**Create Filtered Indexes**

**SQL Server 2012**

[Other Versions](javascript:;)

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

* [SQL Server 2008 R2](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.105).aspx)
* [SQL Server 2008](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.100).aspx)

This topic describes how to create a filtered index in SQL Server 2012 by using SQL Server Management Studio or Transact-SQL. A filtered index is an optimized nonclustered index especially suited to cover queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in the table. A well-designed filtered index can improve query performance as well as reduce index maintenance and storage costs compared with full-table indexes.

Filtered indexes can provide the following advantages over full-table indexes:

* Improved query performance and plan quality

A well-designed filtered index improves query performance and execution plan quality because it is smaller than a full-table nonclustered index and has filtered statistics. The filtered statistics are more accurate than full-table statistics because they cover only the rows in the filtered index.

* Reduced index maintenance costs

An index is maintained only when data manipulation language (DML) statements affect the data in the index. A filtered index reduces index maintenance costs compared with a full-table nonclustered index because it is smaller and is only maintained when the data in the index is changed. It is possible to have a large number of filtered indexes, especially when they contain data that is changed infrequently. Similarly, if a filtered index contains only the frequently modified data, the smaller size of the index reduces the cost of updating the statistics.

* Reduced index storage costs

Creating a filtered index can reduce disk storage for nonclustered indexes when a full-table index is not necessary. You can replace a full-table nonclustered index with multiple filtered indexes without significantly increasing the storage requirements.

In This Topic

* Before you begin:

[Design Considerations](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.110).aspx#Design)

[Limitations and Restrictions](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.110).aspx#Restrictions)

[Security](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.110).aspx#Security)

* To create a filtered index, using:

[SQL Server Management Studio](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.110).aspx#SSMSProcedure)

[Transact-SQL](http://msdn.microsoft.com/en-us/library/cc280372(d=printer,v=sql.110).aspx#TsqlProcedure)

[Before You Begin](javascript:void(0))

**Design Considerations**

* When a column only has 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. The resulting index will be smaller and cost less to maintain than a full-table nonclustered index defined on the same key columns.
* When a table has heterogeneous data rows, you can create a filtered index for one or more categories of data. This can improve the performance of queries on these data rows by narrowing the focus of a query to a specific area of the table. Again, the resulting index will be smaller and cost less to maintain than a full-table nonclustered index.

**Limitations and Restrictions**

* 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 query optimizer considers a filtered index for a query that selects from a view if the query results will be correct.
* Filtered indexes have the following advantages over indexed views:
  + Reduced index maintenance costs. For example, the query processor uses fewer CPU resources to update a filtered index than an indexed view.
  + Improved plan quality. For example, during query compilation, the query optimizer considers using a filtered index in more situations than the equivalent indexed view.
  + Online index rebuilds. You can rebuild filtered indexes while they are available for queries. Online index rebuilds are not supported for indexed views. For more information, see the REBUILD option for [ALTER INDEX (Transact-SQL)](http://msdn.microsoft.com/en-us/library/ms188388.aspx).
  + Non-unique indexes. Filtered indexes can be non-unique, whereas indexed views must be unique.
* Filtered indexes are defined on one table and only support simple comparison operators. If you need a filter expression that references multiple tables or has complex logic, you should create a view.
* A column in the filtered index expression does not need to be a key or included column in the filtered index definition if the filtered index expression is equivalent to the query predicate and the query does not return the column in the filtered index expression with the query results.
* A column in the filtered index expression should be a key or included column in the filtered index definition if the query predicate uses the column in a comparison that is not equivalent to the filtered index expression.
* A column in the filtered index expression should be a key or included column in the filtered index definition if the column is in the query result set.
* The clustered index key of the table does not need to be a key or included column in the filtered index definition. The clustered index key is automatically included in all nonclustered indexes, including filtered indexes.
* If the comparison operator specified in the filtered index expression of the filtered index results in an implicit or explicit data conversion, an error will occur if the conversion occurs on the left side of a comparison operator. A solution is to write the filtered index expression with the data conversion operator (CAST or CONVERT) on the right side of the comparison operator.

**Security**

**Permissions**

Requires ALTER permission on the table or view. User must be a member of the **sysadmin** fixed server role or the **db\_ddladmin** and **db\_owner** fixed database roles. To modify the filtered index expression, use CREATE INDEX WITH DROP\_EXISTING.

[Using SQL Server Management Studio](javascript:void(0))

**To create a filtered index**

1. In Object Explorer, click the plus sign to expand the database that contains the table on which you want to create a filtered index.
2. Click the plus sign to expand the Tables folder.
3. Click the plus sign to expand the table on which you want to create a filtered index.
4. Right-click the Indexes folder, point to New Index, and select Non-Clustered Index….
5. In the New Index dialog box, on the General page, enter the name of the new index in the Index name box.
6. Under Index key columns, click Add….
7. In the Select Columns from table\_name dialog box, select the check box or check boxes of the table column or columns to be added to the unique index.
8. Click OK.
9. On the Filter page, under Filter Expression, enter SQL expression that you’ll use to create the filtered index.
10. Click OK.

[Using Transact-SQL](javascript:void(0))

**To create a filtered index**

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

USE AdventureWorks2012;

GO

-- Looks for an existing filtered index named "FIBillOfMaterialsWithEndDate"

-- and deletes it from the table Production.BillOfMaterials if found.

IF EXISTS (SELECT name FROM sys.indexes

    WHERE name = N'FIBillOfMaterialsWithEndDate'

    AND object\_id = OBJECT\_ID (N'Production.BillOfMaterials'))

DROP INDEX FIBillOfMaterialsWithEndDate

    ON Production.BillOfMaterials

GO

-- Creates a filtered index "FIBillOfMaterialsWithEndDate"

-- on the table Production.BillOfMaterials

-- using the columms ComponentID and StartDate.

CREATE NONCLUSTERED INDEX FIBillOfMaterialsWithEndDate

    ON Production.BillOfMaterials (ComponentID, StartDate)

    WHERE EndDate IS NOT NULL ;

GO

The filtered index above is valid for the following query. You can display the query execution plan to determine if the query optimizer used the filtered index.

USE AdventureWorks2012;

GO

SELECT ProductAssemblyID, ComponentID, StartDate

FROM Production.BillOfMaterials

WHERE EndDate IS NOT NULL

    AND ComponentID = 5

    AND StartDate > '01/01/2008' ;

GO

**To ensure that a filtered index is used in a SQL query**

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

USE AdventureWorks2012;

GO

SELECT ComponentID, StartDate FROM Production.BillOfMaterials

WITH ( INDEX ( FIBillOfMaterialsWithEndDate ) )

WHERE EndDate IN ('20000825', '20000908', '20000918');

GO