**Lesson Introduction**

If you understand how SQL Server 2012 accesses data, you can create indexes to facilitate the searching of data. One of the important aspects of database operations is the users’ perceived access to the data. To ensure that the users’ perceived access to the data is not impaired, you should create the right type of indexes on the right columns in the right tables.

# How SQL Server 2012 Retrieves Data

SQL Server 2012 can access data in the following two ways.

* It can scan all the pages that are associated with a table. This method is known as a table scan.
* It can use an index to traverse through the index structure to retrieve the rows that meet the specified criteria.

The query optimizer, a component of the SQL Server 2012 that is responsible for generating execution plans, determines if it is more efficient to make use of an index to support a query or perform a table scan to retrieve the results.

The way that SQL Server 2012 accesses data from the table depends on whether you create the SQL Server 2012 indexes. You can create indexes to provide order to the data by using clustered indexes. Alternatively, you can create nonclustered indexes that provide a pointer to the data.

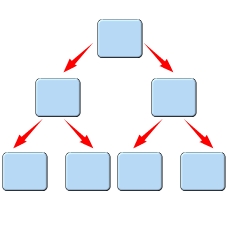
# What Is a Heap?

SQL Server 2012 stores data in an 8 KB page. You can store many rows within a single page, and SQL Server 2012 groups eight of these pages into a 64 KB extent. When the data in the pages of a single extent belong to the same object, such as a table, it is known as a uniform extent. However, data from different objects can also belong to the same extent. This is known as a mixed extent.

As a result, data from a single table may be spread across many extents across the disk. When a table does not contain a clustered index, the data is stored in no particular order and is known as a heap.

To retrieve all the data from a heap, SQL Server 2012 should identify all the pages and extents that contain the data within the heap. To identify the pages and extents, you can query the ID column and the Index\_ID column of the sys.partitions table. The Index\_ID value of zero indicates that the data for the table is stored in a heap. The First\_IAM\_page column in the sys.partitions table points to a page known as the index allocation map. This page maintains the heap by storing information about where SQL Server 2012 stores the extents of a particular heap.

# What Is a Clustered Index?



You can create only a single clustered index against one or more columns of a table or a view. A maximum of 16 columns can be a part of a clustered index. When the clustered index is created, a Balance-Tree or B-Tree index structure is created. A clustered index is identified in the sys.partitions table and has an Index\_ID value of one.

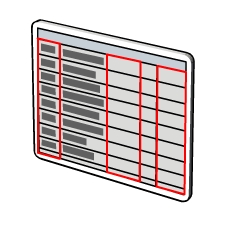
A B-Tree index structure contains three levels. At the top of the B-Tree is the root level, a single page that holds the root of the index data. For example, if a clustered index is created on the column Last name, the values of the Last name column are stored in the root page. However, the root page is limited to a single page, which is 8 KB. Because a whole data may reference many rows of data that extend beyond the 8KB limit, the root page will contain pointers to data in the index at the second level, which is known as the nonleaf level, or the intermediate level.

The nonleaf level is an intermediate level within the B-Tree structure that contains all the data referenced to the column that has been indexed. Two pages initially flow from the root page to host the index data by using the B-Tree layout. If the index data does not fit into the two pages, another intermediary level is created below the two nonleaf levels. This level creates another four pages to store the indexed data. To accommodate all the indexed data within the B-Tree structure, more intermediate levels are created if required.

The final level of the index structure is the leaf level. In a clustered index, the leaf level is the actual data itself. This level contains the data that makes up the entire rows of a table in sorted order based on the column that has been indexed. You should define the clustered index with as few columns as possible to keep the size of the index small. The index should have one or more of the following attributes.

* It should be unique or should contain many distinct values.
* It should be defined as IDENTITY because the column is guaranteed to be unique within the table.
* It should be accessed sequentially.
* It should be frequently used to sort the data retrieved from a table.

# What Is a Nonclustered Index?



You can create up to 249 nonclustered indexes for a table or a view. These indexes are created to typically support the queries that are used to retrieve data from a table or a view. A nonclustered index is identified in the sys.partitions table with an Index\_ID value of 2 to 250.

When you create a nonclustered index, it creates a B-Tree index structure. Similar to clustered index, the B-Tree index structure for nonclustered indexes contains three levels—root level, nonleaf level, and leaf level. The root level and the nonleaf level are similar to those in a clustered index. However, the leaf level in a nonclustered index is not the actual data itself. This level contains pointers to the data that is stored outside the B-Tree structure.

You should use a nonclustered index for queries that have the following attributes.

* Create nonclustered indexes on JOINS or GROUP BY clauses defined in queries.
* Contain columns frequently involved in search conditions of a query, such as the WHERE clause, that return exact matches.

### Lesson Introduction

Creating an index on a single column is a straightforward process. However, there may be situations in which you may want to create indexes on multiple columns. You can better facilitate the searching of data by creating indexes on multiple columns. In this respect and dependant on the queries that you use, you may get better database performance. Sometimes, you may face a situation, where you want to enforce uniqueness on more than one column. In SQL Server 2012, you can easily create unique indexes. In addition, index partitioning can be used to spread the contents of the index across different data files. You can also use indexes on computed columns.

Use the following syntax to create indexes.

|  |
| --- |
| CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX *index\_name*      ON <object> **(** *column* [ ASC | DESC ] [ **,**...*n* ] **)**      [ INCLUDE **(** *column\_name* [ **,**...*n* ] **)** ]  [ WHERE <filter\_predicate> ]      [ WITH **(** <relational\_index\_option> [ **,**...*n* ] **)** ]      [ ON { *partition\_scheme\_name* **(** *column\_name* **)**           | *filegroup\_name*           | default           }      ]  [ FILESTREAM\_ON { *filestream\_filegroup\_name* | *partition\_scheme\_name* | "NULL" } ]  [ ; ] |

The following table describes the clauses used in the syntax.

|  |  |
| --- | --- |
| **Clause** | **Description** |
| UNIQUE | ***This creates a unique index on a table or view. In a unique index no two rows are permitted to have the same index key value. Duplicate values must be removed before a unique index can be used. Columns that are used in a unique index should be set to NOT NULL, because multiple null values are considered as duplicates when a unique index is created.*** |
| [ CLUSTERED | NONCLUSTERED ] | This clause defines the type of index that will be created. If jut the CREATE INDEX is specified then a nonclustered index is created by default as you can have more than one nonclustered index. |
| ***index\_name*** | This is the name of the index. Index names must be unique within a table or view but do not have to be unique within a database. |
| ***ON <object> ( column [ ASC | DESC ] [ ,...n ] )*** | ***This clause defines the column in a table or view and whether the index is assorted into ascending or descending order. You can specify two or more columns to create a composite index that cannot exceed 16 columns or 900 bytes in length. Columns that are of the large object (LOB) data types ntext, text, varchar(max), nvarchar(max), varbinary(max), xml, or image cannot be specified as key columns for an index.*** |
| ***[ INCLUDE ( column\_name [ ,...n ] ) ]*** | ***This clause is used with nonclustered indexes only. You can specify the non-key columns to be added to the leaf level of the nonclustered index. You can specify two or more columns to create a composite index that cannot exceed 16 columns or 900 bytes in length.*** |
| ***[ WHERE <filter\_predicate> ]*** | This clause creates a filtered index by specifying which rows to include in the index. The filtered index must be a nonclustered index on a table and creates filtered statistics for the data rows in the filtered index.  The filter predicate uses simple comparison logic and cannot reference a computed column, a UDT column, a spatial data type column, or a hierarchyID data type column. |
| ***[ WITH ( <relational\_index\_option> [ ,...n ] ) ]*** | ***This clause includes additional settings that can be used to optimize an index.*** |
| ***[ ON { partition\_scheme\_name ( column\_name ) | filegroup\_name| default}*** | ***This clause defines the location on which the index is stored. If omitted, the index will be stored on the same filegroup location as the table on which the index is defined. However, you can specify a separate filegroup that can be used to separate the table data from the index data for improved performance. However, you must maintain both the index and table at the same time. For example, back up the index and data at the same time for the correct alignment.******You can also set the option of default. This will store the index in the filegroup that has been defined as the database default filegroup. Alternatively, you can specify the partition scheme that defines the filegroups onto which the partitions of a partitioned index will be mapped.*** |
| ***[ FILESTREAM\_ON { filestream\_filegroup\_name | partition\_scheme\_name | "NULL" } ]*** | ***This clause specifies the placement of FILESTREAM data for the table when a clustered index is created. The FILESTREAM\_ON clause allows FILESTREAM data to be moved to a different FILESTREAM filegroup or partition scheme. If NULL is specified it defines that there is no filestream filegroup. filestream\_filegroup\_name is the name of a FILESTREAM filegroup.*** |

