**Clustered Index Design Guidelines**

**SQL Server 2008 R2**

[Other Versions](javascript:;)

http://i.technet.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms190639(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms190639(d=printer,v=sql.90).aspx)

Clustered indexes sort and store the data rows in the table based on their key values. There can only be one clustered index per table, because the data rows themselves can only be sorted in one order. For more information about clustered index architecture, see [Clustered Index Structures](http://technet.microsoft.com/en-us/library/ms177443(v=sql.105).aspx)

With few exceptions, every table should have a clustered index defined on the column, or columns, that offer the following:

* Can be used for frequently used queries.
* Provide a high degree of uniqueness.

|  |
| --- |
| **NoteNote** |
| When you create a PRIMARY KEY constraint, a unique index on the column, or columns, is automatically created. By default, this index is clustered; however, you can specify a nonclustered index when you create the constraint. |

* Can be used in range queries.

If the clustered index is not created with the UNIQUE property, the Database Engine automatically adds a 4-byte **uniqueifier** column to the table. When it is required, the Database Engine automatically adds a **uniqueifier** value to a row to make each key unique. This column and its values are used internally and cannot be seen or accessed by users.

[Query Considerations](javascript:void(0))

Before you create clustered indexes, understand how your data will be accessed. Consider using a clustered index for queries that do the following:

* Return a range of values by using operators such as BETWEEN, >, >=, <, and <=.

After the row with the first value is found by using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent. For example, if a query retrieves records between a range of sales order numbers, a clustered index on the column **SalesOrderNumber** can quickly locate the row that contains the starting sales order number, and then retrieve all successive rows in the table until the last sales order number is reached.

* Return large result sets.
* Use JOIN clauses; typically these are foreign key columns.
* Use ORDER BY, or GROUP BY clauses.

An index on the columns specified in the ORDER BY or GROUP BY clause may remove the need for the Database Engine to sort the data, because the rows are already sorted. This improves query performance.

[Column Considerations](javascript:void(0))

Generally, you should define the clustered index key with as few columns as possible. Consider columns that have one or more of the following attributes:

* Are unique or contain many distinct values

For example, an employee ID uniquely identifies employees. A clustered index or PRIMARY KEY constraint on the **EmployeeID** column would improve the performance of queries that search for employee information based on the employee ID number. Alternatively, a clustered index could be created on **LastName**, **FirstName**, **MiddleName** because employee records are frequently grouped and queried in this way, and the combination of these columns would still provide a high degree of difference.

* Are accessed sequentially

For example, a product ID uniquely identifies products in the **Production.Product** table in the AdventureWorks2008R2 database. Queries in which a sequential search is specified, such as WHERE ProductID BETWEEN 980 and 999, would benefit from a clustered index on **ProductID**. This is because the rows would be stored in sorted order on that key column.

* Defined as IDENTITY because the column is guaranteed to be unique within the table.
* Used frequently to sort the data retrieved from a table.

It can be a good idea to cluster, that is physically sort, the table on that column to save the cost of a sort operation every time the column is queried.

Clustered indexes are not a good choice for the following attributes:

* Columns that undergo frequent changes

This causes in the whole row to move, because the Database Engine must keep the data values of a row in physical order. This is an important consideration in high-volume transaction processing systems in which data is typically volatile.

* Wide keys

Wide keys are a composite of several columns or several large-size columns. The key values from the clustered index are used by all nonclustered indexes as lookup keys. Any nonclustered indexes defined on the same table will be significantly larger because the nonclustered index entries contain the clustering key and also the key columns defined for that nonclustered index.

[Index Options](javascript:void(0))

There are several index options that can be specified when you create a clustered index. Because clustered indexes are typically quite large, you should give special consideration to the following options:

* SORT\_IN\_TEMPDB
* DROP\_EXISTING
* FILLFACTOR
* ONLINE

For more information, see [Setting Index Options](http://technet.microsoft.com/en-us/library/ms188677(v=sql.105).aspx).

**Setting Index Options**

**SQL Server 2008 R2**

[Other Versions](javascript:;)



* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms188677(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms188677(d=printer,v=sql.90).aspx)
* [SQL Server 2012](http://technet.microsoft.com/en-us/library/ms188677(d=printer,v=sql.110).aspx)

When you design, create, or modify an index, there are several index options to consider. These options can be specified when you first create an index or when you rebuild an index. Additionally, you can set some index options at any time by using the SET clause of the ALTER INDEX statement.

