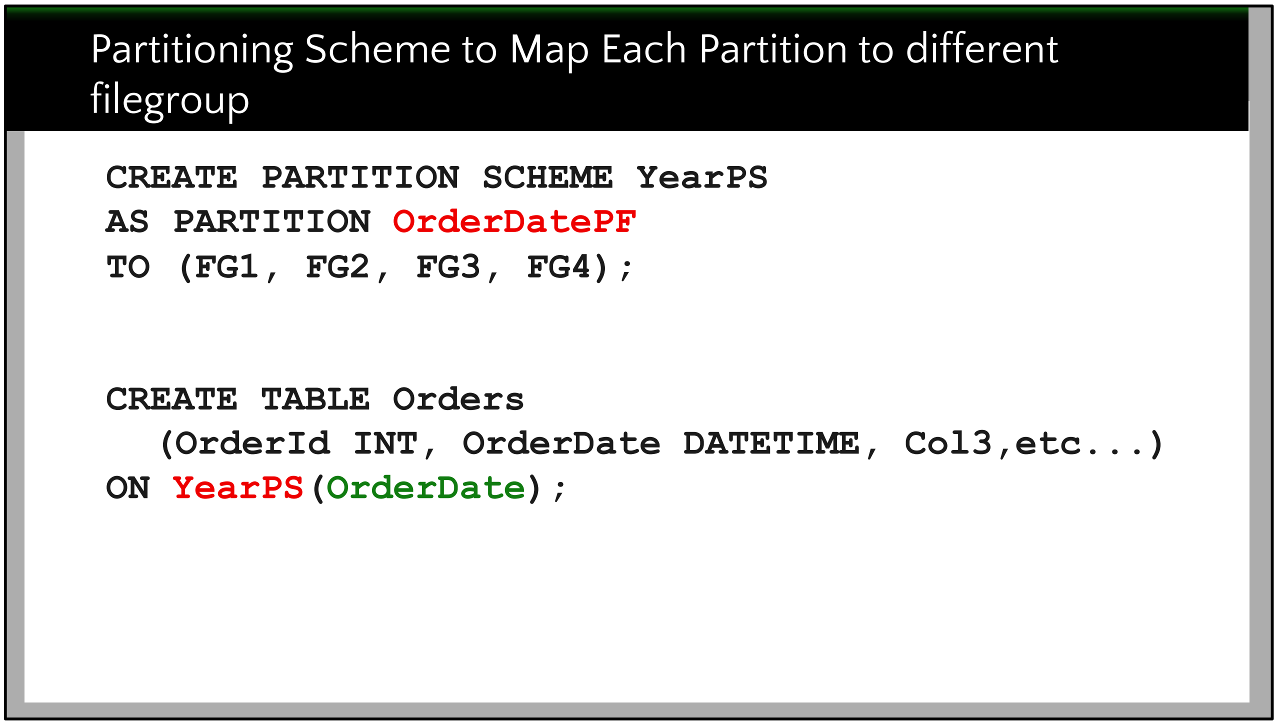
Partitioning Scheme

A **partitioning scheme** defines **how data is distributed across partitions** based on a **partition function**.

- The partition function, as defined earlier, maps rows to partitions based on the values in a specified column (often a date or ID).
- Then partitioning scheme maps those partitions to filegroups.

Benefits of Parititioning:

- **Improved Query Performance**: Queries can target specific partitions.
- **Efficient Maintenance**: You can switch, truncate, or rebuild partitions individually.
- **Better Storage Management**: Distribute data across multiple filegroups/disks.