The *[ WITH ( <relational\_index\_option> [ ,...n ] ) ]* clause helps youdefine the following additional options that can be set with  
the CREATE INDEX statement.

|  |  |
| --- | --- |
| **Option** | **Description** |
| DATA\_COMPRESSION | Specifies the data compression option for the specified index, partition number, or range of partitions. The options are as follows: ***NONE - Index or specified partitions are not compressed.******ROW - Index or specified partitions are compressed by using row compression.******PAGE - Index or specified partitions are compressed by using page compression.*** |
| ***FILLFACTOR = Fillfactor Value*** | Controls the amount of free space that is defined within the leaf level of a nonclustered index by defining an integer value for how full a page can be. |
| ***PAD\_INDEX = ON|OFF*** | Controls the amount of free space that is defined within the non leaf level of a nonclustered index. This option must be used in conjunction of the FILLFACTOR option and uses the same integer value. |
| SORT\_IN\_TEMPDB = { ON | OFF } | When set to ON, the intermediate sort results that are used to build the index are stored in tempdb. This may reduce the time required to create an index if tempdb is on a different set of disks than the user database. However, this increases the amount of disk space that is used during the index build. When set to OFF, the index is stored in the same database as the index. The default is OFF. |
| IGNORE\_DUP\_KEY = { ON | OFF } | Specifies the error response to duplicate key values in a multiple-row insert operation on a unique clustered or unique nonclustered index. The default is OFF and when set to OFF an error message is issued and the entire INSERT transaction is rolled back. When set to on only the rows violating the unique index fail. |
| ***[STATISTICS\_NORECOMPUTE = { ON | OFF }*** | Specifies whether distribution statistics are recomputed. The default is OFF, which means that statistics are updated automatically. ON means that out-of-date statistics are not automatically recomputed. |
| ***DROP\_EXISTING = { ON | OFF }*** | Specifies that the named, pre-existing clustered, or nonclustered is dropped and rebuilt. The default is OFF. The existing index is dropped and rebuilt. The index name specified must be the same as a currently existing index; however, the index definition can be modified. |
| ONLINE = { ON | OFF } | Specifies whether underlying tables and associated indexes are available for queries and data modification during the index operation. The default is OFF. This prevents all user access to the underlying table for the duration of the operation. ON enables queries or updates to the underlying table and indexes to proceed. |
| ALLOW\_ROW\_LOCKS = { ON | OFF } | Specifies whether row locks are allowed. The default is ON. Row locks are not allowed when this option is set to OFF. |
| ALLOW\_PAGE\_LOCKS = { ON | OFF } | Specifies whether page locks are allowed. The default is ON. Page locks are not allowed when this option is set to OFF. |
| MAXDOP = max\_degree\_of\_parallelism | Use MAXDOP to limit the number of processors used in a parallel plan execution. The maximum is 64 processors. *max\_degree\_of\_parallelism* can be set to the following values:   * 1 - Suppresses parallel plan generation. * >1 - Restricts the maximum number of processors used in a parallel index operation to the specified number or fewer based on the current system workload. * 0 (default) - Uses the actual number of processors or fewer based on the current system workload. |

**Using SQL Server Management Studio**

To create an index in the SQL Server Management Studio console, perform the following steps.

1. In **Object Explorer**, right-click the table and click **Design**. The table opens in Table Designer.
2. On the **Table Designer** menu, click **Indexes/Keys**.
3. Click **Add**. The Selected Primary/Unique Key or Index list displays the system-assigned name of the new index.
4. In the grid, click **Type**.
5. In the list, to the right of the property, select **Index**.
6. Under **Columns**, select the columns that you want to index. You can select up to 16 columns. For optimal performance, select only one or two columns per index. For each column you select, indicate whether the index arranges values of this column in ascending or descending order.
7. In the grid, click **Is Unique**.
8. In the list, to the right of the property, select **Yes**.
9. Select **Ignore duplicate keys**.
10. If you want to ignore data that would create a duplicate key in the unique index (with the INSERT statement), select **Yes**.

The index is created in the database when you save the table or diagram

Unlike primary keys, you can define up to 249 unique constraints on a table. Defining unique constraints can be useful if you need to enforce uniqueness on records that have a unique business meaning. In this demonstration, you will view how to create a unique index in SQL Server 2008.

# Creating Partitioned Indexes



Similar to tables, you can also place indexes on separate partitions based on a partition schema. Clustered and nonclustered indexes can be separated horizontally and placed on separate physical disks by using filegroups based on a range of values.

To create a partitioned index, perform the following steps.

1. **Create a Partition Function**. The partition function specifies the data type of the key column that is used to partition the data. It also sets the boundaries for each partition.
2. **Create a Partition Scheme**. The partition scheme maps the partitions defined in the partition function against the filegroups that will store each partition of data.
3. **Map the partition scheme within a CREATE INDEX statement**. The CREATE INDEX statement is used to define that the index is stored against the partition scheme, which defines the filegroups and the boundaries of the indexed data that are stored in a specific filegroup.

[Click to view and print code examples.](https://business.microsoftelearning.com/plo_content/org6024/course247638/media/crse10364ae_01_02_04_ja01.mht)

An index is partitioned in the same way as a table. This is known as index alignment and occurs when you issue a CREATE INDEX statement. However, you can use a separate partitioning strategy for the indexes that support the table to reduce the disk I/O and improve the performance.

Note that if you create a partitioned index on a unique index, which is either clustered or nonclustered, there can be a conflict between the column that is used as a unique identifier of the rows and the column that is used to define the partition boundaries. Therefore, you must select the partitioning column from those that are used in the unique index key. For example, an Orders table uses an orderid to identify orders. However, the orderdate column is used as the basis for partitioning the index. You can use both the orderid and the orderdate in a composite index for index partitioning.

# Creating Indexes to Cover Queries

You can create nonclustered indexes to cover the queries that are issued by applications. Covering a query involves defining a nonclustered index that incorporates the common columns that are retrieved by queries. The data is stored at the leaf level in the nonclustered index. If the query includes columns in the select list that are stored in the leaf level of the nonclustered index, SQL Server 2012 does not query the data pages to retrieve the data because the information is stored at the leaf level of the nonclustered index.

Tip: When you create an index, statistics are generated to inform the query optimizer about the usefulness of the index. In a composite index, the statistics are only based on the first column defined on the index. Therefore, select a column that has unique records.

In addition, filtered indexes have been introduced in SQL Server 2012. Filtered indexes use a WHERE predicate in the CREATE INDEX statement that controls the rows that are used to populate the leaf level of a nonclustered index. Filtered indexes can be useful when a table contains many years worth of data. When 90% of the queries only retrieve the data for the last year, it would be worthwhile to create a filtered index on the last years data.

To create indexes that cover your queries, you should know the queries that your client applications issue. The ability to retrieve results from the leaf level of a nonclustered index can improve the speed with which the data is retrieved despite the fact that the size of the key index length is bigger, because more columns are used to cover the query. However, used in conjunction with filtered indexes you can reduce the size of the index size by populating the index only with the data that is required for the most common queries. It can also reduce the storage cost and maintenance window that is required to maintain indexes.

**Covered Indexes**

Performance gains are achieved when the index contains all columns in the query. The query optimizer can locate all the column values in the index; table or clustered index data is not accessed resulting in fewer disk I/O operations.

CREATE NONCLUSTERED INDEX NCL\_Firstname\_Lastname\_Email

ON Person.Contact (Lastname, Firstname,  EmailAddress);

GO

You can extend the functionality of nonclustered indexes by adding non-key columns to the leaf level of the nonclustered index. By including non-key columns, you can create nonclustered indexes that cover more queries. This is because the non-key columns have the following benefits.

* They can be data types not allowed as index key columns.
* They are not considered by the Database Engine when calculating the number of index key columns or index key size.
* They can be used to overcome the limitation imposed on key columns of 16 columns or 900 bytes.

**Filtered Indexes**

When a column contains only a small number of relevant values for queries, you can create a filtered index on the subset of values. For example, when the values in a column are mostly NULL and the query selects only from the non-NULL values, you can create a filtered index for the non-NULL data rows.

For example, the AdventureWorks database has a Production.BillOfMaterials table with 2679 rows. Only 199 rows contain a non-NULL value for the EndDate column; the other 2480 rows contain NULL. The following filtered index will cover queries that return the columns defined in the index and that select only rows with a non-NULL value for the EndDate column.

