**CREATE VIEW (Transact-SQL)**

Creates a virtual table that represents the data in one or more tables in an alternative way. CREATE VIEW must be the first statement in a query batch.

Topic link icon[Transact-SQL Syntax Conventions](http://technet.microsoft.com/en-us/library/ms177563.aspx)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

CREATE VIEW [ schema\_name . ] view\_name [ (column [ ,...n ] ) ]

[ WITH <view\_attribute> [ ,...n ] ]

AS select\_statement

[ WITH CHECK OPTION ] [ ; ]

<view\_attribute> ::=

{

[ ENCRYPTION ]

    [ SCHEMABINDING ]

    [ VIEW\_METADATA ]     }

http://i.msdn.microsoft.com/Global/Images/clear.gif Arguments

*schema\_name*

Is the name of the schema to which the view belongs.

*view\_name*

Is the name of the view. View names must follow the rules for identifiers. Specifying the view owner name is optional.

*column*

Is the name to be used for a column in a view. A column name is required only when a column is derived from an arithmetic expression, a function, or a constant; when two or more columns may otherwise have the same name, typically because of a join; or when a column in a view is specified a name different from that of the column from which it is derived. Column names can also be assigned in the SELECT statement.

If *column* is not specified, the view columns acquire the same names as the columns in the SELECT statement.

|  |
| --- |
| **ms187956.note(en-us,SQL.100).gifNote:** |
| In the columns for the view, the permissions for a column name apply across a CREATE VIEW or ALTER VIEW statement, regardless of the source of the underlying data. For example, if permissions are granted on the **SalesOrderID** column in a CREATE VIEW statement, an ALTER VIEW statement can name the **SalesOrderID** column with a different column name, such as **OrderRef**, and still have the permissions associated with the view using **SalesOrderID**. |

AS

Specifies the actions the view is to perform.

*select\_statement*

Is the SELECT statement that defines the view. The statement can use more than one table and other views. Appropriate permissions are required to select from the objects referenced in the SELECT clause of the view that is created.

A view does not have to be a simple subset of the rows and columns of one particular table. A view can be created that uses more than one table or other views with a SELECT clause of any complexity.

In an indexed view definition, the SELECT statement must be a single table statement or a multitable JOIN with optional aggregation.

The SELECT clauses in a view definition cannot include the following:

* COMPUTE or COMPUTE BY clauses
* An ORDER BY clause, unless there is also a TOP clause in the select list of the SELECT statement

|  |
| --- |
| **ms187956.note(en-us,SQL.100).gifNote:** |
| The ORDER BY clause is used only to determine the rows that are returned by the TOP clause in the view definition. The ORDER BY clause does not guarantee ordered results when the view is queried, unless ORDER BY is also specified in the query itself. |

* The INTO keyword
* The OPTION clause
* A reference to a temporary table or a table variable.

Because *select\_statement* uses the SELECT statement, it is valid to use <join\_hint> and <table\_hint> hints as specified in the FROM clause. For more information, see [FROM (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms177634.aspx) and [SELECT (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms189499.aspx).

Functions and multiple SELECT statements separated by UNION or UNION ALL can be used in *select\_statement*.

CHECK OPTION

Forces all data modification statements executed against the view to follow the criteria set within *select\_statement*. When a row is modified through a view, the WITH CHECK OPTION makes sure the data remains visible through the view after the modification is committed.

|  |
| --- |
| **ms187956.note(en-us,SQL.100).gifNote:** |
| Any updates performed directly to a view's underlying tables are not verified against the view, even if CHECK OPTION is specified. |

ENCRYPTION

Encrypts the entries in [sys.syscomments](http://technet.microsoft.com/en-us/library/ms186293.aspx) that contain the text of the CREATE VIEW statement. Using WITH ENCRYPTION prevents the view from being published as part of SQL Server replication.