Partitioning Scheme to Map Each Partition to different filegroup

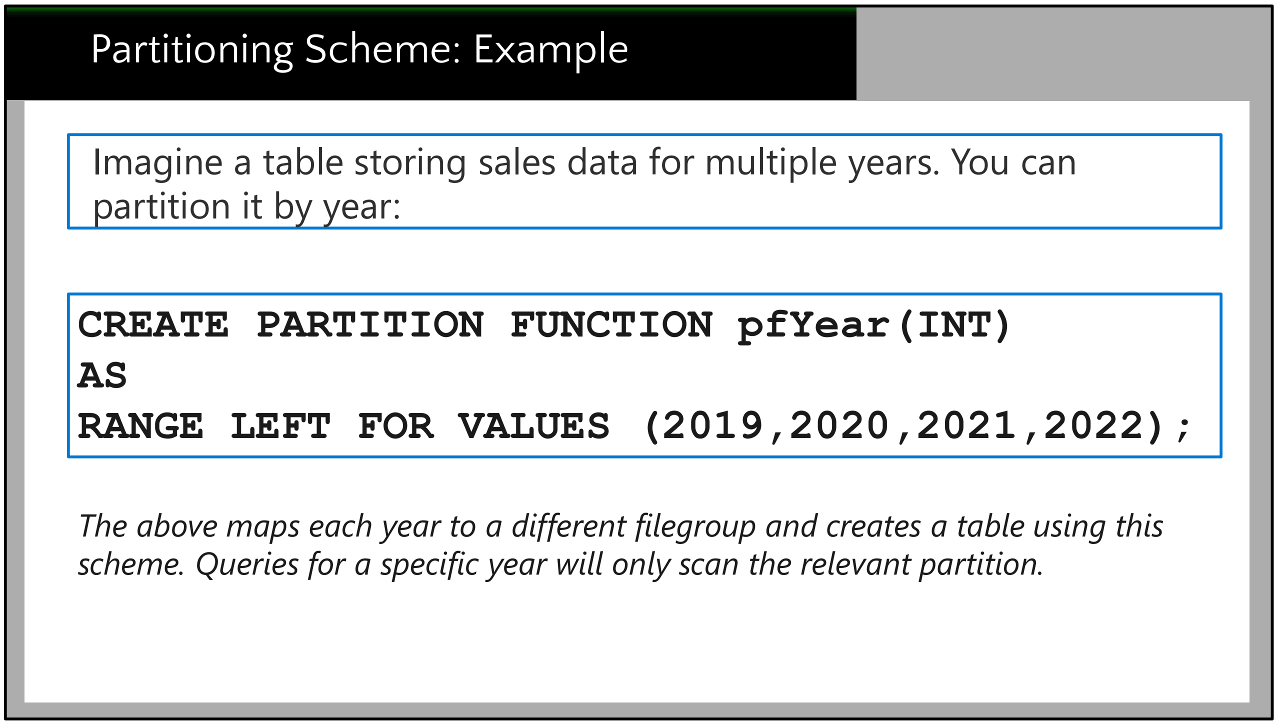
```
CREATE PARTITION SCHEME YearPS
AS PARTITION OrderDatePF
TO (FG1, FG2, FG3, FG4);


CREATE TABLE Orders
   (OrderId INT, OrderDate DATETIME, Col3,etc...)
ON YearPS(OrderDate);
```

A partition scheme maps the partitions of a partitioned table or index to one or more filegroups.

The values that map the rows of a table or index into partitions are specified in the partition function. A partition function and file groups must be first created before creating a partition scheme.

In this example, we are creating a partition scheme called YearPS, using the OrderDatePF partition function. The scheme maps values using the partition function to file groups, FG1, FG2, FG3, and FG4.

With the function and scheme in place we can now create the table. In this example we're creating a table called orders. And instead of placing the table on a filegroup, we specify the partition scheme name, called YearsPS with the column name OrderDate to provide the value. The value of the order date is passed to the partition function to determine which partition the OrderDate is mapped to.