CREATE NONCLUSTERED INDEX FIBillOfMaterialsWithEndDate ON Production.BillOfMaterials (ComponentID, StartDate)

WHERE EndDate IS NOT NULL;

GO

The following query makes the best use of the index created. Any other query that calls for columns that are NULL would require SQL Server 2012 to retrieve the data from the data pages.

SELECT ComponentID, StartDate FROM Production.BillOfMaterials WHERE EndDate IS NOT NULL; GO

You cannot create a filtered index on a view. However, the query optimizer can benefit from a filtered index defined on a table that is referenced in a view.

The following example creates a nonclustered index on the Person.Address table with four included columns. The index key column is PostalCode and the non-key columns are AddressLine1, AddressLine2, City and StateProvinceID.

USE AdventureWorks;GO

CREATE NONCLUSTERED INDEX IX\_Address\_PostalCode

ON Person.Address (PostalCode)

INCLUDE (AddressLine1, AddressLine2, City, StateProvinceID);

GO

# How to Create Indexes on Computed Columns

A computed column is created by using an expression that can use other columns in the same table. You can create an index on a computed column in the following conditions are met.

* The computed column is deterministic and precise.
* The ANSI\_NULL connection level option is set to ON when the CREATE TABLE statement is run.
* The computed column is not based on image, ntext and text data type.

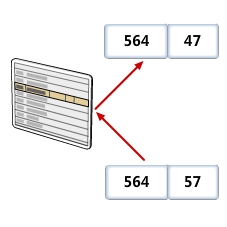
However, after these conditions are met, the index might not be used by the query optimizer. When an index is created on computed columns, it is used only by client applications that have the following connection settings.

* ANSI\_NULLS is set to ON.
* ANSI\_PADDING is set to ON.
* ANSI\_WARNING is set to ON.
* CONCAT\_NULL\_YIELDS\_NULL is set to ON.
* QUOTED\_IDENTIFIER is set to ON.
* ARITHABORT is set to ON.
* NUMERIC\_ROUNDABORT is set to OFF.

### Lesson Introduction

After the indexes are operational, there can be situations that can slow down a computer running SQL Server 2012 based on the index usage. Typically, this may manifest itself as phone calls to the helpdesk about slow updates and inserts against a transactional system. An index on this type of system is not static and will change as modifications are made to the data. You can employ techniques that limit the impact of changes to the indexes to the benefit the users.

# How SQL Server 2008 Handles Rows in Pages



Different types of data modifications can take place in a database. These modifications can affect the data in a table and its associated indexes. This, in turn, affects how the data is stored in the pages and extents in the data files. When you insert a new row in a table, SQL Server 2012 accommodates it wherever space is available in the heap.

However, the deletion of row can have an impact on both the data and index pages. When you issue a DELETE statement, which affects the data at the leaf level of an index, the data is not removed immediately. Instead, it is initially marked as a ghost record. This reduces locking of adjacent rows to the deleted records. SQL Server 2012 periodically checks for the existence of ghost records and removes them. In addition, a gap is left where the previous ghost record resided. This space can be reused if new data is inserted.

Updates can often take place without causing the data to move. However, there can be situation where SQL Server 2008 converts an UPDATE statement into a DELETE statement followed by an INSERT statement. An example of such a situation is an update made to data that is part of a clustered index, and the new value requires the record to be located in a different part of the clustered index.

In addition, an update to a row might need more space than is available in the page that it is currently occupying. In such situations, the row is moved to a new data page and a forwarding pointer is left in its original location ensuring that nonclustered indexes need not be changed. If future updates on the same record reduce the size down to fit in its original location, the forwarding pointer is removed.

Tip: Sometimes, when you run DBCC ShowContig against a table, the amount of rows that are returned in the results is greater than a value that is returned when you run a SELECT COUNT(\*) statement against the same table. This is an indication that the table contains rows that has forwarding pointers.

# Using the FILLFACTOR Option to Control Page Splits

The FILLFACTOR option controls the free space in the leaf level of the index only. You can use it only when the index is created or rebuilt. You use the WITH option in the CREATE INDEX statement and set a percentage value that determines the amount of space that should be filled in each page. The following example creates a nonclustered index named NCL\_Contact\_Lastname on the Lastname column of the Person.Contact table. The WITH option specifies a fill factor value of 80 percent, which means that 20 percent of a page will contain free space.

CREATE NONCLUSTERED INDEX  NCL\_Contact\_Lastname

ON Person.Contact(Lastname ASC)

WITH FILLFACTOR = 80

You can select any value between 1 and 100. The default value is zero, which completely fills a page. In this case, the value of zero and 100 are the same. A correctly selected fill factor value can reduce potential page splits by providing enough space for index expansion as data is added to the underlying table.

You can use the following fill factor setting for differing database requirement.

* Use a low fill factor value for transactional (OLTP) systems to provide maximum growth in table and disk space is not a major concern.
* Use a high fill factor value for analytical (OLAP) systems. If you place the data close together, less disk IO is required to retrieve the data.

You can also define a default value for the fill factor. You must show advanced options and then run the sp\_configure 'fill factor' as shown in the following example.

Use AdventureWorks;

GO

sp\_configure 'show advanced options', 1;

GO

RECONFIGURE;

GO

sp\_configure 'fill factor', 100;

GO

RECONFIGURE;

GO

# Using PAD\_INDEX Option to Control Page Splits

You can use the PAD\_INDEX option to control the free space on the nonleaf level of an index only. You cannot specify a percentage value for this option as you can with the FILLFACTOR option. You must use the FILLFACTOR option if you want to use the PAD\_INDEX option because the PAD\_INDEX option inherits the percentage of free space from the FILLFACTOR option. However, SQL Server 2008 can easily accommodate one row of the maximum index size regardless of the fill factor percentage.

Therefore, if you run the following statement, an error is returned.

CREATE NONCLUSTERED INDEX NCL\_Contact\_Lastname

ON Person.Contact(Lastname ASC)

WITH PAD\_INDEX = 80

Msg 153, Level 15, State 1, Line 5

Invalid usage of the option PAD\_INDEX in the INDEX statement.

To use of the PAD\_INDEX option the following statement must be executed.

CREATE NONCLUSTERED INDEX NCL\_Contact\_Lastname

ON Person.Contact(Lastname ASC)

WITH (FILLFACTOR = 80, PAD\_INDEX = ON)

### Lesson Introduction

To maintain the operation of a computer running SQL Server 2012, you should continuously review the state of the indexes that are present on the system. SQL Server 2012 provides several tools and techniques that you can use to evaluate a specific index. The tools and techniques that are presented in this lesson will provide you with the ability to make more informed decisions on how you should maintain your indexes.

# Understanding Index Statistics

When you create an index, statistics are automatically generated on the index. On a single column index, the statistics are based on the data in the column. In a multi-column index, the statistics are based on the first column defined in the CREATE INDEX statement. These statistics are used by the query optimizer to determine the usefulness of the query when retrieving data from the database and modifying the data in the tables. You can also generate statistics on a column that is not the first column defined in a multi-column index or on columns that are not a part of an index by using the CREATE STATISTICS statement. The following table describes the statistics that are generated when the CREATE INDEX and CREATE STATISTICS statement are used.

**Selectivity**

Selectivity is defined as a percentage of rows in a table that are accessed or returned by a query. Selectivity is a statistic used for the SELECT, UPDATE, and DELETE statements.

Data is said to be highly selective when the number of rows returned from a SELECT, UPDATE, and DELETE statement represents a low percentage of the total possible rows in a table.

Data is categorized as low selective when the number of rows returned from a SELECT, UPDATE, and DELETE statement represents a high percentage of the total possible rows in a table.

When deciding the columns on which to build an index, you should consider the selectivity of the data. For example, an index on a column that contains unique values, such as a ProductID column, which can also return a single value, will promote greater use of the index rather than queries that have the potential for a table scan.

**Density**

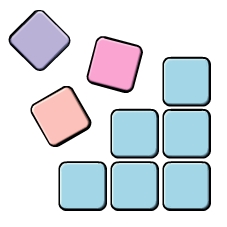
Density is defined as the average percentage of duplicate rows in an index. An index with a large number of duplicates has a high density and a unique index has a low density. This in part is related to selectivity. For example, an index with many duplicate values, such as last name, has a high density and lower selectivity than an index on a column such as a National Insurance number or passport number.

**Distribution**

Distribution is defined as the amount of data that exists for a range of values. If there are few unique values in a table, data retrieval may be slow due to the distribution of data. For example, a customer table may show a high occurrence of the same city.

This can be related to density because there may be much duplication of data. However, the data can be focused in a specific range. For example, the  Last name Smith can affect the distribution of data in a table if there are many Smith's in the table.

# Data Fragmentation



It is a common misconception that fragmentation affects database performance. However, this effect depends upon the environment on which the database is based.