SCHEMABINDING

Binds the view to the schema of the underlying table or tables. When SCHEMABINDING is specified, the base table or tables cannot be modified in a way that would affect the view definition. The view definition itself must first be modified or dropped to remove dependencies on the table that is to be modified. When you use SCHEMABINDING, the *select\_statement* must include the two-part names (*schema***.***object*) of tables, views, or user-defined functions that are referenced. All referenced objects must be in the same database.

Views or tables that participate in a view created with the SCHEMABINDING clause cannot be dropped unless that view is dropped or changed so that it no longer has schema binding. Otherwise, the Database Engine raises an error. Also, executing ALTER TABLE statements on tables that participate in views that have schema binding fail when these statements affect the view definition.

SCHEMABINDING cannot be specified if the view contains alias data type columns.

VIEW\_METADATA

Specifies that the instance of SQL Server will return to the DB-Library, ODBC, and OLE DB APIs the metadata information about the view, instead of the base table or tables, when browse-mode metadata is being requested for a query that references the view. Browse-mode metadata is additional metadata that the instance of SQL Server returns to these client-side APIs. This metadata enables the client-side APIs to implement updatable client-side cursors. Browse-mode metadata includes information about the base table that the columns in the result set belong to.

For views created with VIEW\_METADATA, the browse-mode metadata returns the view name and not the base table names when it describes columns from the view in the result set.

When a view is created by using WITH VIEW\_METADATA, all its columns, except a **timestamp** column, are updatable if the view has INSTEAD OF INSERT or INSTEAD OF UPDATE triggers. For more information about updatable views, see Remarks.

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

A view can be created only in the current database. A view can have a maximum of 1,024 columns.

When querying through a view, the Database Engine checks to make sure that all the database objects referenced anywhere in the statement exist and that they are valid in the context of the statement, and that data modification statements do not violate any data integrity rules. A check that fails returns an error message. A successful check translates the action into an action against the underlying table or tables.

If a view depends on a table or view that was dropped, the Database Engine produces an error message when anyone tries to use the view. If a new table or view is created and the table structure does not change from the previous base table to replace the one dropped, the view again becomes usable. If the new table or view structure changes, the view must be dropped and re-created.

If a view is not created with the SCHEMABINDING clause, [sp\_refreshview](http://technet.microsoft.com/en-us/library/ms187821.aspx) should be run when changes are made to the objects underlying the view that affect the definition of the view. Otherwise, the view might produce unexpected results when it is queried.

When a view is created, information about the view is stored in the following catalog views: [sys.views](http://technet.microsoft.com/en-us/library/ms190334.aspx), [sys.columns](http://technet.microsoft.com/en-us/library/ms176106.aspx), and [sys.sql\_expression\_dependencies](http://technet.microsoft.com/en-us/library/bb677315.aspx). The text of the CREATE VIEW statement is stored in the [sys.sql\_modules](http://technet.microsoft.com/en-us/library/ms175081.aspx) catalog view.

A query that uses an index on a view defined with **numeric** or **float** expressions may have a result that is different from a similar query that does not use the index on the view. This difference may be caused by rounding errors during INSERT, DELETE, or UPDATE actions on underlying tables.

The Database Engine saves the settings of SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS when a view is created. These original settings are used to parse the view when the view is used. Therefore, any client-session settings for SET QUOTED\_IDENTIFIER and SET ANSI\_NULLS do not affect the view definition when the view is accessed.

**Updatable Views**

You can modify the data of an underlying base table through a view, as long as the following conditions are true:

* Any modifications, including UPDATE, INSERT, and DELETE statements, must reference columns from only one base table.
* The columns being modified in the view must directly reference the underlying data in the table columns. The columns cannot be derived in any other way, such as through the following:
  + An aggregate function: AVG, COUNT, SUM, MIN, MAX, GROUPING, STDEV, STDEVP, VAR, and VARP.
  + A computation. The column cannot be computed from an expression that uses other columns. Columns that are formed by using the set operators UNION, UNION ALL, CROSSJOIN, EXCEPT, and INTERSECT amount to a computation and are also not updatable.
* The columns being modified are not affected by GROUP BY, HAVING, or DISTINCT clauses.
* TOP is not used anywhere in the *select\_statement* of the view together with the WITH CHECK OPTION clause.

