Partitioned Tables and Indexes

SQL Server 2016 and later

Updated: January 20, 2016

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. The table or index is treated as a single logical entity when queries or updates are performed on the data. Prior to SQL Server 2016 SP!, partitioned tables and indexes were not available in every edition of SQL Server. For a list of features that are supported by the editions of SQL Server, see Features Supported by the Editions of SQL Server 2016.

Important

SQL Server 2016 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. Computed columns that participate in a partition function must be explicitly marked PERSISTED. 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.

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. An index does not have to participate in the same named partition function to be aligned with its base table. However, the partition function of the index and the base table must be essentially the same, in that 1) the arguments of the partition functions have the same data type, 2) they define the same number of partitions, and 3) they define the same boundary values for partitions.

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

The new, higher limit of 15,000 partitions affects memory, partitioned index operations, DBCC commands, and queries. This section describes the performance implications of increasing the number of partitions above 1,000 and provides workarounds

as needed. With the limit on the maximum number of partitions being increased to 15,000, you can store data for a longer time. However, you should retain data only for as long as it is needed and maintain a balance between performance and number of partitions.

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. Systems with 16 GB of RAM that run many memory-intensive processes may run out of memory on operations that run on a large number of partitions. Therefore, the more memory you have over 16 GB, the less likely you are to encounter performance and memory issues.

Memory limitations can affect the performance or ability of SQL Server to build a partitioned index. This is especially the case when the index is not aligned with its base table or is not aligned with its clustered index, if the table already has a clustered index applied to it.

Partitioned Index Operations

Memory limitations can affect the performance or ability of SQL Server to build a partitioned index. This is especially the case with nonaligned indexes. Creating and rebuilding nonaligned indexes on a table with more than 1,000 partitions is possible, but is not supported. Doing so may cause degraded performance or excessive memory consumption during these 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. For example, a nonaligned partitioned index with 100 partitions requires sufficient memory to serially sort 4,000 (40 * 100) pages at the same time. If this memory is available, the build operation will succeed, but performance may suffer. If this memory is not available, the build operation will fail. Alternatively, an aligned partitioned index with 100 partitions requires only sufficient memory to sort 40 pages, because the sorts are not performed at the same time.

For both aligned and nonaligned indexes, the memory requirement can be greater if SQL Server is applying degrees of parallelism to the build operation on a multiprocessor computer. This is because the greater the degrees of parallelism, the greater the memory requirement. For example, if SQL Server sets degrees of parallelism to 4, a nonaligned partitioned index with 100 partitions requires sufficient memory for four processors to sort 4,000 pages at the same time, or 16,000 pages. If the partitioned index is aligned, the memory requirement is reduced to four processors sorting 40 pages, or 160 (4 * 40) pages. You can use the MAXDOP index option to manually reduce the degrees of parallelism.

DBCC Commands

With a larger number of partitions, DBCC commands could take longer to execute as the number of partitions increases.

Queries

Queries that use partition elimination could have comparable or improved performance with larger number of partitions. Queries that do not use partition elimination could take longer to execute as the number of partitions increases.

For example, assume a table has 100 million rows and columns `A`, `B`, and `C`. In scenario 1, the table is divided into 1000 partitions on column `A`. In scenario 2, the table is divided into 10,000 partitions on column `A`. A query on the table that has a WHERE clause filtering on column `A` will perform partition elimination and scan one partition. That same query may run faster in scenario 2 as there are fewer rows to scan in a partition. A query that has a WHERE clause filtering on column B will scan all partitions. The query may run faster in scenario 1 than in scenario 2 as there are fewer partitions to scan.

Queries that use operators such as TOP or MAX/MIN on columns other than the partitioning column may experience reduced performance with partitioning because all partitions must be evaluated.

Behavior Changes in Statistics Computation During Partitioned Index Operations

Beginning with SQL Server 2012, statistics are not created by scanning all the rows in the table when a partitioned index is created or rebuilt. Instead, the query optimizer uses the default sampling algorithm to generate statistics. After upgrading a database with partitioned indexes, you may notice a difference in the histogram data for these indexes. This change in behavior may not affect query performance. To obtain statistics on partitioned indexes by scanning all the rows in the table, use CREATE STATISTICS or UPDATE STATISTICS with the FULLSCAN clause.

Related Tasks

Tasks	**Topic**
Describes how to create partition functions and partition schemes and then apply these to a table and index.	Create Partitioned Tables and Indexes

Related Content

You may find the following white papers on partitioned table and index strategies and implementations useful.

- Partitioned Table and Index Strategies Using SQL Server 2008
- How to Implement an Automatic Sliding Window
- Bulk Loading into a Partitioned Table
- Project REAL: Data Lifecycle -- Partitioning

- Query Processing Enhancements on Partitioned Tables and Indexes
- Top 10 Best Practices for Building a Large Scale Relational Data Warehouse

Community Additions

Now it is available even in Express editon!

If you are running version 2016, install Service Pack 1- with SP 1 you can use partitioning in every edition of SQL Server:

https://blogs.technet.microsoft.com/dataplatforminsider/2016/11/16/sql-server-2016-service-pack-1-generally-available/

petr_cz
11/20/2016

and developer edition

and developer edition


LukasSteindl
5/24/2016

Really, did I miss something?

"Partitioned tables and indexes are not available in every edition of Microsoft SQL Server. "

I cannot believe it, "every edition", it's always only for Enterprise edition.


WangJiping
5/11/2016