When you add or remove data or modify a table, the index and data pages are adjusted to accommodate the changing data. This can lead to page splits, which increase the size of the table, and the length of time needed to process queries. As a result, data fragmentation occurs. Index 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.

Index fragmentation can be beneficial in transactional (OLTP) environments because these types of environments are write-intensive with a large number of concurrent users. Fragmentation enables gaps to occur in the data pages that can be replaced with new data when it is inserted. However, excessive fragmentation can impair the statistics of an index if the fragmentation is not managed.

As a result, fragmentation in an analytical (OLAP) environment can be detrimental to the performance of the database because this environment is typically a read-intensive environment.

Several methods can be used to keep track of the fragmentation of both data pages and indexes. You can also manage the fragmentation of the indexes by recreating or rebuilding an index.

# Managing Indexes

You can perform the following tasks to manage indexes on a database.

**Modify indexes**

Use the following syntax to alter an index.

|  |
| --- |
| ALTER INDEX { index\_name | ALL }  ON <object>  { REBUILD   [ [PARTITION = ALL]       [ WITH ( <rebuild\_index\_option> [ ,...n ] ) ]  | [ PARTITION = partition\_number  [ WITH ( <single\_partition\_rebuild\_index\_option>                          [ ,...n ] )                  ]              ]          ]    | DISABLE  | REORGANIZE      [ PARTITION = partition\_number ]      [ WITH ( LOB\_COMPACTION = { ON | OFF } ) ]  | SET ( <set\_index\_option> [ ,...n ] )      }  [ ; ] |

The following table describes the clauses used in the syntax.

|  |  |
| --- | --- |
| **Clause** | **Description** |
| index\_name | ALL | This is the name of the index or ALL Specifies all indexes associated with the table or view regardless of the index type. |
| ***ON <object>*** | This clause provides the name of the object on which the indexes reside and can be defined as database.schema.table or view name. |
| ***REBUILD [ WITH (<rebuild\_index\_option> [ ,... n]) ]*** | This clause specifies that the index will be rebuilt by using the same columns, index type, uniqueness attribute, and sort order. This clause is equivalent to DBCC DBREINDEX. REBUILD enables a disabled index. Rebuilding a clustered index does not rebuild associated non-clustered indexes unless the keyword ALL is specified.  When you rebuild an XML index or a spatial index, the options ONLINE = ON and IGNORE\_DUP\_KEY = ON are not valid. |
| ***PARTITION*** | This clause specifies that only one partition of an index will be rebuilt or reorganized. PARTITION cannot be specified if index\_name is not a partitioned index. PARTITION = ALL Rebuilds all partitions. PARTITION = partition\_number is the partition number of a partitioned index that is to be rebuilt or reorganized. partition\_number is a constant expression that can reference variables. These include user-defined type variables or functions and user-defined functions but cannot reference a Transact-SQL statement. partition\_number must exist or the statement fails. |
| ***WITH (<single\_partition\_rebuild\_index\_option>)*** | SORT\_IN\_TEMPDB, MAXDOP, and DATA\_COMPRESSION are the options that can be specified when you rebuild a single partition (PARTITION = n). XML indexes cannot be specified in a single partition rebuild operation.  Rebuilding a partitioned index cannot be performed online. The entire table is locked during this operation. |
| ***DISABLE*** | This clause marks the index as disabled and unavailable for use by the Database Engine. Any index can be disabled. The index definition of a disabled index remains in the system catalog with no underlying index data. Disabling a clustered index prevents user-access to the underlying table data. |
| ***REORGANIZE*** | This clause specifies the index leaf level will be reorganized. This clause is equivalent to DBCC INDEXDEFRAG. ALTER INDEX REORGANIZE statement is always performed online. This means long-term blocking table locks are not held and queries or updates to the underlying table can continue during the ALTER INDEX REORGANIZE transaction. |
| ***WITH ( LOB\_COMPACTION = { ON | OFF } )*** | This clause specifies that all pages that contain large object (LOB) data are compacted. The LOB data types are image, text, ntext, varchar(max), nvarchar(max), varbinary(max), and xml. Compacting this data can improve disk space use. The default is ON. |
| ***SET ( <set\_index option> [ ,... n] )*** | Standard relational options can also be used with the ALTER INDEX statement including  SET cannot be specified for a disabled index  See the CREATE INDEX syntax for more information:   * FILLFACTOR = fillfactor * PAD\_INDEX = { ON | OFF } * SORT\_IN\_TEMPDB = { ON | OFF } * IGNORE\_DUP\_KEY = OFF * STATISTICS\_NORECOMPUTE = { ON | OFF } * ONLINE = OFF * ALLOW\_ROW\_LOCKS = { ON | OFF } * ALLOW\_PAGE\_LOCKS = { ON | OFF }  NOTE: An index cannot be reorganized when ALLOW\_PAGE\_LOCKS is set to OFF * MAXDOP = max\_degree\_of\_parallelism * DATA\_COMPRESSION = NONE | ROW |PAGE |

**Example A: Rebuilding an index**

The following example rebuilds a single index, PK\_Employee\_EmployeeID on the Employee table.

|  |
| --- |
| USE AdventureWorks;  GO  ALTER INDEX PK\_Employee\_EmployeeID ON HumanResources.Employee  REBUILD;  GO |

### Example B: Rebuilding all indexes on a table and specifying options

The following example specifies the keyword ALL. This rebuilds all indexes associated with the table. The options FILLFACTOR, SORT\_IN\_TEMPDB, and STATISTICS\_NORECOMPUTE are set to ON.

|  |
| --- |
| USE AdventureWorks;  GO  ALTER INDEX ALL ON Production.Product  REBUILD WITH (FILLFACTOR = 80, SORT\_IN\_TEMPDB = ON,  STATISTICS\_NORECOMPUTE = ON);  GO |

### Example C: Setting options on an index

The following example sets several options to ON including STATISTICS\_NORECOMPUTE, IGNORE\_DUP\_KEY, and ALLOW\_PAGE\_LOCKS on the index AK\_SalesOrderHeader\_SalesOrderNumber.

|  |
| --- |
| USE AdventureWorks;  GO  ALTER INDEX AK\_SalesOrderHeader\_SalesOrderNumber ON  Sales.SalesOrderHeader  SET (  STATISTICS\_NORECOMPUTE = ON,  IGNORE\_DUP\_KEY = ON,  ALLOW\_PAGE\_LOCKS = ON  ) ;  GO |

### Example D: Disabling an index

The following example disables a nonclustered index named IX\_Employee\_ManagerID on the Employee table.

|  |
| --- |
| USE AdventureWorks;  GO  ALTER INDEX IX\_Employee\_ManagerID ON HumanResources.Employee  DISABLE ;  GO |

### Example E: Changing the compression setting of an index

### The following example rebuilds an index on a nonpartitioned table and sets page level compression.

|  |
| --- |
| ALTER INDEX IX\_INDEX1  ON T1  REBUILD  WITH ( DATA\_COMPRESSION = PAGE )  GO |

**Remove indexes**

You can use the SQL Server Management Studio console or the DROP INDEX statement to remove an index. When an index is removed SQL Server 2008 reclaims the space occupied by the index. Furthermore, should a clustered index be removed, all non clustered indexes are rebuilt automatically.

You cannot remove an index that has an associated primary key or unique constraint. The constraint must be dropped before the index is removed.

You can remove an index in the SQL Server Management Studio console by navigating in Object Explorer to the database. You then navigate to the specific table and then to the Indexes node. Locate the particular index, right-click the index, and then click **Delete**.

Alternatively, you can use the following Transact-SQL statement to remove an index named PK\_AWBuildVersion\_SystemInformationID in the AdventureWorks database.

USE AdventureWorks

GO

DROP INDEX PK\_AWBuildVersion\_SystemInformationID

**Manage statistics**

The auto create statistics is a database option that is set to ON by default. This option creates statistics on an indexed column or a nonindexed column that is used in a JOIN or WHERE clause of a Transact-SQL statement.

You can also use the CREATE STATISTICS statement for columns that do not meet this requirement. The following statement creates statistics named ST\_Address\_City on the City column of the Contact.Address table.

USE AdventureWorks

GO

CREATE STATISTICS ST\_Address\_City ON Contact.Address(City)

When you set the auto update statistics to ON, SQL Server 2012 updates the statistics on indexes when more than 15 percent of the index key values have changed. However, you can manually update the statistics. The following example updates statistics on an index named IX\_Contact\_EmailAddress in the Person.Contact table.