The previous restrictions apply to any subqueries in the FROM clause of the view, just as they apply to the view itself. Generally, the Database Engine must be able to unambiguously trace modifications from the view definition to one base table. For more information, see [Modifying Data Through a View](http://technet.microsoft.com/en-us/library/ms180800.aspx).

If the previous restrictions prevent you from modifying data directly through a view, consider the following options:

* **INSTEAD OF Triggers**   
  INSTEAD OF triggers can be created on a view to make a view updatable. The INSTEAD OF trigger is executed instead of the data modification statement on which the trigger is defined. This trigger lets the user specify the set of actions that must happen to process the data modification statement. Therefore, if an INSTEAD OF trigger exists for a view on a specific data modification statement (INSERT, UPDATE, or DELETE), the corresponding view is updatable through that statement. For more information about INSTEAD OF triggers, see [Designing INSTEAD OF Triggers](http://technet.microsoft.com/en-us/library/ms175521.aspx).
* **Partitioned Views**   
  If the view is a partitioned view, the view is updatable, subject to certain restrictions. When it is needed, the Database Engine distinguishes local partitioned views as the views in which all participating tables and the view are on the same instance of SQL Server, and distributed partitioned views as the views in which at least one of the tables in the view resides on a different or remote server.   
  For more information about partitioned views, see [Creating Partitioned Views](http://technet.microsoft.com/en-us/library/ms181036.aspx).

**Partitioned Views**

A partitioned view is a view defined by a UNION ALL of member tables structured in the same way, but stored separately as multiple tables in either the same instance of SQL Server or in a group of autonomous instances of SQL Server servers, called federated database servers.

|  |
| --- |
| **ms187956.note(en-us,SQL.100).gifNote:** |
| The preferred method for partitioning data local to one server is through partitioned tables. For more information, see [Partitioned Tables and Indexes](http://technet.microsoft.com/en-us/library/ms188706.aspx). |

In designing a partitioning scheme, it must be clear what data belongs to each partition. For example, the data for the Customers table is distributed in three member tables in three server locations: Customers\_33 on Server1, Customers\_66 on Server2, and Customers\_99 on Server3.

A partitioned view on Server1 is defined in the following way:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl63_ctl00_ctl10_code');" \o "Copy Code)

--Partitioned view as defined on Server1

CREATE VIEW Customers

AS

--Select from local member table.

SELECT \*

FROM CompanyData.dbo.Customers\_33

UNION ALL

--Select from member table on Server2.

SELECT \*

FROM Server2.CompanyData.dbo.Customers\_66

UNION ALL

--Select from mmeber table on Server3.

SELECT \*

FROM Server3.CompanyData.dbo.Customers\_99

Generally, a view is said to be a partitioned view if it is of the following form:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl63_ctl00_ctl11_code');" \o "Copy Code)

SELECT <select\_list1>

FROM T1

UNION ALL

SELECT <select\_list2>

FROM T2

UNION ALL

...

SELECT <select\_listn>

FROM Tn

**Conditions for Creating Partitioned Views**

1. The select list
   * All columns in the member tables should be selected in the column list of the view definition.
   * The columns in the same ordinal position of each select list should be of the same type, including collations. It is not sufficient for the columns to be implicitly convertible types, as is generally the case for UNION.   
     Also, at least one column (for example <col>) must appear in all the select lists in the same ordinal position. This <col> should be defined in a way that the member tables T1, ..., Tn have CHECK constraints C1, ..., Cn defined on <col>, respectively.   
     Constraint C1 defined on table T1 must be of the following form:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl63_ctl00_ctl12_code');" \o "Copy Code)

C1 ::= < simple\_interval > [ OR < simple\_interval > OR ...]

