

# Reorganize and Rebuild Indexes in the Database

This topic describes how to reorganize or rebuild a fragmented index in SQL Server 2017 by using SQL Server Management Studio or Transact-SQL. The SQL Server Database Engine automatically maintains indexes whenever insert, update, or delete operations are made to the underlying data. Over time these modifications can cause the information in the index to become scattered in the database (fragmented). Fragmentation exists when indexes have pages in which the logical ordering, based on the key value, does not match the physical ordering inside the data file. Heavily fragmented indexes can degrade query performance and cause your application to respond slowly.

You can remedy index fragmentation by reorganizing or rebuilding an index. For partitioned indexes built on a partition scheme, you can use either of these methods on a complete index or a single partition of an index. Rebuilding an index drops and re-creates the index. This removes fragmentation, reclaims disk space by compacting the pages based on the specified or existing fill factor setting, and reorders the index rows in contiguous pages. When ALL is specified, all indexes on the table are dropped and rebuilt in a single transaction.

Reorganizing an index uses minimal system resources. It defragments the leaf level of clustered and non-clustered indexes on tables and views by physically reordering the leaf-level pages to match the logical, left to right, order of the leaf nodes. Reorganizing also compacts the index pages. Compaction is based on the existing fill factor value.

## Before You Begin

### Detecting Fragmentation

The first step in deciding which defragmentation method to use is to analyze the index to determine the degree of fragmentation. By using the system function [sys.dm\\_db\\_index\\_physical\\_stats](#), you can detect fragmentation in a specific index, all indexes on a table or indexed view, all indexes in a database, or all indexes in all databases. For partitioned indexes, **sys.dm\_db\_index\_physical\_stats** also provides fragmentation information for each partition.

The result set returned by the **sys.dm\_db\_index\_physical\_stats** function includes the following columns.


Column	Description
avg_fragmentation_in_percent	The percent of logical fragmentation (out-of-order pages in the index)
fragment_count	The number of fragments (physically consecutive leaf pages) in the index
avg_fragment_size_in_pages	Average number of pages in one fragment in an index

After the degree of fragmentation is known, use the following table to determine the best method to correct the fragmentation.

avg_fragmentation_in_percent value	Corrective statement
> 5% and <= 30%	ALTER INDEX REORGANIZE
> 30%	ALTER INDEX REBUILD WITH (ONLINE = ON)*


\* Rebuilding an index can be executed online or offline. Reorganizing an index is always executed online. To achieve availability similar to the reorganize option, you should rebuild indexes online.

These values provide a rough guideline for determining the point at which you should switch between ALTER INDEX REORGANIZE and ALTER INDEX REBUILD. However, the actual values may vary from case to case. It is important that you experiment to determine the best threshold for your environment. Very low levels of fragmentation (less than 5 percent) should not be addressed by either of these commands because the benefit from removing such a small amount of fragmentation is almost always vastly outweighed by the cost of reorganizing or rebuilding the index.

 **Note:** In general, fragmentation on small indexes is often not controllable. The pages of small indexes are sometimes stored on mixed extents. Mixed extents are shared by up to eight objects, so the fragmentation in a small index might not be reduced after reorganizing or rebuilding the index.

## Limitations and Restrictions

- Indexes with more than 128 extents are rebuilt in two separate phases: logical and physical. In the logical phase, the existing allocation units used by the index are marked for deallocation, the data rows are copied and sorted, then moved to new allocation units created to store the rebuilt index. In the physical phase, the allocation units previously marked for deallocation are physically dropped in short transactions that happen in the background, and do not require many locks.
- Index options cannot be specified when reorganizing an index.
- The **ALTER INDEX REORGANIZE** statement requires the data file containing the index to have space available, because the operation can only allocate temporary work pages on the same file, not another file within the filegroup. So although the filegroup might have free pages available, the user can still encounter error 1105: "Could not allocate space for object <index name>. <table name> in database <database name> because the 'PRIMARY' filegroup is full."
- 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.

 **Note:** Starting 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. To obtain statistics on partitioned indexes by scanning all the rows in the table, use **CREATE STATISTICS** or **UPDATE STATISTICS** with the **FULLSCAN** clause.

## Permissions

Requires ALTER permission on the table or view. User must be a member of the **sysadmin** fixed server role or the **db\_ddladmin** and **db\_owner** fixed database roles.

## Check the fragmentation of an index

### SQL Server Management Studio

1. In Object Explorer, Expand the database that contains the table on which you want to check an index's fragmentation.
2. Expand the **Tables** folder.
3. Expand the table on which you want to check an index's fragmentation.
4. Expand the **Indexes** folder.
5. Right-click the index of which you want to check the fragmentation and select **Properties**.
6. Under **Select a page**, select **Fragmentation**.

The following information is available on the **Fragmentation** page:

#### Page fullness

Indicates average fullness of the index pages, as a percentage. 100% means the index pages are completely full. 50% means that, on average, each index page is half full.

**Total fragmentation**

The logical fragmentation percentage. This indicates the number of pages in an index that are not stored in order.

**Average row size**

The average size of a leaf level row.

**Depth**

The number of levels in the index, including the leaf level.