USE AdventureWorks

GO

UPDATE STATISTICS Person.Contact (IX\_Contact\_EmailAddress)

# The Database Engine Tuning Advisor Tool

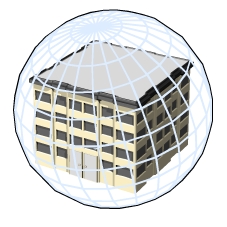
The Database Engine Tuning Advisor tool helps maintain your index. This tool takes a workload file that you specify and applies it to a database or individual tables. It also makes recommendations on which indexes you should use. Remember that the analysis is only as good as the workload file that you provide. A small workload file or a workload file that does not reflect the queries issued against your server will lead to the Database Engine Tuning Advisor tool presenting inappropriate results. If you use the correct workload file, the Database Engine Tuning Advisor tool can be a useful resource in recommending appropriate tuning tips.

### Lesson introduction

One of the latest features included in SQL Server 2008 and SQL Server 2012 is spatial data types and indexes. With the introduction of the geography and the geometry data types, you can work with data that is location-aware.

The ability to work with spatial data can be effectively implemented in conjunction with spatial indexes. Improvements in the performance can be gained when using indexes that work with the spatial data types

# How SQL Server 2012 Uses a Spatial Index



Spatial indexes represent geospatial information by mapping information stored in a table-valued function to the clustered index in the same table. As a result, the prerequisite to creating a spatial index is that a clustered index must already exist.

When you issue a query that uses a spatial index, SQL Server 2012 internally rewrites the query to use the table-valued function, which includes information about the spatial data. SQL Server 2012 then maps the query to the clustered index to return the associated records.

Spatial indexes use a two phases approach when using indexes. First, a primary filter index lookup is applied to provide an approximation of the data that may be required. The filter always returns the likely candidates of objects required from the index. This may include additional values that are known as false-positive records. The purpose of the primary filter is to reduce the number of objects that SQL Server 2012 has to focus on.

The index operation, such as an intersect predicate, is then applied to the reduced set of the objects gained from the primary filter so that the retrieval of the geospatial data through the index works efficiently. At this stage, the false-positive candidate objects determined by the primary filter are removed. The results are then returned.

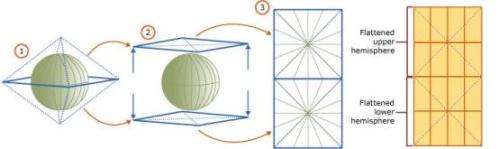
When you create a spatial index, a table-valued function is persisted in the database. The table includes the following four columns.

* A primary key column. This column has a direct mapping to the primary key value in the base table and provides the link between the spatial index table-valued function and the B-Tree representation of the index through a clustered index. The primary key can contain 15 columns, and the combined per-row size of the primary key columns is limited to a maximum of 895 bytes.
* A cell\_id column. This column provides a representation of the four-level grid reference that is used in a spatial index. This is known as a tessellation scheme. One of three tessellation schemes may be stored in this column. The tessellation scheme may not necessarily use a full, four-level grid reference. You can use the following three tessellation schemes.
  + Deep cell optimization. With this scheme only the lowest level cell is stored in the table-valued function as the higher-level grid values are inferred in the lower-level grid reference stored in the cell\_id.
  + Covering cell optimization. In this tessellation scheme, if the object covers all lower grid cells, then only the information of the parent level is recorded. In this instance, a covered cell is counted and recorded in the index, and the cell is not tessellated any further.
  + The cells-per-object rule. Specific to the GEOMETRY data type, this tessellation rule enforces the cells-per-object limit, which determines the maximum number of cells that can be counted for each object, except on level 1. At lower levels, the cells-per-object rule controls the amount of information that can be recorded about the object. This can reduce the storage space of the index but impact the precision. If the object lives outside of the defined area, known as the bounding box, it is given a cellID value of zero.
* A srid column (spatial reference ID column). In this tessellation scheme, you can use only spatial instances with the same SRID when performing operations with SQL Server 2008 spatial data methods on your data. As a result, only geospatial operations can be performed on data that have the same SRID value.
* A cell\_attr column (cell attribute column). The information in this column can determine whether you need to apply a secondary filter or whether the query can be answered by using the primary filter. This provides further optimization of the index. The cell\_attr column defines how objects are stored in the cells in the grid system. The following three values are used to provide this information.

# What Are Tessellation Schemes?

SQL Server 2012 uses two types of tessellation schemes to define the grid-level hierarchy that is used with spatial data. Understanding how the tessellation schemes work will affect how you design and plan the usage of spatial data and spatial indexes. In SQL Server 2012, spatial indexes support the following two tessellation schemes.

**Geography**



This is a scheme for the geography data type. It has a space that is limited to the globe. To break down this space, the geography grid tessellation scheme divides the surface of the earth into its upper and lower hemispheres and then performs the following steps.

1. Projects each hemisphere on the facets of a quadrilateral pyramid
2. Flattens the two pyramids
3. Joins the flattened pyramids to form a non-Euclidean plane

The illustration shows a schematic view of the three-step process. In the pyramids, the dotted lines represent the boundaries of the four facets of each pyramid. Steps 1 and 2 illustrate the earth, the green horizontal line represents the equatorial longitude line and a series of green vertical lines represent several latitude lines. Step 1 shows the pyramids being projected over the two hemispheres. Step 2 shows the pyramids being flattened. Step 3 illustrates the flattened pyramids, after they have been combined to form a non-Euclidean plane, showing a number of projected longitude lines. These projected lines are straightened and vary in length, depending on where they fall on the pyramids.

After the space has been projected onto a flat plane, it is broken down into the four-level grid hierarchy as shown by the fourth image. Different levels can use different grid densities. The illustration shows the plane after it has been broken down into a 4x4 level-1 grid.

As a result, when working with spatial indexes, a spatial index on the geography data type will contain spatial objects for the entire globe.

**Geometry**

|  |  |  |
| --- | --- | --- |
|  |  | Geometry Tessellation  This is the scheme for the geometry data type. The geometry data type deals with a "flat earth" scenario that helps you define spatial mapping for maps that go beyond the earth, for example a map of a stadium or a theme park. However, unlike the geography data type, you must define the area on which the geometry data type will represent. This area is known as a bounding box. The purpose of the bounding box is to limit the spatial objects that SQL Server 2012 works with.  Geometric data can be infinite. In SQL Server 2012, however, a spatial index requires a finite space. To establish a finite space, the geometry grid tessellation scheme uses a bounding box to define the following four coordinates—*x-min***,** *y-min, x-max***, and** *y-max—*which are stored as properties of the spatial index.   * *x-min.* Is the x-coordinate of the lower-left corner of the bounding box. * *y-min.* Is the y-coordinate of the lower-left corner. * *x-max.* Is the x-coordinate of the upper-right corner. * *y-max.* Is the y-coordinate of upper-right corner.   The space outside the bounding box is treated as a single cell that is numbered cellID zero. |
|  |

You can manage spatial indexes by using Transact-SQL statement or SQL Server Management Studio. The following table describes the ways in which you can manage spatial indexes.

**How to Create, Alter, and Drop Spatial Indexes**

[[printer_icon_up](javascript:window.print();) Print this page.](javascript:window.print();)

Use the following syntax to create spatial indexes.

|  |
| --- |
| CREATE SPATIAL INDEX index\_name  ON <object> ( spatial\_column\_name )  {  [ USING <geometry\_grid\_tessellation> ]      WITH ( <bounding\_box>      [ [,] <tesselation\_parameters> [ ,...n ] ]      [ [,] <spatial\_index\_option> [ ,...n ] ] )  |  [ USING <geography\_grid\_tessellation> ]    [ WITH ( [ <tesselation\_parameters> [ ,...n ] ]             [ [,] <spatial\_index\_option> [ ,...n ] ] ) ]  }  [ ON { filegroup\_name | "default" } ] |

The following table describes the clauses used in the syntax.