< simple\_interval > :: =

< col > { < | > | <= | >= | = < value >}

| < col > BETWEEN < value1 > AND < value2 >

| < col > IN ( value\_list )

| < col > { > | >= } < value1 > AND

< col > { < | <= } < value2 >

* + The constraints should be in such a way that any specified value of <col> can satisfy, at most, one of the constraints C1, ..., Cn so that the constraints should form a set of disjointed or nonoverlapping intervals. The column <col> on which the disjointed constraints are defined is called the partitioning column. Note that the partitioning column may have different names in the underlying tables. The constraints should be in an enabled and trusted state for them to meet the previously mentioned conditions of the partitioning column. If the constraints are disabled, re-enable constraint checking by using the CHECK CONSTRAINT *constraint\_name* option of ALTER TABLE, and using the WITH CHECK option to validate them.   
    The following examples show valid sets of constraints:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl63_ctl00_ctl13_code');" \o "Copy Code)

{ [col < 10], [col between 11 and 20] , [col > 20] }

{ [col between 11 and 20], [col between 21 and 30], [col between 31 and 100] }

* + The same column cannot be used multiple times in the select list.

1. Partitioning column
   * The partitioning column is a part of the PRIMARY KEY of the table.
   * It cannot be a computed, identity, default, or **timestamp** column.
   * If there is more than one constraint on the same column in a member table, the Database Engine ignores all the constraints and does not consider them when determining whether the view is a partitioned view. To meet the conditions of the partitioned view, there should be only one partitioning constraint on the partitioning column.
   * There are no restrictions on the updatability of the partitioning column.
2. Member tables, or underlying tables T1, ..., Tn
   * The tables can be either local tables or tables from other computers that are running SQL Server that are referenced either through a four-part name or an OPENDATASOURCE- or OPENROWSET-based name. The OPENDATASOURCE and OPENROWSET syntax can specify a table name, but not a pass-through query. For more information, see [OPENDATASOURCE (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms179856.aspx) and [OPENROWSET (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms190312.aspx).   
     If one or more of the member tables are remote, the view is called **distributed partitioned view**, and additional conditions apply. They are described later in this section.
   * The same table cannot appear two times in the set of tables that are being combined with the UNION ALL statement.
   * The member tables cannot have indexes created on computed columns in the table.
   * The member tables should have all PRIMARY KEY constraints on the same number of columns.
   * All member tables in the view should have the same ANSI padding setting. This can be set by using either the **user options** option in **sp\_configure** or the SET statement.

**Conditions for Modifying Data in Partitioned Views**

The following restrictions apply to statements that modify data in partitioned views:

* The INSERT statement must supply values for all the columns in the view, even if the underlying member tables have a DEFAULT constraint for those columns or if they allow for null values. For those member table columns that have DEFAULT definitions, the statements cannot explicitly use the keyword DEFAULT.
* The value being inserted into the partitioning column should satisfy at least one of the underlying constraints; otherwise, the insert action will fail with a constraint violation.
* UPDATE statements cannot specify the DEFAULT keyword as a value in the SET clause, even if the column has a DEFAULT value defined in the corresponding member table.
* PRIMARY KEY columns cannot be modified through an UPDATE statement if the member tables have **text**, **ntext**, or **image** columns.
* Columns in the view that are an identity column in one or more of the member tables cannot be modified by using an INSERT or UPDATE statement.
* If one of the member tables contains a **timestamp** column, the data cannot be modified by using an INSERT or UPDATE statement.
* If one of the member tables contains a trigger or an ON UPDATE CASCADE/SET NULL/SET DEFAULT or ON DELETE CASCADE/SET NULL/SET DEFAULT constraint, the view cannot be modified.
* INSERT, UPDATE, and DELETE actions against a partitioned view are not allowed if there is a self-join with the same view or with any of the member tables in the statement.
* Bulk importing data into a partitioned view is unsupported by **bcp** or the BULK INSERT and INSERT ... SELECT \* FROM OPENROWSET(BULK...) statements. However, you can insert multiple rows into a partitioned view by using the [INSERT](http://technet.microsoft.com/en-us/library/ms174335.aspx) statement. For more information, see [Bulk Exporting Data from or Bulk Importing Data to a View](http://technet.microsoft.com/en-us/library/ms187086.aspx).