|  |  |  |  |
| --- | --- | --- | --- |
| **Index option** | **Description** | **Setting stored in metadata** | **Related topic** |
| PAD\_INDEX | Sets the percentage of free space in the intermediate level pages during index creation. | Yes | [Fill Factor](http://technet.microsoft.com/en-us/library/ms177459(v=sql.105).aspx) |
| FILLFACTOR | Sets the percentage of free space in the leaf level of each index page during index creation. | Yes | [Fill Factor](http://technet.microsoft.com/en-us/library/ms177459(v=sql.105).aspx) |
| SORT\_IN\_TEMPDB | Determines where the intermediate sort results, generated during index creation, are stored.  When ON, the sort results are stored in [tempdb](http://technet.microsoft.com/en-us/library/ms190768(v=sql.105).aspx). When OFF, the sort results are stored in the filegroup or partition scheme in which the resulting index is stored.  Note**Note**  If a sort operation is not required, or if the sort can be performed in memory, SORT\_IN\_TEMPDB is ignored. | No | [tempdb and Index Creation](http://technet.microsoft.com/en-us/library/ms188281(v=sql.105).aspx) |
| IGNORE\_DUP\_KEY | Specifies the error response when an insert operation attempts to insert duplicate key values into a unique index. The IGNORE\_DUP\_KEY option applies only to insert operations after the index is created or rebuilt. The default is OFF. | Yes | [CREATE INDEX](http://technet.microsoft.com/en-us/library/ms188783(v=sql.105).aspx) |
| STATISTICS\_NORECOMPUTE | Specifies whether out-of-date index statistics should be automatically recomputed. | Yes | [Index Statistics](http://technet.microsoft.com/en-us/library/ms190397(v=sql.105).aspx) |
| DROP\_EXISTING | Indicates the existing index should be dropped and recreated. | No | [Reorganizing and Rebuilding Indexes](http://technet.microsoft.com/en-us/library/ms189858(v=sql.105).aspx) |
| ONLINE | Determines whether concurrent user access to the underlying table or clustered index data and any associated nonclustered indexes is allowed during index operations.  Note**Note**  Online index operations are available only in SQL Server Enterprise, Developer, and Evaluation editions. | No | [Performing Index Operations Online](http://technet.microsoft.com/en-us/library/ms177442(v=sql.105).aspx). |
| ALLOW\_ROW\_LOCKS | Determines whether row locks are used in accessing index data. | Yes | [CREATE INDEX](http://technet.microsoft.com/en-us/library/ms188783(v=sql.105).aspx)  [Customizing Locking for an Index](http://technet.microsoft.com/en-us/library/ms189076(v=sql.105).aspx) |
| ALLOW\_PAGE\_LOCKS | Determines whether page locks are used in accessing index data. | Yes | [CREATE INDEX](http://technet.microsoft.com/en-us/library/ms188783(v=sql.105).aspx)  [Customizing Locking for an Index](http://technet.microsoft.com/en-us/library/ms189076(v=sql.105).aspx) |
| MAXDOP | Sets the maximum number of processors the query processor can use to execute a single index statement. Fewer processors may be used depending on the current system workload.  Note**Note**  Parallel index operations are available only in SQL Server Enterprise, Developer, and Evaluation editions. | No | [Modifying Indexes](http://technet.microsoft.com/en-us/library/ms189329(v=sql.105).aspx) |
| DATA\_COMPRESSION | Specifies the data compression option for the specified table, partition number or range of partitions. Options are NONE, ROW, and PAGE. | Yes | [Creating Compressed Tables and Indexes](http://technet.microsoft.com/en-us/library/cc280449(v=sql.105).aspx) |

To set options on an index

* [ALTER INDEX (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms188388(v=sql.105).aspx)

[Setting Options Without Rebuilding](javascript:void(0))

By using the SET clause in the ALTER INDEX statement, you can set the following index options without rebuilding the index:

* ALLOW\_PAGE\_LOCKS
* ALLOW\_ROW\_LOCKS
* IGNORE\_DUP\_KEY
* STATISTICS\_NORECOMPUTE

These options are immediately applied to the index. Other index options, such as FILLFACTOR and ONLINE, can be specified only when an index is created or rebuilt.

[Viewing Index Option Settings](javascript:void(0))

Not all index options values are stored in metadata. Those values that are stored in metadata can be viewed in the appropriate catalog views. To examine the current option settings on existing indexes, use the [sys.indexes](http://technet.microsoft.com/en-us/library/ms173760(v=sql.105).aspx) catalog view . To examine the current value for STATISTICS\_NORECOMPUTE, use the [sys.stats](http://technet.microsoft.com/en-us/library/ms177623(v=sql.105).aspx) catalog view. For more information, see [Viewing Index Information](http://technet.microsoft.com/en-us/library/ms188280(v=sql.105).aspx).

[Examples](javascript:void(0))

The following example sets the ALLOW\_ROW\_LOCKS and IGNORE\_DUP\_KEY options for the AK\_Product\_ProductNumber index on the Production.Product table.

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_a0b8db25-56c6-4a39-a3d9-f41ecb618f66');" \o "Copy to clipboard.)

USE AdventureWorks2008R2;

GO

--Verify the current values for these options.

SELECT allow\_row\_locks, ignore\_dup\_key

FROM sys.indexes

WHERE name = N'AK\_Product\_ProductNumber';

GO

--Set the ALLOW\_ROW\_LOCKS option OFF and the IGNORE\_DUP\_KEY option ON.

ALTER INDEX AK\_Product\_ProductNumber

ON Production.Product

SET (ALLOW\_ROW\_LOCKS = OFF, IGNORE\_DUP\_KEY = ON);

GO

--Verify the new values for these options.

SELECT allow\_row\_locks, ignore\_dup\_key

FROM sys.indexes

WHERE name = N'AK\_Product\_ProductNumber';

GO