|  |  |
| --- | --- |
| **Item** | **Description** |
| index\_name | This is the name of the index. Index names must be unique within a table but do not have to be unique within a database. |
| ON <object> ( spatial\_column\_name ) | This clause specifies the object (database, schema, or table) on which the index is to be created and the name of spatial column. Multiple spatial indexes can be created on a geometry or geography column. |
| ***USING*** | This parameter indicates the tessellation scheme for the spatial index. This parameter defaults to the following type-specific value.   * geometry: GEOGRAPHY\_GRID * geography: GEOMETRY\_GRID |
| ***WITH <bounding\_box>*** | This clause specifies a numeric four-tuple that defines the four coordinates of the bounding box: the x-min and y-min coordinates of the lower, left corner, and the x-max and y-max coordinates of the upper right corner. This is specified for the GEOMETRY spatial indexes only. |
| ***<tesselation\_parameters> [ ,...n]*** | The following parameters are specific to geospatial indexes:   * **GRIDS:** This parameter defines the density of the grids at each level of a tessellation scheme. The following are the GRIDS parameters:   + LEVEL\_1 - Specifies the first (top) level grid.   + LEVEL\_2 - Specifies the second-level grid.   + LEVEL\_3 - Specifies the third-level grid.   + LEVEL\_4 - Specifies the fourth-level grid.   Each level has a density defined at each level.   * + LOW - Specifies the lowest possible density for the grid at a given level. LOW equates to 16 cells (a 4x4 grid).   + MEDIUM - Specifies the medium density for the grid at a given level. MEDIUM equates to 64 cells (an 8x8 grid).   + HIGH - Specifies the highest possible density for the grid at a given level. HIGH equates to 256 cells (a 16x16 grid).   When you use level names, you can specify the levels in any order and omit levels. If you use the name for any level, you must use the name of any other level that you specify. If you omit a level, its density defaults to MEDIUM. Therefore, GRIDS can be expressed as follows:  GRIDS = (LOW, LOW, MEDIUM, HIGH),   * **CELLS\_PER\_OBJECT = n:** Specifies the number of tessellation cells per object that can be used for a single spatial object in the index by the tessellation process. n can be any integer between 1 and 8192, inclusive. The default number of cells per object is 16. * ***<spatial\_index\_option> [ ,...n]:*** Standard relational options can also be used with spatial indexes including. See the CREATE INDEX syntax for more information:   + FILLFACTOR = fillfactor   + PAD\_INDEX = { ON | OFF }   + SORT\_IN\_TEMPDB = { ON | OFF }   + IGNORE\_DUP\_KEY = OFF   + STATISTICS\_NORECOMPUTE = { ON | OFF }   + DROP\_EXISTING = { ON | OFF }   + ONLINE = OFF   + ALLOW\_ROW\_LOCKS = { ON | OFF }   + ALLOW\_PAGE\_LOCKS = { ON | OFF }   + MAXDOP = max\_degree\_of\_parallelism * ***ON filegroup\_name|Default:*** This creates the specified index on the specified filegroup. If no location is specified and the table is not partitioned, the index uses the same filegroup as the underlying table. The filegroup must already exist. Default creates the specified index on the default filegroup. |

**How to Create, Alter, and Drop Spatial Indexes**

**Example A. Rebuilding an index**

The following example rebuilds a single index, PK\_Employee\_EmployeeID on the Employee table.

|  |
| --- |
| USE AdventureWorks;  GO  ALTER INDEX PK\_Employee\_EmployeeID ON HumanResources.Employee  REBUILD;  GO |

**Example B. Rebuilding all indexes on a table and specifying options**

The following example specifies the keyword ALL. This rebuilds all indexes associated with the table. The options FILLFACTOR, SORT\_IN\_TEMPDB, and STATISTICS\_NORECOMPUTE are set to ON.

USE AdventureWorks;

GO

ALTER INDEX ALL ON Production.Product

REBUILD WITH (FILLFACTOR = 80, SORT\_IN\_TEMPDB = ON,

STATISTICS\_NORECOMPUTE = ON);

GO

**Example C. Setting options on an index**

The following example sets several options to ON including STATISTICS\_NORECOMPUTE, IGNORE\_DUP\_KEY, and ALLOW\_PAGE\_LOCKS on the index AK\_SalesOrderHeader\_SalesOrderNumber.

USE AdventureWorks;

GO

ALTER INDEX AK\_SalesOrderHeader\_SalesOrderNumber ON

Sales.SalesOrderHeader

SET (

STATISTICS\_NORECOMPUTE = ON,

IGNORE\_DUP\_KEY = ON,

ALLOW\_PAGE\_LOCKS = ON

) ;

GO

**Example D. Disabling an index**

The following example disables a nonclustered index named IX\_Employee\_ManagerID on the Employee table.

USE AdventureWorks;

GO

ALTER INDEX IX\_Employee\_ManagerID ON HumanResources.Employee

DISABLE ;

GO

**Example E. Changing the compression setting of an index**

The following example rebuilds an index on a nonpartitioned table and sets page level compression.

ALTER INDEX IX\_INDEX1

ON T1

REBUILD

WITH ( DATA\_COMPRESSION = PAGE )

GO

**How to Create, Alter, and Drop Spatial Indexes Using SSMS**

**Example A. To create a spatial index in SQL Server Management Studio, perform the following steps:**

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine and then expand that instance.
2. Expand the **Databases** node, expand the database that contains the table with the specified index, and then expand the **Tables** node.
3. Expand the table for which you want to create the index.
4. Right-click **Indexes**, and then select **New Index**.
5. In the **Index name** field, specify a name for the index.
6. In the **Index type** list, select **Spatial**.
7. To specify the spatial column that you want to index, click **Add**.
8. In the **Select Columns from** <table name> dialog box, select a column of type **geometry** or **geography**, and then click **OK**. Any other spatial columns then become uneditable. If you want to select a different spatial column, you must first clear the currently selected column.
9. In the **Index key columns** grid, verify the column that you selected.
10. In the **Index Properties** dialog box, in the **Select a page** pane, click **Spatial**.
11. On the **Spatial** page, specify the values that you want to use for the spatial properties of the index.

When creating an index on a **geometry** type column, you must specify the (X-min,Y-min) and (X-max,Y-max) coordinates of the bounding box. For an index on a **geography** type column, the bounding-box fields become read-only after you specify the **Geography grid** tessellation scheme. This is because geography grid tessellation does not use a bounding box.

Optionally, you can specify non-default values for the **Cells Per Object** field and for the grid density at any level of the tessellation scheme. The default number of cells per object is 16, and the default grid density is **Medium**.

1. Click **OK**.

**Example B. Rebuilding all indexes on a table and specifying options**

The following example specifies the keyword ALL. This rebuilds all indexes associated with the table. The options FILLFACTOR, SORT\_IN\_TEMPDB, and STATISTICS\_NORECOMPUTE are set to ON.

USE AdventureWorks;

GO

ALTER INDEX ALL ON Production.Product

REBUILD WITH (FILLFACTOR = 80, SORT\_IN\_TEMPDB = ON,

STATISTICS\_NORECOMPUTE = ON);

GO

**Example C. Setting options on an index**

The following example sets several options to ON including STATISTICS\_NORECOMPUTE, IGNORE\_DUP\_KEY, and ALLOW\_PAGE\_LOCKS on the index AK\_SalesOrderHeader\_SalesOrderNumber.

USE AdventureWorks;

GO

ALTER INDEX AK\_SalesOrderHeader\_SalesOrderNumber ON

Sales.SalesOrderHeader

SET (

STATISTICS\_NORECOMPUTE = ON,

IGNORE\_DUP\_KEY = ON,

ALLOW\_PAGE\_LOCKS = ON

) ;

GO

**Example D. Disabling an index**

The following example disables a nonclustered index named IX\_Employee\_ManagerID on the Employee table.

USE AdventureWorks;

GO

ALTER INDEX IX\_Employee\_ManagerID ON HumanResources.Employee

DISABLE ;

GO

**Example E. Changing the compression setting of an index**

The following example rebuilds an index on a nonpartitioned table and sets page level compression.

ALTER INDEX IX\_INDEX1

ON T1

REBUILD

WITH ( DATA\_COMPRESSION = PAGE )

GO

# How to Retrieve Spatial Index Information

You can use standard metadata functions such as sp\_help, sys.indexes, DBCC ShowContig, DBCC Show Statistics, sys.dm\_db\_index\_usage\_stats, and SQL Server Management Studio standard reports to return information about any index. However, you can use the following metadata queries to return information specific to spatial indexes.

**sys.spatial\_indexes**

When retrieving metadata from the sys.spatial\_indexes, information is inherited from the sys.indexes catalog view. However, the following three columns are specific to spatial data.

* *spatial\_index\_type.* This column displays one of two values:
  + 1 - which defines a geometric spatial index
  + 2 - which defines a geographic spatial index.
* *spatial\_index\_type\_desc.* Provides a descriptive value of the spatial\_index\_type column which can be a value of geometry or geography.
* *tessellation\_scheme.* Describes the tessellation scheme that is used for the spatial index. This can be one of two values; GEOMETRY\_GRID or GEOGRAPHY\_GRID

The following example returns information about spatial indexes within the AdventureWorks database.

USE AdventureWorks

GO

SELECT \* FROM sys.spatial\_indexes

**sys.spatial\_index\_tessellations**

This catalog view provides specific information about the tessellation scheme and parameters of each of the spatial indexes.