|  |
| --- |
| **ms187956.note(en-us,SQL.100).gifNote:** |
| To update a partitioned view, the user must have INSERT, UPDATE, and DELETE permissions on the member tables. |

**Additional Conditions for Distributed Partitioned Views**

For distributed partitioned views (when one or more member tables are remote), the following additional conditions apply:

* A distributed transaction will be started to guarantee atomicity across all nodes affected by the update.
* The XACT\_ABORT SET option should be set to ON for INSERT, UPDATE, or DELETE statements to work.
* Any **smallmoney** and **smalldatetime** columns in remote tables that are referenced in a partitioned view are mapped as **money** and **datetime**, respectively. Therefore, the corresponding columns (in the same ordinal position in the select list) in the local tables should be **money** and **datetime**.
* Any linked server in the partitioned view cannot be a loopback linked server. This is a linked server that points to the same instance of SQL Server.

The setting of the SET ROWCOUNT option is ignored for INSERT, UPDATE, and DELETE actions that involve updatable partitioned views and remote tables.

When the member tables and partitioned view definition are in place, the SQL Server query optimizer builds intelligent plans that use queries efficiently to access data from member tables. With the CHECK constraint definitions, the query processor maps the distribution of key values across the member tables. When a user issues a query, the query processor compares the map to the values specified in the WHERE clause, and builds an execution plan with a minimal amount of data transfer between member servers. Therefore, although some member tables may be located in remote servers, the instance of SQL Server resolves distributed queries so that the amount of distributed data that has to be transferred is minimal. For more information about how SQL Server resolves queries on partitioned views, see [Resolving Distributed Partitioned Views](http://technet.microsoft.com/en-us/library/ms187836.aspx).

**Considerations for Replication**

To create partitioned views on member tables that are involved in replication, the following considerations apply:

* If the underlying tables are involved in merge replication or transactional replication with updating subscriptions, the **uniqueidentifier** column should also be included in the select list.  
  Any INSERT actions into the partitioned view must provide a NEWID() value for the **uniqueidentifier** column. Any UPDATE actions against the **uniqueidentifier** column must supply NEWID() as the value because the DEFAULT keyword cannot be used.
* The replication of updates made by using the view is the same as when tables are replicated in two different databases: the tables are served by different replication agents and the order of the updates is not guaranteed.

http://i.msdn.microsoft.com/Global/Images/clear.gif Permissions

Requires CREATE VIEW permission in the database and ALTER permission on the schema in which the view is being created.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

**A. Using a simple CREATE VIEW**

The following example creates a view by using a simple SELECT statement. A simple view is helpful when a combination of columns is queried frequently. The data from this view comes from the HumanResources.Employee and Person.Contact tables of the AdventureWorks database. The data provides name and hire date information for the employees of Adventure Works Cycles. The view could be created for the person in charge of tracking work anniversaries but without giving this person access to all the data in these tables.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl65_ctl00_ctl00_code');" \o "Copy Code)

USE AdventureWorks ;

GO

IF OBJECT\_ID ('hiredate\_view', 'V') IS NOT NULL

DROP VIEW hiredate\_view ;

GO

CREATE VIEW hiredate\_view

AS

SELECT c.FirstName, c.LastName, e.EmployeeID, e.HireDate

FROM HumanResources.Employee e JOIN Person.Contact c on e.ContactID = c.ContactID ;

GO

**B. Using WITH ENCRYPTION**

The following example uses the WITH ENCRYPTION option and shows computed columns, renamed columns, and multiple columns.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl65_ctl00_ctl01_code');" \o "Copy Code)

USE AdventureWorks ;

GO

IF OBJECT\_ID ('Purchasing.PurchaseOrderReject', 'V') IS NOT NULL

DROP VIEW Purchasing.PurchaseOrderReject ;