**Forwarded records**

The number of records in a heap that have forward pointers to another data location. (This state occurs during an update, when there is not enough room to store the new row in the original location.)

**Ghost rows**

The number of rows that are marked as deleted but not yet removed. These rows will be removed by a clean-up thread, when the server is not busy. This value does not include rows that are being retained due to an outstanding snapshot isolation transaction.

**Index type**

The type of index. Possible values are **Clustered index**, **Nonclustered index**, and **Primary XML**. Tables can also be stored as a heap (without indexes), but then this Index Properties page cannot be opened.

**Leaf-level rows**

The number of leaf level rows.

**Maximum row size**

The maximum leaf-level row size.

**Minimum row size**

The minimum leaf-level row size.

**Pages**

The total number of data pages.

**Partition ID**

The partition ID of the b-tree containing the index.

**Version ghost rows**

The number of ghost records that are being retained due to an outstanding snapshot isolation transaction.

## Transact-SQL

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**.

```
USE AdventureWorks2012;
GO
-- Find the average fragmentation percentage of all indexes
-- in the HumanResources.Employee table.
SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats (DB_ID(N'AdventureWorks2012'),
OBJECT_ID(N'HumanResources.Employee'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
ON a.object_id = b.object_id AND a.index_id = b.index_id;
GO
```

The statement above might return a result set similar to the following:

index_id	name	avg_fragmentation_in_percent
1	PK_Employee_BusinessEntityID	0
2	IX_Employee_OrganizationalNode	0
3	IX_Employee_OrganizationalLevel_OrganizationalNode	0
5	AK_Employee_LoginID	66.6666666666667
6	AK_Employee_NationalIDNumber	50
7	AK_Employee_rowguid	0

6 row(s) affected)

For more information, see [sys.dm\\_db\\_index\\_physical\\_stats \(Transact-SQL\)](#).

## Reorganize or rebuild an index

### SQL Server Management Studio

1. In Object Explorer, Expand the database that contains the table on which you want to reorganize an index.
2. Expand the **Tables** folder.
3. Expand the table on which you want to reorganize an index.
4. Expand the **Indexes** folder.
5. Right-click the index you want to reorganize and select **Reorganize**.
6. In the **Reorganize Indexes** dialog box, verify that the correct index is in the **Indexes to be reorganized** grid and click **OK**.
7. Select the **Compact large object column data** check box to specify that all pages that contain large object (LOB) data are also compacted.
8. Click **OK**.

### Transact-SQL

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**.

```
USE AdventureWorks2012;
GO
-- Reorganize the IX_Employee_OrganizationalLevel_OrganizationalNode
-- index on the HumanResources.Employee table.

ALTER INDEX IX_Employee_OrganizationalLevel_OrganizationalNode
ON HumanResources.Employee
REORGANIZE ;
GO
```

## Reorganize all indexes in a table

### SQL Server Management Studio

1. In Object Explorer, Expand the database that contains the table on which you want to reorganize an index.
2. Expand the **Tables** folder.
3. Expand the table on which you want to reorganize the indexes.
4. Right-click the **Indexes** folder and select **Reorganize All**.
5. In the **Reorganize Indexes** dialog box, verify that the correct indexes are in the **Indexes to be reorganized**. To remove an index from the **Indexes to be reorganized** grid, select the index and then press the **Delete** key.
6. Select the **Compact large object column data** check box to specify that all pages that contain large object (LOB) data are also compacted.
7. Click **OK**.

### Transact SQL

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**.

```
USE AdventureWorks2012;
GO
-- Reorganize all indexes on the HumanResources.Employee table.
ALTER INDEX ALL ON HumanResources.Employee
REORGANIZE ;
GO
```

## Rebuild an index

### SQL Server Management Studio

1. In Object Explorer, Expand the database that contains the table on which you want to reorganize an index.
2. Expand the **Tables** folder.

3. Expand the table on which you want to reorganize an index.
4. Expand the **Indexes** folder.
5. Right-click the index you want to reorganize and select **Rebuild**.
6. In the **Rebuild Indexes** dialog box, verify that the correct index is in the **Indexes to be rebuilt** grid and click **OK**.
7. Select the **Compact large object column data** check box to specify that all pages that contain large object (LOB) data are also compacted.
8. Click **OK**.

## Rebuild a defragmented index

### Transact-SQL

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query window and click **Execute**. The example rebuilds a single index on the **Employee** table.

```
USE AdventureWorks2012;
GO
ALTER INDEX PK_Employee_BusinessEntityID ON HumanResources.Employee
REBUILD;
GO
```

## Rebuild all indexes in a table

### Transact-SQL

1. In **Object Explorer**, connect to an instance of Database Engine.
2. On the Standard bar, click **New Query**.
3. Copy and paste the following example into the query. The example specifies the keyword **ALL**. This rebuilds all indexes associated with the table. Three options are specified.

```
USE AdventureWorks2012;
GO
ALTER INDEX ALL ON Production.Product
REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
STATISTICS_NORECOMPUTE = ON);
GO
```

For more information, see [ALTER INDEX \(Transact-SQL\)](#).