The following columns are returned from the sys.spatial\_index\_tessellations query.

* *object\_id.* This is the ID of the object on which the index is defined.
* *index\_id.* This is the ID of the spatial index in which the indexed column is defined.
* *tessellation\_scheme.* This is the name of the tessellation scheme. It can either be GEOMETRY\_GRID or GEOGRAPHY\_GRID.
* *bounding\_box\_xmin.* This is the X-coordinate of the lower-left corner of the bounding box. NULL indicates GEOGRAPHY\_GRID is being used.
* *bounding\_box\_ymin.* This is the Y-coordinate of the lower-left corner of the bounding box. NULL indicates GEOGRAPHY\_GRID is being used.
* *bounding\_box\_xmax.* This is the X-coordinate of the upper-right corner of the bounding box. NULL indicates GEOGRAPHY\_GRID is being used.
* *bounding\_box\_ymax.* This is the Y-coordinate of the upper-right corner of the bounding box. NULL indicates GEOGRAPHY\_GRID is being used.
* *level\_1\_grid.* This is the grid density for the top-level grid. If tessellation\_scheme is GEOMETRY\_GRID or GEOGRAPHY\_GRID, one of the following grid density can be applied.
  + 16 = 4 by 4 grid (LOW)
  + 64 = 8 by 8 grid (MEDIUM)
  + 256 = 16 by 16 grid (HIGH)

These settings also apply to level\_2\_grid, level\_3\_grid, and level\_4\_grid columns.

* *level\_1\_grid\_desc.* This provides a descriptive version of the level\_1\_grid value and includes the value LOW, MEDIUM, or HIGH. These settings also apply to level\_2\_grid\_desc, level\_3\_grid\_desc, and level\_4\_grid\_desc.
* *cells\_per\_object.* This defines the number of cells per spatial object.

The following example returns information about spatial indexes tessellation in the AdventureWorks database.

USE AdventureWorks

GO

SELECT \* FROM sys.spatial\_index\_tessellations

**SQL Server Management Studio**

To view properties of an index by using SQL Server Management Studio, perform the following steps.

1. In **Object Explorer**, connect to an instance of the SQL Server Database Engine, and then expand that instance.
2. Expand the **Databases** node, expand the database that includes the table containing the index, and then expand the **Tables** node.
3. Expand the table to which the index belongs and then expand the **Indexes** node.
4. Right-click the index, and then click **Properties**. The Index Properties dialog box opens.

In the **Select a page** pane, click the page for the properties that you want to view.

# Best Practices



For best results, you can abide by the following best practices when creating and optimizing indexes in SQL Server 2012.

* Create indexes on columns that join tables, including primary and foreign keys to improve query performance.
* Place clustered indexes on primary key columns that consist of integer data with an Identity constraint to keep the size of the index small.
* Use nonclustered indexes to facilitate search arguments within queries.
* Use composite nonclustered indexes to cover the most common queries.
* Use the CREATE STATISTICS statement on columns that are not defined first in a composite index to generate more efficient execution plans.
* Ensure that the table data and the index data are backed up at the same time, if your indexes are on a separate filegroups to the table data.
* Drop any indexes that are no longer used. Use the Database Engine Tuning Advisor tool to help identify unused indexes.
* Use the FILLFACTOR option to optimize performance by reducing page splits.
* Use the DBCC SHOWCONTIG statement to assess the level of fragmentation of databases.
* Rebuild indexes if fragmentation is greater than 30 percent. Reorganize indexes if fragmentation is below 30 percent or there is little time to rebuild indexes.
* Use the DROP\_EXISTING option to rebuild indexes quickly.
* Remember to recompile stored procedures to take advantage of indexes that have been rebuilt.
* Use spatial indexes to facilitate searches on geospatial data.
* Ensure a primary key exists on the table on which a spatial index is defined.
* Monitor queries that retrieve spatial data to ensure that the spatial index is being used. If not, add index hints to queries to force the use of a spatial index.

**TEST**

What is the maximum size of a page in a heap?

Select the one best answer.



8 bytes.

That is not correct. The correct response is: 8 KB. The maximum size of a page in a heap is 8 KB.



8 KB.

That is correct. The maximum size of a page in a heap is 8 KB.



8 MB.

That is not correct. The correct response is: 8 KB. The maximum size of a page in a heap is 8 KB.



8 GB.

You have a table that stores location data in a column that uses a geometry data type. You query this column frequently. Which index should you create to improve query performance against the column?

Select the one best answer.



Heap.

That is not correct. The correct response is: Spatial index. To improve query performance against the column, you should create a spatial index.



Nonclustered index.

That is not correct. The correct response is: Spatial index. To improve query performance against the column, you should create a spatial index.



Spatial index.

That is correct. To improve query performance against the column, you should create a spatial index.



Clustered index.

Which index physically reorder and incorporate data within its index structure?

Select the one best answer.



Clustered index.

That is correct. A clustered index physically reorders and incorporates data within its index structure.



Heap.

That is not correct. The correct response is: Clustered index. This index physically reorders and incorporates data within its index structure.



Nonclustered index.

That is not correct. The correct response is: Clustered index. This index physically reorders and incorporates data within its index structure.



Spatial index.

How many nonclustered indexes can be defined on a table or view?

Select the one best answer.



1.

That is not correct. The correct response is: 249. You can define 249 nonclustered indexes on a table or view.



249.

That is correct. You can define 249 nonclustered indexes on a table or view.



1024.

That is not correct. The correct response is: 249. You can define 249 nonclustered indexes on a table or view.



250.

You are a database administrator for Adventure Works. The database for Adventure Works includes a table in the Sales schema named SalesOrderDetails. In this table, a column named SalesOrderID contains values of the integer data type with unique numbers. This column provides a reference to a sales order. Queries against this table typically display the results ordered by the SalesOrderID column. Which type of index will help improve the performance of such queries?

Select the one best answer.



Clustered index on the SalesOrderID column.

That is correct. A clustered index can physically reorder the data in the SalesOrderID column in ascending order to improve the performance of queries.



Nonclustered index on the SalesOrderID column.

That is not correct. The correct response is: Clustered index on the SalesOrderID column. A clustered index can physically reorder the data in the SalesOrderID column in ascending order to improve the performance of queries.



A geospatial index.

That is not correct. The correct response is: Clustered index on the SalesOrderID column. A clustered index can physically reorder the data in the SalesOrderID column in ascending order to improve the performance of queries.



A composite index.

You are a database developer for Adventure Works. You need to define a clustered index on the ContactID column of the Person.Contact table in the Sales database. The index should be named CL\_ContactID and sort the values in descending order. Which statement should you use to define the clustered index?

Select the one best answer.



USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID)

That is not correct. This statement creates an index, however, in an ascending order. The correct answer is: USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID DESC). This statement is used to define the clustered index.



USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID DESC)

That is correct. USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID DESC) is used to define the clustered index.



USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON (ContactID DESC) OF Person.Contact

That is not correct. This statement generates a syntax error. The correct answer is: USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID DESC). This statement is used to define the clustered index.



USE Adventureworks GO CREATE NON CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID)

That is not correct. This statement creates a non clustered index. The correct answer is: USE Adventureworks GO CREATE CLUSTERED INDEX CL\_ContactID ON Person.Contact(ContactID DESC). This statement is used to define the clustered index.

You are a database developer for Adventure Works. You need to create a new unique multi-column, nonclustered index named NCL\_PC\_Add1\_City on the Person.Address table in the Adventureworks database. The index must include the Address1, City, and PostalCode columns. You must ensure that index statistics are generated on the PostalCode column. Which statement should you use to accomplish this task?

Select the one best answer.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (City, PostalCode, Address1); GO

That is not correct. Although this statement creates the multicolumn index, it generates statistics on the City column only. The correct answer is: USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (PostalCode, Address1, City); GO. This statement ensures that the index statistics are generated on the PostalCode column.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (Address1, City, PostalCode); GO

That is not correct. Although this statement creates the multicolumn index, it generates statistics on the Address1 column only. The correct answer is: USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (PostalCode, Address1, City); GO. This statement ensures that the index statistics are generated on the PostalCode column.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (PostalCode, Address1, City); GO

That is correct. USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (PostalCode, Address1, City); GO ensures that the index statistics are generated on the PostalCode column.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_PC\_Add1\_City ON Person.Contact (PostalCode); GO

You have developed the Adventure Works database. 90 percent of the queries against the Production.Product table in the Adventureworks database is for products that do not have a discontinued date in the DiscontinuedDate column. You need to create a nonclustered index named NCL\_Name\_StandardCost with the Name and StandardCost column that will include data from the current products. Which statement meets this objective?