GO

CREATE VIEW Purchasing.PurchaseOrderReject

WITH ENCRYPTION

AS

SELECT PurchaseOrderID, ReceivedQty, RejectedQty,

RejectedQty / ReceivedQty AS RejectRatio, DueDate

FROM Purchasing.PurchaseOrderDetail

WHERE RejectedQty / ReceivedQty > 0

AND DueDate > CONVERT(DATETIME,'20010630',101) ;

GO

**C. Using WITH CHECK OPTION**

The following example shows a view named SeattleOnly that references five tables and allows for data modifications to apply only to employees who live in Seattle.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl65_ctl00_ctl02_code');" \o "Copy Code)

USE AdventureWorks ;

GO

IF OBJECT\_ID ('dbo.SeattleOnly', 'V') IS NOT NULL

DROP VIEW dbo.SeattleOnly ;

GO

CREATE VIEW dbo.SeattleOnly

AS

SELECT c.LastName, c.FirstName, a.City, s.StateProvinceCode

FROM Person.Contact AS c

JOIN HumanResources.Employee AS e ON c.ContactID = e.ContactID

JOIN HumanResources.EmployeeAddress AS ea ON e.EmployeeID = ea.EmployeeID

JOIN Person.Address AS a ON ea.AddressID = a.AddressID

JOIN Person.StateProvince AS s ON a.StateProvinceID = s.StateProvinceID

WHERE a.City = 'Seattle'

WITH CHECK OPTION ;

GO

**D. Using built-in functions within a view**

The following example shows a view definition that includes a built-in function. When you use functions, you must specify a column name for the derived column.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl65_ctl00_ctl03_code');" \o "Copy Code)

USE AdventureWorks ;

GO

IF OBJECT\_ID ('Sales.SalesPersonPerform', 'V') IS NOT NULL

DROP VIEW Sales.SalesPersonPerform ;

GO

CREATE VIEW Sales.SalesPersonPerform

AS

SELECT TOP 100 SalesPersonID, SUM(TotalDue) AS TotalSales

FROM Sales.SalesOrderHeader

WHERE OrderDate > CONVERT(DATETIME,'20001231',101)

GROUP BY SalesPersonID;

GO

**E. Using partitioned data**

The following example uses tables named SUPPLY1, SUPPLY2, SUPPLY3, and SUPPLY4. These tables correspond to the supplier tables from four offices, located in different countries/regions.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl65_ctl00_ctl04_code');" \o "Copy Code)

--Create the tables and insert the values.

CREATE TABLE dbo.SUPPLY1 (

supplyID INT PRIMARY KEY CHECK (supplyID BETWEEN 1 and 150),

supplier CHAR(50)

);

CREATE TABLE dbo.SUPPLY2 (

supplyID INT PRIMARY KEY CHECK (supplyID BETWEEN 151 and 300),

supplier CHAR(50)

);

CREATE TABLE dbo.SUPPLY3 (

supplyID INT PRIMARY KEY CHECK (supplyID BETWEEN 301 and 450),

supplier CHAR(50)

);

CREATE TABLE dbo.SUPPLY4 (

supplyID INT PRIMARY KEY CHECK (supplyID BETWEEN 451 and 600),

supplier CHAR(50)

);

GO

INSERT dbo.SUPPLY1 VALUES ('1', 'CaliforniaCorp');

INSERT dbo.SUPPLY1 VALUES ('5', 'BraziliaLtd');

INSERT dbo.SUPPLY2 VALUES ('231', 'FarEast');

INSERT dbo.SUPPLY2 VALUES ('280', 'NZ');

INSERT dbo.SUPPLY3 VALUES ('321', 'EuroGroup');

INSERT dbo.SUPPLY3 VALUES ('442', 'UKArchip');

INSERT dbo.SUPPLY4 VALUES ('475', 'India');

INSERT dbo.SUPPLY4 VALUES ('521', 'Afrique');

GO

--Create the view that combines all supplier tables.