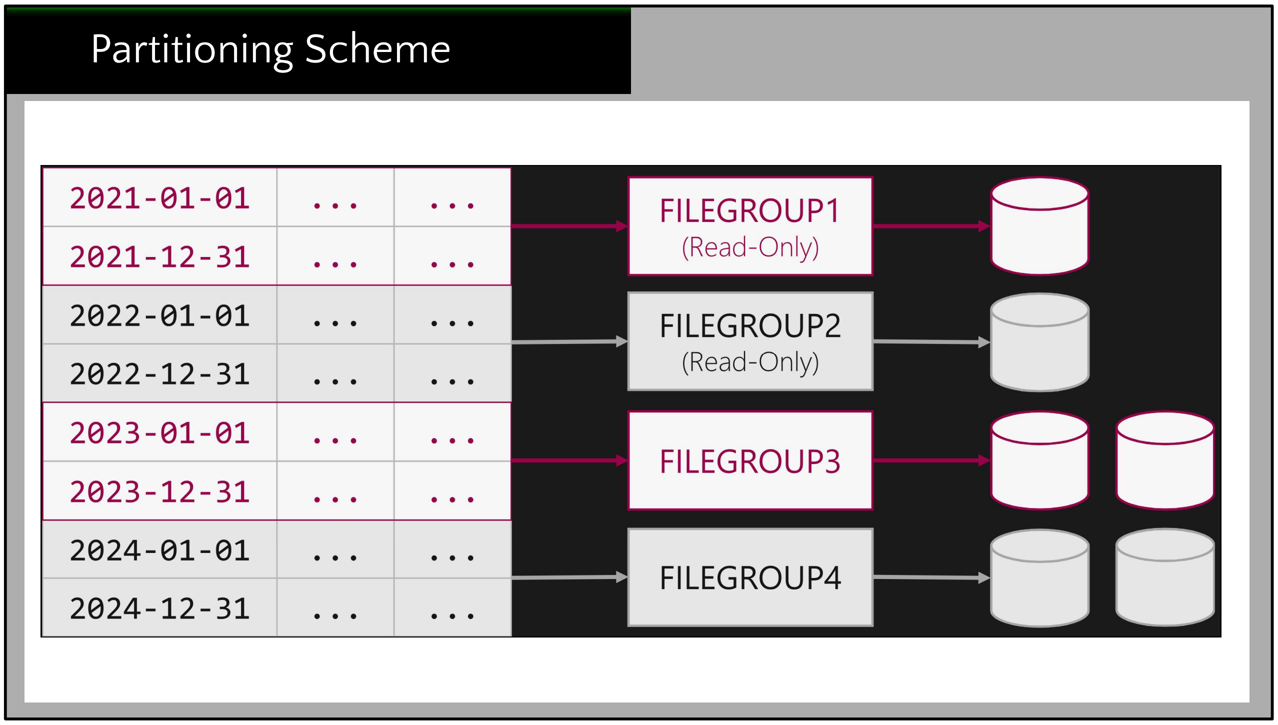


Partitioning Scheme: Example

Imagine a table storing sales data for multiple years. You can partition it by year:

```
CREATE PARTITION FUNCTION pfYear(INT)
AS
RANGE LEFT FOR VALUES (2019,2020,2021,2022);
```

The above maps each year to a different filegroup and creates a table using this scheme. Queries for a specific year will only scan the relevant partition.



https://www.cathrinewilhelmsen.net/table-partitioning-in-sql-server/

The partition scheme maps the logical partitions to physical filegroups. It is possible to map each partition to its own filegroup or all partitions to one filegroup.
A filegroup contains one or more data files that can be spread on one or more disks.
Filegroups can be set to read-only, and filegroups can be backed up and restored individually.

There are many benefits of mapping each partition to its own filegroup. Less frequently accessed data can be placed on slower disks and more frequently accessed data can be placed on faster disks. Historical, unchanging data can be set to read-only and then be excluded from regular backups. If data needs to be restored it is possible to restore the partitions with the most critical data first.

Practical Use of Making a Filegroup Read-Only in SQL Server
https://axial-sql.com/info/practical-use-of-making-a-filegroup-read-only-in-sql-server/