Select the one best answer.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); WHERE DiscontinuedDate IS NOT NULL GO

That is not correct. This statement creates an index on all records that have a date in the DiscontinueDate column. The correct answer is: USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO. This statement can create a nonclustered index named NCL\_Name\_StandardCost with the Name and StandardCost column that will include data from the current products.



USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO

That is the correct. USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO can create a nonclustered index named NCL\_Name\_StandardCost with the Name and StandardCost column that will include data from the current products.



USE Adventureworks GO CREATE CLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO

That is not correct. This statement creates a clustered index. The correct answer is: USE Adventureworks GO CREATE NONCLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO. This statement can create a nonclustered index named NCL\_Name\_StandardCost with the Name and StandardCost column that will include data from the current products.



USE Adventureworks GO CREATE CLUSTERED INDEX NCL\_Name\_StandardCost ON Production.Product (Name, StandardCost); GO

Which index feature can help you incorporate more than 16 columns at the leaf level of an index?

Select the one best answer.



UNIQUE.

That is not correct. The correct response is: INCLUDE. This option helps you incorporate more than 16 columns within the leaf level of an index.



INCLUDE.

That is correct. The INCLUDE option helps you incorporate more than 16 columns within the leaf level of an index.



PRIMARYKEY.

That is not correct. The correct response is: INCLUDE. This option helps you incorporate more than 16 columns within the leaf level of an index



VIEW.

A new row is inserted into a table. However, the row does not fit into a data page. As a result, the data page is divided into two, with 50 percent of the data in one page and 50 percent in the other page. Which term is used to describe this process?

Select the one best answer.



Row split.

That is not correct. The correct answer is: Page split. This option divides the data page into two.



Table split.

That is not correct. The correct answer is: Page split. This option divides the data page into two.



Table scan.

That is not correct. The correct answer is: Page split. This option divides the data page into two.



Page split.

Which index option incorporates free space into the non-leaf level of an index?

Select the one best answer.



FREE\_SPACE.

That is not correct. This option does not exist. The correct answer is: PAD\_INDEX. This option incorporates free space into the non-leaf level of an index.



PAD\_INDEX.

That is correct. PAD\_INDEX option incorporates free space into the non-leaf level of an index.



INCLUDE.

That is not correct. This option is not used in this context. The correct answer is: PAD\_INDEX. This option incorporates free space into the non-leaf level of an index.



FILL\_FACTOR.

You are a database developer for Wingtip Toys. You have determined that there is excessive page splits during the peak periods that the table Sales.SalesOrderDetail is used. There is a nonclustered index named NCL\_SalesOrderID on the SalesOrderID column. To reduce the effects of page splits, you determine that you require 40 percent free space in the table data. Which statement should you use?

Select the one best answer.



USE AdventureWorks; GO ALTER INDEX ALL ON Sales.SalesOrderDetail REBUILD; GO

That is not correct. This statement rebuilds the index on the Sales.SalesOrderDetails table. The correct answer is: USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD WITH (FILLFACTOR = 60); GO. This statement leaves 40% free space on the index.



USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD WITH (FILLFACTOR = 60); GO

That is correct. USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD WITH (FILLFACTOR = 60); GO leaves 40% free space on the index.



USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD GO

That is not correct. This statement rebuilds the index with no free space incorporated in the index. The correct answer is: USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD WITH (FILLFACTOR = 60); GO. This statement leaves 40% free space on the index.



USE AdventureWorks; GO ALTER INDEX NCL\_SalesOrderID ON Sales.SalesOrderDetail REBUILD WITH (FILLFACTOR = 40); GO

With respect to index statistics, which is the correct definition of selectivity?

Select the one best answer.



The average percentage of duplicate rows within the index.

That is not correct. This is the definition of density. The correct answer is: A percentage of rows in a table that are accessed or returned by a query.



A percentage of rows in a table that are accessed or returned by a query.

That is correct. A percentage of rows in a table that are accessed or returned by a query.



The amount of data that exists in all tables in a database.

That is not correct. The correct answer is: A percentage of rows in a table that are accessed or returned by a query.



The amount of data that exists for a range of values.

You are a database developer for Contoso Ltd. You need to rebuild a clustered index named CL\_ContactID on the ContactID column of the Person.Contact table. The index must be compressed at page level. Which statement should you use to build the clustered index?

Select the one best answer.



INDEX ALTER CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = PAGE ) GO

That is not correct. A syntax error will be returned. You can alter an index to add data compression. The correct answer is: ALTER INDEX CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = PAGE ) GO. This statement compresses the index at page level.



ALTER INDEX CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = PAGE ) GO

That is correct. ALTER INDEX CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = PAGE ) GO compresses the index at page level.



ALTER INDEX CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = ROW ) GO

That is not correct. This statement adds row level data compression to the index. The correct answer is: ALTER INDEX CL\_ContactID ON Person.Contact REBUILD WITH ( DATA\_COMPRESSION = PAGE ) GO. This statement compresses the index at page level.



ALTER INDEX CL\_ContactID ON Person.Contact REBUILD GO

What type of a file can you use as a workload file within the Database Engine Tuning Advisor tool?

Select the one best answer.



.trc.

That is not correct. This is one of the files that you can use as a workload file in the Database Engine Tuning Advisor tool. The correct answer is: All of the above.



.xml.

That is not correct. This is one of the files that you can use as a workload file in the Database Engine Tuning Advisor tool. The correct answer is: All of the above.



.sql.

That is not correct. This is one of the files that you can use as a workload file in the Database Engine Tuning Advisor tool. The correct answer is: All of the above.



All of the above.

Which is an important prerequisite for using spatial indexes on a table?

Select the one best answer.



A nonclustered index must already exist on the table.

That is not correct. Nonclustered indexes are not a pre requisite for creating spatial indexes. The correct answer is: A clustered index must already exist on the table. This is an important prerequisite for using spatial indexes on a table.



A table must contain one column with a CHAR data type defined.

That is not correct. One column with a CHAR data type defined is not a pre-requisite for creating spatial indexes. The correct answer is: A clustered index must already exist on the table. This is an important prerequisite for using spatial indexes on a table.



A clustered index must already exist on the table.

That is correct. A clustered index must already exist on the table is an important prerequisite for using spatial indexes on a table.



All of the above.

You are a database developer for Adventure Works. The database contains a table name Person.Address and you have added a column named Reference that uses the Geography data type. You need to create a spatial index named SIN\_Reference on this column with the highest level of granularity. Which statement should you use?

Select the one best answer.



CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = MEDIUM, LEVEL\_2 = MEDIUM ) );

That is not correct. This statement creates the spatial index without the highest level of granularity. The correct answer is: CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = HIGH, LEVEL\_2 = HIGH ) );. This statement can create a spatial index named SIN\_Reference on the column with the highest level of granularity.



CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = HIGH, LEVEL\_2 = HIGH ) );

That is correct. CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = HIGH, LEVEL\_2 = HIGH ) ); can create a spatial index named SIN\_Reference on the column with the highest level of granularity.



CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = LOW, LEVEL\_2 = LOW ) );

That is not correct. This statement creates the spatial index without the highest level of granularity. The correct answer is: CREATE SPATIAL INDEX SIN\_Reference ON Person.Address(Reference) WITH ( GRIDS = ( LEVEL\_3 = HIGH, LEVEL\_2 = HIGH ) );. This statement can create a spatial index named SIN\_Reference on the column with the highest level of granularity.



CREATE SPATIAL INDEX SIN\_Reference ON (Reference) WITH ( GRIDS = ( LEVEL\_3 = HIGH, LEVEL\_2 = HIGH ) );

When defining a spatial index on a GEOMETRY column you must define an area on which the spatial index will be based. What is the name of this area?

Select the one best answer.



Level 0.

That is not correct. This refers to the area beyond the bounding box. The correct answer is: Limited index.



Limited index.

That is not correct. This reference does not exist. The correct answer is: Limited index.



Spatial area.

That is not correct. This reference does not exist. The correct answer is: Limited index.



Bounding box.

You need to run a metadata query that provides specific information about the tessellation scheme and parameters used on spatial indexes. Which metadata query should you use?

Select the one best answer.



Sys.spatial\_indexes.

That is not correct. This statement does not return tessellation information about a spatial index. The correct answer is: sys.spatial\_index\_tessellations. This statement returns tessellation information about a spatial index.



Sys.indexes.

That is not correct. This statement does not return tessellation information about a spatial index. The correct answer is: sys.spatial\_index\_tessellations. This statement returns tessellation information about a spatial index.



sys.spatial\_index\_tessellations.

That is correct. sys.spatial\_index\_tessellations returns tessellation information about a spatial index



Sp\_help.