CREATE VIEW all\_supplier\_view

WITH SCHEMABINDING

AS

SELECT supplyID, supplier

FROM dbo.SUPPLY1

UNION ALL

SELECT supplyID, supplier

FROM dbo.SUPPLY2

UNION ALL

SELECT supplyID, supplier

FROM dbo.SUPPLY3

UNION ALL

SELECT supplyID, supplier

FROM dbo.SUPPLY4;

**ALTER VIEW (Transact-SQL)**

Modifies a previously created view. This includes an indexed view. ALTER VIEW does not affect dependent stored procedures or triggers and does not change permissions. For more information about the parameters used in the ALTER VIEW statement, see [CREATE VIEW (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms187956.aspx).

Topic link icon[Transact-SQL Syntax Conventions](http://technet.microsoft.com/en-us/library/ms177563.aspx)

 Syntax

ALTER VIEW [ schema\_name . ] view\_name [ ( column [ ,...n ] ) ]

[ WITH <view\_attribute> [ ,...n ] ]

AS select\_statement

[ WITH CHECK OPTION ] [ ; ]

<view\_attribute> ::=

{

[ ENCRYPTION ]

    [ SCHEMABINDING ]

    [ VIEW\_METADATA ]

}

 Arguments

*schema\_name*

Is the name of the schema to which the view belongs.

*view\_name*

Is the view to change.

*column*

Is the name of one or more columns, separated by commas, that are to be part of the specified view.

|  |
| --- |
| **Important:** |
| Column permissions are maintained only when columns have the same name before and after ALTER VIEW is performed. |
| **Note:** |
| In the columns for the view, the permissions for a column name apply across a CREATE VIEW or ALTER VIEW statement, regardless of the source of the underlying data. For example, if permissions are granted on the **SalesOrderID** column in a CREATE VIEW statement, an ALTER VIEW statement can rename the **SalesOrderID** column, such as to **OrderRef**, and still have the permissions associated with the view using **SalesOrderID**. |

ENCRYPTION

Encrypts the entries in [sys.syscomments](http://technet.microsoft.com/en-us/library/ms186293.aspx) that contain the text of the ALTER VIEW statement. WITH ENCRYPTION prevents the view from being published as part of SQL Server replication.

SCHEMABINDING

Binds the view to the schema of the underlying table or tables. When SCHEMABINDING is specified, the base tables cannot be modified in a way that would affect the view definition. The view definition itself must first be modified or dropped to remove dependencies on the table to be modified. When you use SCHEMABINDING, the *select\_statement* must include the two-part names (*schema***.***object*) of tables, views, or user-defined functions that are referenced. All referenced objects must be in the same database.

Views or tables that participate in a view created with the SCHEMABINDING clause cannot be dropped, unless that view is dropped or changed so that it no longer has schema binding. Otherwise, the Database Engine raises an error. Also, executing ALTER TABLE statements on tables that participate in views that have schema binding fail if these statements affect the view definition.

SCHEMABINDING cannot be specified if the view contains alias data type columns.

VIEW\_METADATA

Specifies that the instance of SQL Server will return to the DB-Library, ODBC, and OLE DB APIs the metadata information about the view, instead of the base table or tables, when browse-mode metadata is being requested for a query that references the view. Browse-mode metadata is additional metadata that the instance of Database Engine returns to the client-side DB-Library, ODBC, and OLE DB APIs. This metadata enables the client-side APIs to implement updatable client-side cursors. Browse-mode metadata includes information about the base table that the columns in the result set belong to.

For views created with VIEW\_METADATA, the browse-mode metadata returns the view name and not the base table names when it describes columns from the view in the result set.

When a view is created by using WITH VIEW\_METADATA, all its columns, except a **timestamp** column, are updatable if the view has INSERT or UPDATE INSTEAD OF triggers. For more information, see the Remarks section in [CREATE VIEW (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms187956.aspx).

AS

Are the actions the view is to take.

