# PARTITIONS

SQL Server supports table and index partitioning. The data of partitioned tables and indexes is divided into units that can be spread across more than one filegroup in a database. The data is partitioned horizontally, so that groups of rows are mapped into individual partitions. All partitions of a single index or table must reside in the same database.

**Important**

SQL Server 2017 supports up to 15,000 partitions by default. In versions earlier than SQL Server 2012, the number of partitions was limited to 1,000 by default.On x86-based systems, creating a table or index with more than 1000 partitions is

possible, but is not supported.

### Benefits OF Partitioning:

Partitioning large tables or indexes can have the following manageability and performance benefits.

* 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 is not partitioned.
* You can perform maintenance 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 or rebuild one or more partitions of an index.
* You may improve query performance, based on the types of queries you frequently run and on your hardware configuration. For example, the query optimizer can process equi-join queries between two or more partitioned tables faster when the partitioning columns in the tables are the same, because the partitions themselves can be joined.

When SQL Server performs data sorting for I/O operations, it sorts the data first by partition. SQL Server accesses one drive at a time, and this might reduce performance. To improve data sorting performance, stripe the data files of your partitions across more than one disk by setting up a RAID. In this way, although SQL Server still sorts data by partition, it can access all the drives of each partition at the same time.

In addition, you can improve performance by enabling lock escalation at the partition level instead of a whole table. This can reduce lock contention on the table.

### Components and Concepts

The following terms are applicable to table and index partitioning.

#### Partition function

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 certain column, called a **partitioning column**. That is, the partition function defines the number of partitions that the table will have and how the boundaries of the partitions are defined. **For example**, given a table that contains sales order data, you may want to partition the table into twelve (monthly) partitions based on a **datetime** column such as a sales date.

#### Partition scheme

A database object that maps the partitions of a partition function to a set of filegroups. The primary reason for placing your partitions on separate filegroups is to make sure that you can independently perform backup operations on partitions. This is because you can perform backups on individual filegroups.

#### Partitioning column

The column of a table or index that a partition function uses to partition the table or index. **Timestamp**, **ntext**, **text**, **image**, **xml**, **varchar(max)**, **nvarchar(max)**, or **varbinary(max)** data types, (CLR) user-defined type cannot be specified.

#### Aligned index

An index that is built on the same partition scheme as its corresponding table. When a table and its indexes are in alignment, SQL Server can switch partitions quickly and efficiently while maintaining the partition structure of both the table and its indexes.

#### Nonaligned index

An index partitioned independently from its corresponding table. That is, the index has a different partition scheme or is placed on a separate filegroup from the base table. Designing an nonaligned partitioned index can be useful in the following cases:

* The base table has not been partitioned.
* The index key is unique and it does not contain the partitioning column of the table.
* You want the base table to participate in collocated joins with more tables using different join columns.

Partition elimination The process by which the query optimizer accesses only the relevant partitions to satisfy the filter criteria of the query.

### Performance Guidelines

#### Processor Cores and Number of Partitions Guidelines

To maximize performance with parallel operations, we recommend that you use the same number of partitions as processor cores, up to a maximum of 64 (which is the maximum number of parallel processors that SQL Server can utilize).

#### Memory Usage and Guidelines

We recommend that you use at least 16 GB of RAM if a large number of partitions are in use. If the system does not have enough memory, Data Manipulation Language (DML) statements, Data Definition Language (DDL) statements and other operations can fail due to insufficient memory.

#### Partitioned Index Operations

Creating and rebuilding aligned indexes could take longer to execute as the number of partitions increases. We recommend that you do not run multiple create and rebuild index commands at the same time as you may run into performance and memory issues.

When SQL Server performs sorting to build partitioned indexes, it first builds one sort table for each partition. It then builds the sort tables either in the respective filegroup of each partition or in **tempdb**, if the SORT\_IN\_TEMPDB index option is specified. Each sort table requires a minimum amount of memory to build. When you are building a partitioned index that is aligned with its base table, sort tables are built one at a time, using less memory. However, when you are building a nonaligned partitioned index, the sort tables are built at the same time. As a result, there must be sufficient memory to handle these concurrent sorts. The larger the number of partitions, the more memory required. The minimum size for each sort table, for each partition, is 40 pages, with 8 kilobytes per page.

**Left boundary and Right boundary**

**Create Partition Tables and Indexes**

Creating a partitioned table or index typically happens in four parts:

1. Create a filegroup or filegroups and corresponding files that will hold the partitions specified by the partition scheme.

2. Create a partition function that maps the rows of a table or index into partitions based on the values of a specified column.

3. Create a partition scheme that maps the partitions of a partitioned table or index to the new filegroups.

4. Create or modify a table or index and specify the partition scheme as the storage location

**Create Partition Schema**

**Modify Partition function**