*select\_statement*

Is the SELECT statement that defines the view.

WITH CHECK OPTION

Forces all data modification statements that are executed against the view to follow the criteria set within *select\_statement*.

 Remarks

For more information about ALTER VIEW, see Remarks in [CREATE VIEW (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms187956.aspx).

|  |
| --- |
| **Note:** |
| If the previous view definition was created by using WITH ENCRYPTION or CHECK OPTION, these options are enabled only if they are included in ALTER VIEW. |

If a view currently used is modified by using ALTER VIEW, the Database Engine takes an exclusive schema lock on the view. When the lock is granted, and there are no active users of the view, the Database Engine deletes all copies of the view from the procedure cache. Existing plans referencing the view remain in the cache but are recompiled when invoked.

ALTER VIEW can be applied to indexed views; however, ALTER VIEW unconditionally drops all indexes on the view.

 Permissions

To execute ALTER VIEW, at a minimum, ALTER permission on OBJECT is required.

 Examples

The following example creates a view that contains all employees and their hire dates called EmployeeHireDate. Permissions are granted to the view, but requirements are changed to select employees whose hire dates fall before a certain date. Then, ALTER VIEW is used to replace the view.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl00_code');" \o "Copy Code)

USE AdventureWorks ;

GO

CREATE VIEW HumanResources.EmployeeHireDate

AS

SELECT c.FirstName, c.LastName, e.HireDate

FROM HumanResources.Employee AS e JOIN Person.Contact AS c

ON e.ContactID = c.ContactID ;

GO

The view must be changed to include only the employees that were hired before 1997. If ALTER VIEW is not used, but instead the view is dropped and re-created, the previously used GRANT statement and any other statements that deal with permissions pertaining to this view must be re-entered.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl34_ctl00_ctl01_code');" \o "Copy Code)

ALTER VIEW HumanResources.EmployeeHireDate

AS

SELECT c.FirstName, c.LastName, e.HireDate

FROM HumanResources.Employee AS e JOIN Person.Contact AS c

ON e.ContactID = c.ContactID

WHERE HireDate < CONVERT(DATETIME,'19980101',101) ;

GO

**DROP VIEW (Transact-SQL)**

Removes one or more views from the current database. DROP VIEW can be executed against indexed views.

Topic link icon[Transact-SQL Syntax Conventions](http://technet.microsoft.com/en-us/library/ms177563.aspx)

 Syntax

DROP VIEW [ schema\_name . ] view\_name [ ...,n ] [ ; ]

 Arguments

*schema\_name*

Is the name of the schema to which the view belongs.

*view\_name*

Is the name of the view to remove.

 Remarks

When you drop a view, the definition of the view and other information about the view is deleted from the system catalog. All permissions for the view are also deleted.

Any view on a table that is dropped by using DROP TABLE must be dropped explicitly by using DROP VIEW.

When executed against an indexed view, DROP VIEW automatically drops all indexes on a view. To display all indexes on a view, use [sp\_helpindex](http://technet.microsoft.com/en-us/library/ms188771.aspx).

When querying through a view, the Database Engine checks to make sure that all the database objects referenced in the statement exist and that they are valid in the context of the statement, and that data modification statements do not violate any data integrity rules. A check that fails returns an error message. A successful check translates the action into an action against the underlying table or tables. If the underlying tables or views have changed since the view was originally created, it may be useful to drop and re-create the view.

For more information about determining dependencies for a specific view, see [sys.sql\_dependencies (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms174402.aspx).

For more information about viewing the text of the view, see [sp\_helptext (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms176112.aspx).

 Permissions

To execute DROP VIEW, at a minimum, ALTER permission on SCHEMA or CONTROL permission on OBJECT is required.

 Examples

The following example removes the view Reorder.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl27_ctl00_ctl00_code');" \o "Copy Code)

USE AdventureWorks ;

GO

IF OBJECT\_ID ('dbo.Reorder', 'V') IS NOT NULL

DROP VIEW dbo.Reorder ;

GO